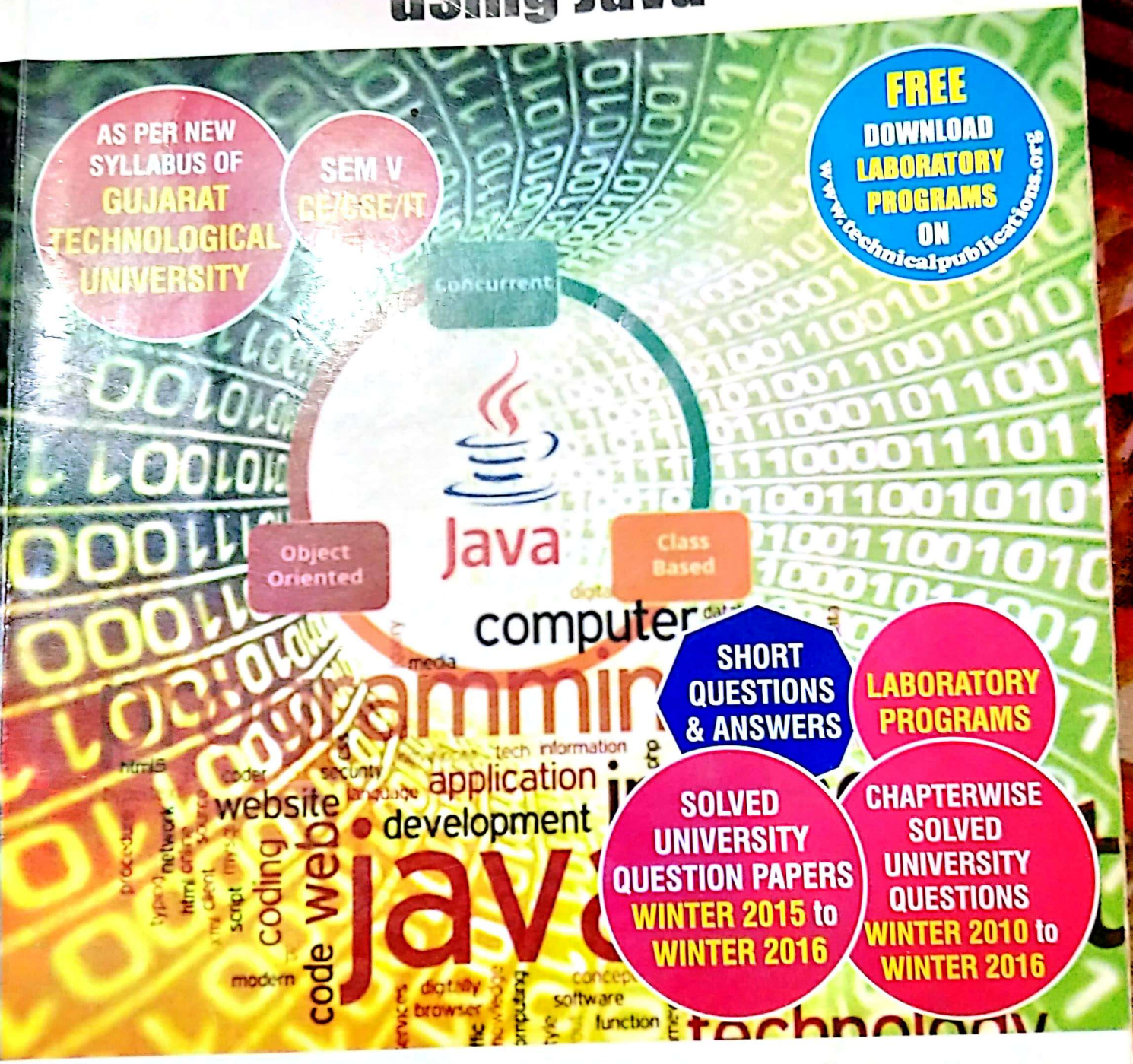
Object Oriented Programming using Java



TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

A. A. Puntambekar

As per New Syllabus of

GUJARAT TECHNOLOGICAL UNIVERSITY

OBJECT ORIENTED PROGRAMMING USING JAVA

Mrs. Anuradha A. Puntambekar

M.E. (Computer)
Formerly Assistant Professor in
P.E.S. Modern College of Engineering, Pune

Price : ₹ 395/-

ISBN 978-93-332-0381-4

TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

OBJECT ORIENTED PROGRAMMING USING JAVA

Semester - V (CE / CSE / IT)

First printed in India : January 2013
First Edition : July 2015 (As per New Syllabus of Gujarat Technological University)
Second Revised Edition : June 2016
Third Revised Edition : July 2016
Fourth Revised Edition : June 2017

Published by :

TECHNICAL PUBLICATIONS
Amit Residency, Office No.1, 412, Shaniwar Peth, Pune - 411030, M.S. INDIA.
Ph.: +91-020-24495496/97, Telefax : +91-020-24495497
Email : technical@vtubooks.com Website : www.technicalpublications.org

Printer :

Rachana Offset
S.no 15, Sukhsagar Nagar,
Waghjai Chowk, Arihant Marg, Katraj,
Pune - 411 046

Price : ₹ 395/-

ISBN 978-93-332-0381-4

9789333203814 [4]

PREF

The importa
various enginee
inspired me to
subject **Object**

The book
book provides
methods to e
illustrations.
arranged in
care has bee
of the subje

Represe
students in

The bo
of the su
more inte
subject te
more.

I wis
reality.
occasio
Techni
quality

Ar
appre

PREFACE

The importance of **Object Oriented Programming using Java** is well known in various engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject **Object Oriented Programming using Java**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Author
A. A. Puntambekar

Dedicated to God.

SYLLABUS

Object Oriented Programming using Java (2150704)

1. **Basics of Java : (Chapter - 1)**

Features of Java, Byte code and Java virtual machine, JDK, Data types, Operator, Control statements - if else, nested if, if-else ladders, switch, while, do-while, for, for-each, break, continue.

2. **Array and String : (Chapter - 2)**

Single and multidimensional array, String class, StringBuffer class, Operations on string, Command line argument, Use of wrapper class.

3. **Classes, Objects and Methods : (Chapter - 3)**

Class, Object, Object reference, Constructor, Constructor overloading, Method overloading, Recursion, Passing and returning object form method, New operator, this and static keyword, finalize() method, Access control, Modifiers, Nested class, Inner class, Anonymous inner class, Abstract class.

4. **Inheritance and Interfaces : (Chapter - 4)**

Use of inheritance, Inheriting data members and methods, Constructor in inheritance, Multilevel inheritance - Method overriding handle multilevel constructors - Super keyword,Stop inheritance - Final keywords, Creation and implementation of an interface, Interface reference, Instance of operator, Interface inheritance, Dynamic method dispatch, Understanding of Java object class,Comparison between abstract class and interface, Understanding of system.out.println - statement.

5. **Package : (Chapter - 5)**

Use of package, CLASSPATH, Import statement, Static import, Access control.

6. **Exception Handling : (Chapter - 6)**

Exception and error, Use of try, Catch, Throw, Throws and finally, Built in exception, Custom exception, Throwable class.

7. **Multithreaded Programming : (Chapter - 7)**

Use of multithread programming, Thread class and runnable interface, Thread priority, Thread synchronization, Thread communication, Deadlock.

8. **IO Programming : (Chapter - 8)**

Introduction to stream, Byte stream, Character stream, Readers and writers, File class, File InputStream, File OutputStream, InputStreamReader, OutputStreamWriter, FileReader, FileWriter, Buffered Reader.

9. **Collection Classes : (Chapter - 9)**

List, AbstractList, ArrayList, LinkedList, Enumeration, Vector, Properties, Introduction to Java.util package.

8. IO Programming : (Chapter - 8)

Introduction to stream, Byte stream, Character stream, Readers and writers, File class, File InputStream, File OutputStream, InputStreamReader, OutputStreamWriter, FileReader, FileWriter, Buffered Reader.

9. Collection Classes : (Chapter - 9)

List, AbstractList, ArrayList, LinkedList, Enumeration, Vector, Properties, Introduction to Java.util package.

10. Networking with Java.net : (Chapter - 10)

InetAddress class,Socket class, DatagramSocket class, DatagramPacket class.

11. Introduction to OOM as Design Technique : (Chapter - 11)

Modeling concepts, Abstraction, The three models, Class model, State model and interaction model.

12. Class Modeling : (Chapter - 12)

Object and class concepts, Link and association, Generalization and inheritance.

13. Advanced Class Modeling : (Chapter - 13)

Advanced object and class concepts, Association ends, N-ary associations, Aggregation, Abstract classes, Multiple inheritance, Metadata, Constraints, Derived data, Packages.

14. State Modeling : (Chapter - 14)

Events, States, Transition and conditions, State diagram, State diagram behavior.

15. Interaction Modeling : (Chapter - 15)

Use case models, Sequence models, Activity models.

Table of Contents

Chapter - 1 Basics of Java (1 - 1) to (1 - 32)

1.1 Introduction ..... 1 - 2
1.2 Features of Java ..... 1 - 2
1.3 Comparison of Java with C and C++ ..... 1 - 4
1.4 Byte Code and Virtual Machine ..... 1 - 5
1.4.1 Java Virtual Machine(JVM) ..... 1 - 5
1.5 JDK ..... 1 - 6
1.6 Java Program Structure ..... 1 - 8
1.7 Writing First Java Program ..... 1 - 9
1.8 Java Tokens ..... 1 - 11
1.9 Variables ..... 1 - 13
1.10 Data Types ..... 1 - 14
1.11 Escape Sequences ..... 1 - 16
1.12 Operators ..... 1 - 17
1.12.1 Precedence Relation ..... 1 - 18
1.13 Control Statements ..... 1 - 21
1.14 University Questions with Answers ..... 1 - 31

Chapter - 2 Arrays and String (2 - 1) to (2 - 24)

2.1 Arrays ..... 2 - 2
2.1.1 Single Dimensional Array ..... 2 - 2
2.1.2 Two Dimensional Array ..... 2 - 5
2.2 String Class ..... 2 - 9
2.3 Operations on String ..... 2 - 10

2.3.2 Concatenating the Strings using + Operator ..... 2 - 11
2.3.3 Character Extraction ..... 2 - 12
2.3.4 String Comparison ..... 2 - 12
2.3.5 Searching for the Substring ..... 2 - 13
2.3.6 Replacing the Character from String ..... 2 - 14
2.3.7 Upper Case and Lower Case ..... 2 - 14
2.4 String Buffer Class ..... 2 - 16
2.5 Command Line Argument ..... 2 - 21
2.6 Use of Wrapper Class ..... 2 - 22
2.7 University Questions with Answers ..... 2 - 24

Chapter - 3 Classes, Objects and Methods (3 - 1) to (3 - 52)

3.1 Introduction ..... 3 - 2
3.2 Class ..... 3 - 2
3.2.1 Data Field Declaration ..... 3 - 2
3.2.2 Method Field Declaration ..... 3 - 3
3.3 Object ..... 3 - 3
3.4 Assessing Class Members ..... 3 - 4
3.5 Object Reference ..... 3 - 6
3.6 Constructor ..... 3 - 8
3.6.1 Properties of Constructor ..... 3 - 10
3.6.2 Constructor Overloading ..... 3 - 11
3.7 Method Overloading ..... 3 - 20
3.8 Recursion ..... 3 - 22
3.9 Passing and Returning Object from Method ..... 3 - 26
3.9.1 How to Pass Objects to the Method ? ..... 3 - 26
3.9.2 How to Return an Object from a Method ? ..... 3 - 27
3.10 New Operator ..... 3 - 28
3.11 this Keyword ..... 3 - 32

6.3.5 Using finally
6.3.6 Using throws
6.4 Built in Exception
6.4.1 Checked and Unchecked Exceptions
6.5 Custom Exception
6.6 Throwable Class
6.7 University Questions with Answers

Chapter - 7 Multithreaded Programming (7 - 1) to (7 -

7.1 Use of Multithread Programming
7.1.1 Difference between Multiple Processes and Multiple Threads
7.2 Thread States
7.2.1 Life Cycle Model
7.3 Thread Class and Runnable Interface
7.4 Extending Thread Class
7.5 Implementing Runnable Interface
7.6 Thread Priority
7.6.1 Thread Priorities
7.6.2 Daemon Thread
7.6.3 Handler for Uncaught Exception
7.7 Multi-threading
7.7.1 Creation of Multiple Threads by Extending the Thread Class
7.7.2 Creation of Multiple Threads by Implementing Runnable Interface
7.8 Thread Synchronization
7.9 Thread Communication
7.10 Deadlock
7.11 University Questions with Answers

Chapter - 8 Input Output Programming (8 - 1) to (8 - 26)

8.1 Introduction to Stream
8.2 Types of Streams

8.2.1 Byte Stream ..... 8 - 2
8.2.2 Character Stream ..... 8 - 3
8.3 Readers and Writers ..... 8 - 3
8.4 File Class ..... 8 - 5
8.4.1 Obtaining Properties of File ..... 8 - 6
8.5 InputStream and OutputStream ..... 8 - 10
8.6 File Input Stream and File Output Stream ..... 8 - 11
8.7 Input Stream Reader and Output Stream Writer ..... 8 - 13
8.8 FileReader and FileWriter ..... 8 - 15
8.9 Buffered Reader ..... 8 - 18
8.10 Programming Examples on I/O ..... 8 - 20

Chapter - 9 Collection Classes **(9 - 1) to (9 - 24)**

9.1 Introduction ..... 9 - 2
9.2 Commonly used Collection Classes ..... 9 - 3
9.3 List ..... 9 - 4
9.4 Abstract List ..... 9 - 4
9.5 ArrayList ..... 9 - 7
9.6 Linked List ..... 9 - 8
9.7 Enumeration ..... 9 - 17
9.8 Vector Properties ..... 9 - 18
9.9 Introduction to Java Util Package ..... 9 - 22
9.10 University Questions with Answers ..... 9 - 23

Chapter - 10 Networking with Java.net **(10 - 1) to (10 - 14)**

10.1 Introduction ..... 10 - 2
10.2 InetAddress Class ..... 10 - 3
10.3 Inet4Address and Inet6Address ..... 10 - 8
10.4 Socket Class ..... 10 - 9

10.5 Datagram Socket Class and Datagram ...

Chapter - 11 Introduction to OOMD Techniques (11 - 1) to (11 ...

11.1 Introduction ...
11.1.1 What is Object Orientation ? ...
11.2 Modeling Concepts ...
11.3 Modeling Design Technique ...
11.4 Abstraction ...
11.5 The Three Models ...
11.6 Class Model ...
11.7 State Model ...
11.8 Interaction Model ...
11.9 Relationship among the Models ...
11.10 University Questions with Answers ...

Chapter - 12 Class Modeling (12 - 1) to (12. ...

12.1 Introduction to Class Modeling ...
12.2 Object and Class Concepts ...
12.2.1 Objects ...
12.2.2 Classes ...
12.2.3 Class Diagrams ...
12.2.4 Values and Attributes ...
12.2.5 Operations and Methods ...
12.2.6 Summary of Class Notations ...
12.3 Link and Association Concepts ...
12.3.1 Links and Associations ...
12.3.2 Multiplicity ...
12.3.3 Association End Names ...
12.3.4 Ordering ...
12.3.5 Bags and Sequences ...
12.3.6 Association Classes ...

12.3.7 Qualified Associations ..... 12 - 11
12.4 Generalization and Inheritance ..... 12 - 11
12.4.1 Definition ..... 12 - 11
12.4.2 Use of Generalization ..... 12 - 11
12.4.3 Overriding Features ..... 12 - 15
12.5 University Questions with Answers ..... 12 - 16
12.6 Exercise ..... 12 - 17

Chapter - 13 Advanced Class Modeling (13 - 1) to (13 - 18)

13.1 Advanced Object and Class Concepts ..... 13 - 2
13.1.1 Enumerations ..... 13 - 2
13.1.2 Multiplicity ..... 13 - 2
13.1.3 Scope ..... 13 - 2
13.1.4 Visibility ..... 13 - 3
13.2 Association Ends ..... 13 - 3
13.3 N-ary Associations ..... 13 - 3
13.4 Aggregation ..... 13 - 5
13.4.1 Difference between Aggregation and Association ..... 13 - 6
13.4.2 Difference between Aggregation and Composition ..... 13 - 6
13.4.3 Propagation of Operations ..... 13 - 7
13.5 Abstract Classes ..... 13 - 7
13.6 Multiple Inheritance ..... 13 - 8
13.6.1 Accidental Multiple Inheritance ..... 13 - 9
13.6.2 Workarounds ..... 13 - 10
13.7 Metadata ..... 13 - 10
13.8 Constraints ..... 13 - 12
13.8.1 Constraints on Object ..... 13 - 13
13.8.2 Constraints on Generalization Sets ..... 13 - 13
13.8.3 Constraints on Links ..... 13 - 14
13.8.4 Use of Constraints ..... 13 - 15
13.9 Derived Data ..... 13 - 15
13.10 Packages ..... 13 - 16
13.11 University Questions with Answers ..... 13 - 16
13.12 Exercise ..... 13 - 18

13.11 University Questions with Answers ..... 13 -

Chapter - 14 State Modeling **(14 - 1) to (14 - 20)**

14.1 Introduction ..... 14 -
14.2 Events ..... 14 -
14.2.1 Signal Event ..... 14 -
14.2.2 Change Event ..... 14 -
14.2.3 Time Event ..... 14 -
14.3 States ..... 14 -
14.4 Transition and Conditions ..... 14 -
14.5 State Diagram ..... 14 -
14.5.1 One Shot State Diagram ..... 14 -
14.5.2 Summary of Basic Notations ..... 14 -
14.6 State Diagram Behavior ..... 14 - 10
14.6.1 Activity Effects ..... 14 - 10
14.6.2 Do-Activities ..... 14 - 11
14.6.3 Entry and Exit Activities ..... 14 - 11
14.6.4 Completion Transition ..... 14 - 12
14.6.5 Sending Signals ..... 14 - 13
14.6.6 Examples ..... 14 - 13
14.7 Nested States ..... 14 - 18
14.8 University Questions with Answers ..... 14 - 19

Chapter - 15 Interaction Modeling **(15 - 1) to (15 - 40)**

15.1 Introduction ..... 15 - 2
15.2 Use Case Model ..... 15 - 2
15.2.1 Actor ..... 15 - 2
15.2.2 Use Cases ..... 15 - 2
15.2.3 Use Case Diagram ..... 15 - 3
15.2.4 Guidelines for Use Case Models ..... 15 - 5
15.3 Sequence Model ..... 15 - 7
15.3.1 Scenario ..... 15 - 7
15 - 7

15.3.2 Sequence Diagram ..... 15 - 8
15.3.3 Guidelines for Sequence Models ..... 15 - 10
15.4 Activity Model ..... 15 - 12
15.4.1 Activities ..... 15 - 13
15.4.2 Branches ..... 15 - 13
15.4.3 Initiation and Termination ..... 15 - 14
15.4.4 Concurrent Activities ..... 15 - 14
15.4.5 Executable Activity Diagram ..... 15 - 15
15.4.6 Guideline for Activity Model ..... 15 - 16
15.5 Use Case Relationships ..... 15 - 18
15.5.1 Include Relationships ..... 15 - 18
15.5.2 Extend Relationship ..... 15 - 19
15.5.3 Generalization ..... 15 - 20
15.5.4 Combination of Use Case Relationships ..... 15 - 20
15.5.5 Guideline for Use Case Relationship ..... 15 - 21
15.5.6 Examples ..... 15 - 22
15.6 Procedural Sequence Models ..... 15 - 30
15.6.1 Sequence Diagrams with Passive Objects ..... 15 - 30
15.6.2 Sequence Diagram with Transient Objects ..... 15 - 30
15.6.3 Guidelines for Procedural Sequence Models ..... 15 - 32
15.7 Special Constructs for Activity Models ..... 15 - 33
15.7.1 Sending and Receiving Signals ..... 15 - 34
15.7.2 Swimlanes ..... 15 - 34
15.7.3 Object Flows ..... 15 - 35
15.7.4 Synchronization Bars ..... 15 - 36
15.7.5 Examples ..... 15 - 37
15.8 University Questions with Answers ..... 15 - 39

bject Oriented Programming using Java - Laboratory (L - 1) to (L - 28)

1 Basics of Java

Syllabus

Features of Java, Byte code and Java virtual machine, JDK, Data types, Operator, Control statements - if , else, nested if, if-else ladders, Switch, while, do-while, for, for-each, break, continue.

Contents

1.1 Introduction

1.2 Features of Java **May-12,Winter-13**

. . . . . **Summer-14** · · · · · · · · · · · · Marks 7

1.3 Comparison of Java with C and C++

1.4 Byte Code and Virtual Machine . . . . . **Summer-13,Winter-14** · · · · Marks 3

1.5 JDK

1.6 Java Program Structure

1.7 Writing First Java Program

1.8 Java Tokens

1.9 Variables

1.10 Data Types

1.11 Escape Sequences

1.12 Operators

1.13 Control Statements

1.14 University Questions with Answers

1.1 Introduction

- Java was invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to complete the first working version of Java.
- This language was initially called as **Oak** but was renamed as Java in 1995.
- Following table shows the evolution of Java language.

Period	Events took place
June -1991	The project for Java language was initiated by James Gosling, Patrick Naughton, Chris Warth Ed Frank and Mike Sheridan at Sun Microsystems, Inc.
1992	This language was officially known as Oak.
1995	Sun Microsystems released the first public implementation as Java 1.0
1998-1999	The Java 2.0 was released. In this version there was a support for various platforms such as J2EE, J2ME.
2006	For marketing purposes, Sun renamed to J2 versions as Java EE, Java ME and Java SE.
2007	Sun released Java as open source software.
2010	Oracle acquired Sun Microsystems.

1.2 Features of Java

GTU : May-12, Winter-13, Summer-14, Marks 7

Following are the features of Java which makes it as a revolutionary programming language-

1. Java can be compiled and interpreted

Normally programming languages can be either compiled or interpreted but Java is a language which can be compiled as well as interpreted. First, Java compiler translates the Java source program into a special code called **bytecode.** Then Java interpreter interprets this bytecode to obtain the equivalent machine code. This machine code is then directly executed to obtain the output.

2. Java is a platform independent and portable programming language

Platform independence is the most exciting feature of Java program. That means programs in Java can be executed on variety of platforms. This feature is based on the goal - *write once, run anywhere, and at anytime forever.*

Java supports portability in 2 ways - Java compiler generates the byte code which can be further used to obtain the corresponding machine code. Secondly the primitive data types used in Java are machine independent.

3. Java is known as an object oriented programming language

Java is a true object oriented language as everything in java is an object. In Java, all the code and data lies within the classes and object.

4. Java is robust and secure

Java ensures the reliable code. The data types are strictly checked at the compile time as well as run time in Java. The memory management technique is supported by garbage collection technique. This technique eliminates various memory management problems. The exception handling feature of Java helps the programmer to handle the serious errors delicately without crashing the overall system.

5. Java is a designed for distributed systems

This feature is very much useful in networking environment. In Java, two different objects on different computers can communicate with each other. This can be achieved by Remote Method Invocation(RMI). This feature is very much useful in Client-Server communication.

6. Java is simple and small programming language

Java is very simple programming language. Even though you have no programming background, you can learn this language very easily. The programmers who have worked on C++ can learn this language very efficiently.

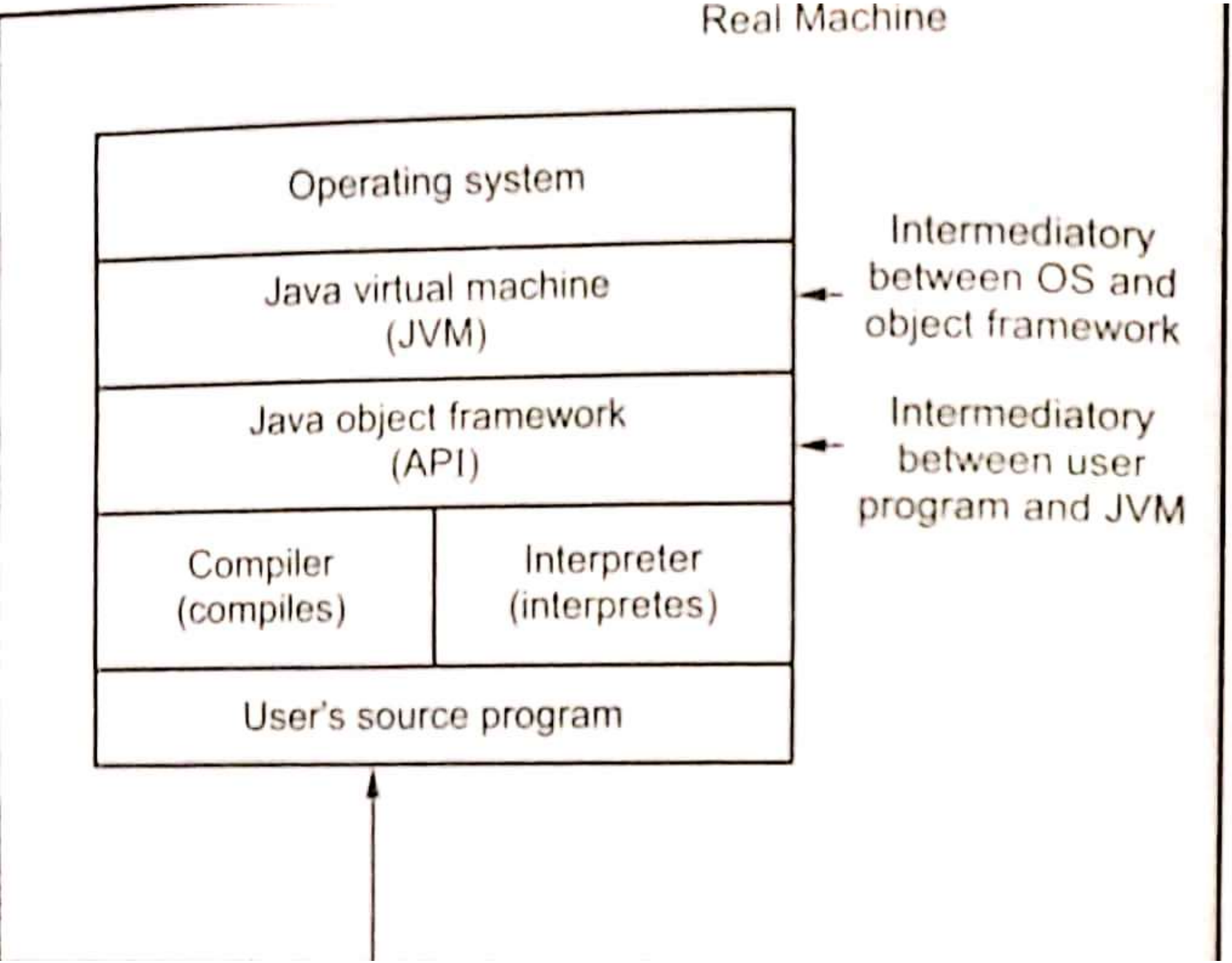
7. Java is a multithreaded and interactive language

Java supports multi-threaded programming which allows a programmer to write such a program that can perform many tasks simultaneously. This allows the programmer to develop the interactive systems.

8. Java is known for its high performance, scalability, monitoring and manageability

Due to the use of bytecode the Java has high performance. The use of multi-threading also helps to improve the performance of the Java. The J2SE helps to increase the scalability in Java. For monitoring and management Java has large number of Application Programming Interfaces(API). There are tools available for monitoring and tracking the information at the application level.

Sr. No.	C	C++	Java
1	The C language needs to be **compiled**.	The C++ is a language that needs to be **compiled.**	The Java is a language that ge **interpreted** and **compiled.**
2.	The C is platform **dependant.**	C++ is platform **dependant.**	Java is a **platform independen**
3.	There is no concept of **threading** in C.	C++ does not support multi-threading programming	Java supports **multi-threading**
4.	Using C, the GUI applications cannot be created	C++ does not have facility to create and implement the graphical user interface.	Using Java, one can design ve interactive and **user friendly graphical user interface.**
5.	C does not support database oriented applicaiton	Database handling using C++ is very complex. The C++ does not allow the database connectivity	**Java servlets** can be created which can interact with the **databases** like MS-ACCESS, Oracle, My-SQL and so on.
6.	C cannot be embedded inscripting language.	C++ code cannot be embedded in any scripting language.	Java code can be **embedded within a scripting language** b means of Applet programming
7	There is no concept of **inheritance.**	C++ supports **multiple inheritance.**	Java does **not support multipl inheritance** however it makes use of interface
8.	C uses **pointers.**	In C++ we can use the **pointers**	In Java there is **no concept of pointers.**
9.	C does not support tempelate.	C++ supports **templates**	Java does not support the concept of templates.


Real Machine
Operating system
Java virtual machine
(JVM)
Java object framework
(API)
Compiler
(compiles)
Interpreter
(interpretes)
User's source program
Intermediatory
between OS and
object framework
Intermediatory
between user
program and JVM

Documentation Section

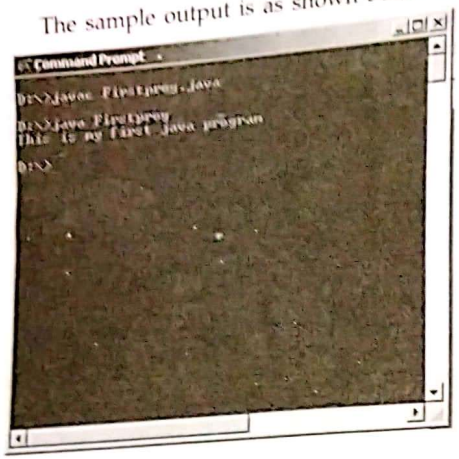
package statements section

import statements section

Class Definition

```
 Main Method class
{
        public static void main(String[] args)
       {
        //main method definition
        }
}
```

The sample output is as shown below for the program *Firstprog.java*



Program explanation

In our First Java program, on the first line we have written a comment statement as

```
/*
    This is my First Java Program
    */
```

This is a multi-line comment. Even a single line comment like C++ is also allowed Java. Hence if the statement would be

```
//This is my First Java Program
```

is perfectly allowed in Java. Then actual program starts with

```
class Firstprog
```

Here class is a keyword and Firstprog is a variable name of class. Note that the na of the class and name of your Java program **should be the same.** The class definiti should be within the curly brackets. Then comes

```
public static void main(String args[])
```

This line is for a function **void main().** The main is of type *public static void* .

The *public* is a access mode of the *main()* by which the class visibility can be defin Typically *main* must be declared *as public.*

The parameter which is passed to main is String args[]. Here *String* is a class na and *args[]* is an array which receives the command line arguments when the progr runs.

```
System.out.println("This is my first java program");
```


To print any message on the console *println* is a method in which the string "This is my first java program" is written. After println method, the newline is invoked. That means next output if any is on the new line. This *println* is invoked along with *System.out*. The *System* is a predefined class and *out* is an output stream which is related to console. The output as shown in above figure itself is a self explanatory. Also remember one fact while writing the Java programs that it is a **case sensitive** programming language like C or C++.

Now let us discuss few Java programming aspects such as keyword, operator, data type and so on.

1.8 Java Tokens

The smallest individual and logical unit of the java statements are called **tokens**. In Java there are five types of tokens used. These tokens are -

1. Reserved keywords
2. Identifiers
3. Literals
4. Operators
5. Separators

Reserved keywords

Keywords are the reserved words which are enlisted as below-

abstract	default	int	super
assert	double	interface	switch
boolean	else	long	this
byte	extends	native	throw
break	final	new	throws
case	for	package	transient
catch	float	private	try
char	goto	protected	void
class	if	public	volatile
const	implements	return	while
continue	import	short	true
do	instanceof	static	false
			null

Identifiers

Identifiers are the kind of tokens defined by the programmer. They are used naming the classes, methods, objects, variables, interfaces and packages in the progra

Following are the rules to be followed for the identifiers-

1. The identifiers can be written using alphabets, digits, underscore and dollar sign
2. They should not contain any other special character within them.
3. There should not be a space within them.
4. The identifier must not start with a digit, it should always start with alphabet.
5. The identifiers are case-sensitive. For example -

```
int a;
int A;
```

In above code a and A are treated as two different identifiers.

6. The identifiers can be of any length.

Literals

Literals are the kind of variables that store the sequence of characters for represen the constant values. Five types of literals are-

1. Integer literal
2. Floating point literal
3. Boolean literal
4. Character literal
5. String literal

As the name suggests the corresponding data type constants can be stored w these literals.

Operators

Operators are the symbols used in the expression for evaluating them.

Separators

For dividing the group of code and arranging them systematically certain symbo used, which are known as separators. Following table describes various separators.

Name of the Separator	Description
( )	Parenthesis are used to - enclose the arguments passed to it to define precedence in the expression for surrounding cast type
{ }	Curly brackets are used for - initialising array for defining the block
[ ]	Square brackets are used for - declaring array types dereferencing array values
;	Used to terminate the statement
,	Commas are used to separate the contents
.	Period is used to separate the package name from subpackages

1.9 Variables

A variable is an identifier that denotes the storage location.

Variable is a fundamental unit of storage in Java. The variables are used in combination with identifiers, data types, operators and some value for initialization. The variables must be declared before its use. The **syntax** of variable declaration will be -

```
data_type name_of_variable [=initialization][,=initialization][,...];
```

Following are some rules for variable declaration -

- The variable name should not begin with digits.
- No special character is allowed in identifier except underscore.
- There should not be any blank space with the identifier name.
- The identifier name should not be a keyword.
- The identifier name should be meaningful.

For Example :

```
int a,b;
char m='a';
byte k=12,p,t=22;
```

The variables have a scope which defines their visibility and a lifetime.

```
/*
This program introduces use of various data type
in the program
*/
class DatatypeDemo
{
 public static void main(String args[])
 {
  byte a=10;
  short b=10*128;
  int c=10000* 128;
  long d=10000*1000*128;
  double e=99.9998;
  char f='a';
  boolean g=true;
  boolean h=false;
  System.out.println("The value of a=: "+a);
  System.out.println("The value of b=: "+b);
  System.out.println("The value of c=: "+c);
  System.out.println("The value of d=: "+d);
  System.out.println("The value of e=: "+e);
  System.out.println("The value of f=: "+f);
  f++;
  System.out.println("The value of f after increment=: "+f);
  System.out.println("The value of g=: "+g);
  System.out.println("The value of h=: "+h);
 }
}
```

dynamic ic initialization

Output

```
The value of a=: 10
The value of b=: 1280
The value of c=: 1280000
The value of d=: 1280000000
The value of e=: 99.9998
The value of f=: a
```

```
The value of f after increment=: b
The value of g=: true
The value of h=: false
```

Dynamic initialization

Java is a flexible programming language which allows the dynamic initialization variables. In other words, at the time of declaration one can initialize the variables (*that in the above program we have done dynamic initialization*). In Java we can declare variable at any place before it is used.

Review Question

1. *What are various data types used in Java.* **GTU : Dec.-11, Marks 7**

1.11 Escape Sequences

The escape sequences are the characters used in print statement. Various su characters and their meaning is as given below -

Character	Meaning
\n	New line
\b	For back space
\t	Tab space
\\	printing \ character
\"	printing \" character "
\'	printing \' character '

Java Program [EscapeDemo.java]

```
/*    *
This java program illustrates
use of escape sequence
*/
class EscapeDemo
{
public static void main(String args[])
{
 System.out.print("\nThis is a newline demo\n");
 System.out.println("\t This is a tab demo");
 System.out.println("\" is printed");
}
}
```


Output

```
This is a newline demo
  This is a tab demo
is printed
```

1.12 Operators

Various operators that are used in Java are enlisted in following table -

Type	Operator	Meaning	Example
Arithmetic	+	Addition or unary plus	c=a+b
	–	Subtraction or unary minus	d= – a
	*	Multiplication	c=a*b
	/	Division	c=a/b
	%	Mod	c=a%b
Relational	<	Less than	a<4
	>	Greater than	b>10
	<=	Less than equal to	b<=10
	>=	Greater than equal to	a>=5
	==	Equal to	x==100
	!=	Not equal to	m!=8
Logical	&&	And operator	0&&1
	||	Or operator	0 ||1
Assignment	=	is assigned to	a=5
Increment	++	Increment by one	++i or i++
Decrement	– –	Decrement by one	– – k or k– –

Conditional operator

The conditional operator is "?" The syntax of conditional operator is

Condition?expression1:expression2

Where expression1 denotes the true condition and expression2 denotes false ondition.

. Program for demonstrating relational operators

```
lass RelOperDemo

ublic static void main(String[] args)
{
 System.out.println("\n Performing Relational operations ");
 int a=10,b=20,c=10;
 System.out.println("a= "+a);
 System.out.println("b= "+b);
 System.out.println("\n The a<b is "+(a<b));
 System.out.println("\n The a>b is "+(a>b));
 System.out.println("\n The a==c is "+(a==c));
 System.out.println("\n The a<=b is "+(a<=b));
 System.out.println("\n The a>=c is "+(a>=c));
 System.out.println("\n The a!=b is "+(a!=b));
```

Output

```
erforming Relational operations
= 10
```

the left operand only then the right operand won't be evaluated and hence the order of evaluation will always be from left to right irrespective of whether you place the operand within parentheses or not. For example : Suppose we have an expression as 'operand1 && operand2' and 'operand1' is evaluated to be 'false', then there is no point evaluating the right operand 'operand2' as irrespective of whether that operand is 'true' or 'false' the expression will always be 'false' only

Shift operators

There are left, right shift and unsigned right shift operators in Java.

Left shift operator :

The left shift operator is denoted by <<. The syntax of left shift is

value<<n where the bits in **value** are shifted by **n** positions to the left.

For example 5 << 2 will result in

0000 0101 << 2 → '0000 → 1010 →'0001 0100 = 20

Right shift operator :

The right shift operator is denoted by **>>.** The syntax of right shift is *Value>>n* where the bits in **value** are shifted by **n** positions to the right.

For example 5 >> 2 will result in

0000 0101 >> 2 →'0000 0010 '→0000 0001 = 1

Unsigned right shift operator :

The unsigned right shift operator is denoted by >>>. The syntax of unsigned right shift operator is *Value>>>n* where the bits in **value** are shifted by **n** positions to the right and fills MSB with 0.

For example 5 >>> 2 will result in

0000 0101 >>> 2 →'0000 0010 →'0000 0001 = 1

Review Question

1. *Explain various operators used in Java.*

1.13 Control Statements

Programmers can take decisions in their program with the help of control statements. Various control statements that can be used in java are -

1. if statement
2. if else statement
3. while statement

```
if(raining==true)
{
System.out.println("I won't go out");
System.out.println("I will watch T.V. Serial");
System.out.println("Also will enjoy coffee");
}
else

{
System.out.println("I will go out");
System.out.println("And will meet my friend");
System.out.println("we will go for a movie");
System.out.println("Any how I will enjoy my life");
}
```

if...else if statement

The syntax of if...else if statement is

```
if(is condition true?)
statement
else if(another condition)
statement
else if(another condition)
statement
else
statement
```

For example

```
if(age==1)
 System.out.println("You are an infant");
else if(age==10)
 System.out.println("You are a kid");
else if(age==20)
 System.out.println("You are grown up now");
else
System.out.println("You are an adult");
```

```
    System.out.println("The x is greatest");
  else
    System.out.println("The z is greatest");
}
else
{
   if(y>z)
      System.out.println("The y is greatest");
   else
      System.out.println("The z is greatest");
  }
 }
}
```

Output

```
The y is greatest
```

Java Program [whiledemo.ja...]

```
/*
This is java program which illustrates
while statement
*/
class whiledemo
{
public static void main(String args[])
{
  int count=1,i=0;
  while(count<=5)
  {
   i=i+1;
   System.out.println("The value of i= "+i);
   count++;
  }
}
}
```

Output

```
The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5
```

4. do... while statement

This is similar to while statement but the only difference between the two is that in case of do...while statement the statements inside the do...while must be executed at least once. This means that the statement inside the do...while body gets executed first and then the while condition is checked for next execution of the statement, whereas in the while statement first of all the condition given in the while is checked first and then the statements inside the while body get executed when the condition is true.

Syntax

```
or (statement 1;statement 2;statement 3)
execute this statement;
```

Compound for loop :

```
or(statement 1;statement 2; statement 3)

execute this statement;
execute this statement;
execute this statement;
that's all;
```

Here

Statement 1 is always for initialization of conditional variables,

Statement 2 is always for terminating condition of the for loop,

Statement 3 is for representing the stepping for the next condition.

For example :

```
or(int i=1;i<=5;i++)

System.out.println("Java is an interesting language");
System.out.println("Java is a wonderful language");
System.out.println("And simplicity is its beauty");
```

Let us see a simple Java program which makes use of for loop.

Java Program [forloop.java]

```
*
This program shows the use of for loop
/
lass forloop
```

Example 1.13.2 *Write a program in Java to print the first n Fibonacci numbers.*

GTU : Dec.-11, Marks 5

Solution :

```
lass Fibonacci

    public static void main(String args[])
    {
        int a,b, c, n;
        a=b=1;
        for(n=1;n<=10;n++)
        {
            System.out.print(" "+a);
            c=a+b;
            a=b;
            b=c;
        }
}
```

Output

Q.3 *Explain features of Java.* **[Refer section 1.2]**

Summer 2014

Q.4 *Explain features of Java.* **[Refer section 1.2]**

Winter 2014

Q.5 *JVM is platform independent. Justify.* **[Refer section 1.4]**

2 Arrays and String

5

Syllabus

Single and multidimensional array, String class, StringBuffer class, Operations on string, Command line argument, Use of wrapper class.

Contents

2.1 *Arrays*

2.2 *String Class*

2.3 *Operations on String*

2.4 *String Buffer Class* ..... ***Winter-12,13, Summer-14*** · *Marks 7*

2.5 *Command Line Argument*

2.6 *Use of Wrapper Class*

2.7 *University Questions with Answers*

and to allocate the memory -

```
array_name=new data_type[size];
```

where *array_name* represent name of the array, **new** is a keyword used to allocate memory for arrays, *data_type* specifies the data type of array elements and *size* represents the size of an array. For example :

```
a=new int[10];
```

[Note that in C/C++ this declaration is as good as int a[10]]

After this declaration the array a will be created as follows

Array a[10]

0	0	0	0	0	0	0	0	0	

That means after the above mentioned declaration of array, all the elements of array will get initialized to zero. Note that, always, while declaring the array in Java, we make use of the keyword **new**, and thus we actually make use of dynamic memory allocation. Therefore arrays are **allocated dynamically** in Java. Let us discuss on simple programs based on arrays.

Java Program [SampleArray.Java]

```
/*
This is a Java program which makes use of arrays
*/
class SampleArray
{
 public static void main(String args[])
 {
  int a[];
  a=new int[10];
  System.out.println("\tStoring the numbers in array");
  a[0]=1;
  a[1]=2;
  a[2]=3;
```

```
public class RaggedArray
{
    public static void main(String args[])
    {
        int A[][] = new int[3][];
        A[0] = new int[4];
        A[1] = new int[2];
        A[2] = new int[3];
        System.out.println("Total Number of Rows: " + A.length);
        A[0][0] = 1;
        A[0][1] = 2;
        A[0][2] = 3;
        A[0][3] = 4;
        // 2nd row
        A[1][0] = 5;
        A[1][1] = 6;
        // 3rd row
        A[2][0] = 7;
        A[2][1] = 8;
        A[2][2] = 9;
        System.out.println("\nArray Representation");
        for(int i = 0; i < A.length; i++)
        {
            for(int j = 0; j < A[i].length; j++)
            {
```

```
{

int a[][]=new int[3][3];
int k=0;
System.out.println("\tStoring the numbers in array");
 for(int i=0;i<3;i++)
 {
  for(int j=0;j<3;j++)
  {
    a[i][j]=k+10;
    k=k+10;
  }
 }
 System.out.println("You have stored...");
 for(int i=0;i<3;i++)
 {
  for(int j=0;j<3;j++)
  {
    System.out.print(" "+a[i][j]);
  }
  System.out.println();
 }
}
}
```

Output

```
      Storing the numbers in array
You have stored...
 10 20 30
 40 50 60
 70 80 90
```

The typical application of two dimensional arrays is performing matrix operat
Let have some simple java program in which various matrix operations are performe

Java Program[Matrix.java]

```
/*
This is a Java program which performs matrix operations
*/
class Matrix
{
public static void main(String args[])
throws java.io.IOException
{

int a[][]={
        {10,20,30}
```

```
            System.out.print(" "+c[i][j]);
           }
           System.out.println();
          }
          break;
case '3':for(int i=0;i<3;i++)
          {
          for(int j=0;j<3;j++)
          {
          c[i][j]=0;
          for(int k=0;k<3;k++)
          {
          c[i][j]=c[i][j]+a[i][k]*b[k][j];
         }
        }
       }
       System.out.println("The Multiplication is...");
       for(int i=0;i<3;i++)
       {
        for(int j=0;j<3;j++)
        {
         System.out.print(" "+c[i][j]);
        }
        System.out.println();
       }
       break;
 }
}
}
```

Output (Run1)

```
Main Menu
1. Addition
2. Subtraction
3. Multiplication
Enter your choice
1
The Addition is...
11 22 33
44 55 66
77 88 99
```

Output (Run2)

```
Main Menu
1. Addition
2. Subtraction
3. Multiplication
Enter your choice
```



```
2
The Subtraction is...
9 18 27
36 45 54
63 72 81
```

Output (Run3)

```
Main Menu
1. Addition
2. Subtraction
3. Multiplication
Enter your choice
3
The Multiplication is...
300 360 420
660 810 960
1020 1260 1500
```

Review Question

1. *Explain the concept of one and two dimensional array with the help of examples.*

GTU : Dec.-11, Marks 7

2.2 String Class

String is a collection of characters. In Java **String** defines the object. We can make use of **String** to denote the collection of characters. Let us look at a sample program in which the string object is used in order to handle some text.

Java Program [StringDemo.Java]

```
/*This is a Java program which makes use of strings
*/

class StringDemo

{
public static void main(String args[])
{
String s="Hello, How are You?";
System.out.println(s);
}
```

Output

```
Hello, How are You?
```

Alternatively, String can be denoted as the array of characters. For example :

```
char str[]={'P','R','O','G','R','A','M'};
```

[illegible] Finding the Length of a String

[illegible] We can find the length of a given string using the method length(). The program shown for use of length() method for strings

[illegible] Program (file: [illegible])

```
[illegible]

public static void main(String[] args)
{

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
}
```

program is slightly

Java Program

```
class Test
{
public static void main(String[] args)
{
String str=new String("I like ");
String fruit=new String("mango very much");
System.out.println(str.concat(fruit));
}
}
```

Output

```
I like mango very much
```

2.3.3 Character Extraction

As we know that the string is a collection of characters. **String** class provi facility to extract the character from the String object. There is a method **charAt(index)** with the help of which we can extract the character denoted by ndex in the array.

For example :

```
String fruit=new String("mango");
char ch;
ch=fruit.charAt(2);
System.out.println(ch);
```

The output of above code fragment will be n because at the index position 2 of bject fruit the character n is present.

2.3.4 String Comparison

Sometimes we need to know whether two strings are equal or not. We use t uals() for that purpose. This method is of Boolean type. That is, if two str ual then it returns true otherwise it returns false.

The syntax is -

```
class StringCompareDemo
{
public static void main(String[] args)
{
String str1=new String("INDIA");
String str2=new String("india");
 if(str1.equals(str2)==true)
  System.out.println("\n The two strings are equal");
 else
  System.out.println("\n The two strings are not equal");
}
```

Output

```
D:\>javac StringCompareDemo.java

D:\> java StringCompareDemo
     The two strings are not equal
```

The above program will generate the message "The two strings are not equal" nd str2 are not equal but if we write same str1 and str2 then naturally the outp e "The two strings are equal". This string comparison is case sensitive. But if we o compare the two strings without caring for their case differences then we can u ethod **equalsIgnoreCase().**

2.3.5 Searching for the Substring

We can look for the desired substring from the given string using a m **ubstring().** The syntax of method substring() is as follows -

```
String substring(int start_index,int end_index)
```

Java Program [SubstringDemo.Java]

```
class SubstringDemo
{
public static void main(String[] args)
{
String str1=new String("I love my country very much");
  System.out.println("\n One substring from the given sentence
          is:"+str1.substring(2,6));
  System.out.println("\n And another substring
                    is:"+str1.substring(18));
}
```

Output

```
I like mango very much
```

We can make use of a method **concat()** for concatenating two strings. The program is slightly modified for the use of concat() -

Java Program

```
class Test
{
public static void main(String[] args)
{
String str = new String("I like ");
String fruit = new String("mango very much");
System.out.println(str.concat(fruit));
}
}
```

Output

```
I like mango very much
```

2.3.3 Character Extraction

As we know that the string is a collection of characters. **String** class provides facility to extract the character from the String object. There is a method **charAt(index)** with the help of which we can extract the character denoted by index in the array.

For example :

```
String fruit = new String("mango");
char ch;
ch = fruit.charAt(2);
System.out.println(ch);
```

The output of above code fragment will be n because at the index position 2 of object fruit the character n is present.

2.3.4 String Comparison

Sometimes we need to know whether two strings are equal or not. We use equals() for that purpose. This method is of Boolean type. That is, if two strings equal then it returns true otherwise it returns false.

The syntax is -

```
boolean equals(String Str);
```


Let us see one illustrative program -

Java Program [StringCompareDemo.Java]

```
class StringCompareDemo

public static void main(String[] args)
{
String str1 = new String("INDIA");
String str2 = new String("India");
 if(str1.equals(str2) == true)
  System.out.println("\n The two strings are equal");
 else
  System.out.println("\n The two strings are not equal");
}
```

Output

```
D:\>javac StringCompareDemo.java

D:\> java StringCompareDemo
     The two strings are not equal
```

The above program will generate the message "The two strings are not equal" if str1 and str2 are not equal but if we write same str1 and str2 then naturally the output will be "The two strings are equal". This string comparison is case sensitive. But if we want to compare the two strings without caring for their case differences then we can use the method **equalsIgnoreCase()**.

2.3.5 Searching for the Substring

We can look for the desired substring from the given string using a method **substring()**. The syntax of method substring() is as follows -

```
String substring(int start_index,int end_index)
```

Java Program [SubstringDemo.Java]

```
class SubstringDemo

public static void main(String[] args)
{
String str1 = new String("I love my country very much");
  System.out.println("\n One substring from the given sentence
          is " + str1.substring(2,6));
  System.out.println("\n And another substring
                    is:" + str1.substring(18));
 }
```

```
D:\>javac SubstringDemo.java

D:\>java SubstringDemo

One substring from the given sentence is:love
And another substring is:very much
```

2.3.6 Replacing the Character from String

We can replace the character by some desired character. For example

Java Program [replaceDemo.java]

```
class replaceDemo
{
public static void main(String[] args)
{
String str=new String("Nisha is Indian ");
String s=str.replace('i','a');
System.out.println(s);

 }
}
```

Output

```
D:\>javac replaceDemo.java
D:\>java replaceDemo
Nasha as Indaan
```

2.3.7 Upper Case and Lower Case

We can convert the given string to either upper case or lower case using the m **toUpperCase()** and **toLowerCase().** Following program demonstrates the handling case of the string -

Java Program [CaseDemo.Java]

```
class caseDemo
{
public static void main(String[] args)
{
 String str=new String("Nisha is Indian ");
 System.out.println("\n The original string is: "+str);
 String str_upper= str.toUpperCase();
 System.out.println("\n The Upper case string is: "+str_upper);
 String str_lower= str.toLowerCase();
 System.out.println("\n The Lower case string is: "+str_lower);
}
```



```
}
```

Output

```
D:\>javac caseDemo.java
D:\>java caseDemo

The original string is: Nisha is Indian

The Upper case string is: NISHA IS INDIAN

The Lower case string is: nisha is indian
```

Example 2.3.1 *Write a method that counts the number of occurrences of given character in a string.* **GTU : Dec.-11, Marks 7**

Solution :

Java Program[CharOccurCount.java]

```
import java.lang.*;
class CharOccurCount
{
public static void main(String[] args)
{
 String name="malayalam";
 int i,count=0;
 System.out.println("\n In the word 'Malayalam'...");
 for(i=0;i<name.length();i++)
 if(name.charAt(i)=='a')
  count++;
 System.out.println("Occurrences of 'a' is: "+count);
 count=0;
for(i=0;i<name.length();i++)
 if(name.charAt(i)=='l')
  count++;
 System.out.println("Occurrences of 'l' is: "+count);
 count=0;
for(i=0;i<name.length();i++)
 if(name.charAt(i)=='m')
  count++;
 System.out.println("Occurrences of 'm' is: "+count);
}
```

Output

```
In the word 'Malayalam'...
```

```
Occurrences of 'a' is: 4
Occurrences of 'T' is: 2
Occurrences of 'm' is: 2
```

2.4 String Buffer Class

GTU : Winter-12,13, Summer-14, Marks

The **StringBuffer** is a class which is alternative to the **String** class. But StringBuf class is more flexible to use than the **String** class. That means, using **StringBuffer** can insert some components to the existing string or modify the existing string bu case of **String** class once the string is defined then it remains fixed. The StringBuffer the StringBuilder are almost one and the same. The StringBuilder or StringBuffer h three constructors and 30 methods. Following are some simple methods used StringBuffer -

Name of method	Description
append(String str)	Appends the string to the buffer
capacity()	It returns the capacity of the string buffer
charAt(int index)	It returns a specific character from the sequence which is specified by the index.
delete(int start,int end)	It deletes the characters from the string specified by the starting and ending index.
insert(int offset,char ch)	It inserts the character at the position specified by the offset.
length()	It returns the length of the string buffer.
setCharAt(int index,char ch)	The character specified by the index from the stringbuffer is set to ch.
setLength(int new_len)	It sets the length of the string buffer.
toString()	It converts the string representing data in this string buffe
replace(int start,int end,String str)	It replaces the characters specified by the new string
reverse()	The character sequence is reversed.

These methods can be illustrated with the help of following Java Programs -

Example 2.4.1 *Write a Java Program illustrating the difference between the capacity and length function of StringBuffer.*

Solution :

Java Program [StringBuffDemo1.java]

```
public class StringBuffDemo1{
public static void main(String args[])
{
StringBuffer str=new StringBuffer("Java");
System.out.println("The String Buffer is "+str);
System.out.println("The length is "+str.length());
System.out.println("The capacity is"+ str.capacity());
}
}
```

Creating string buffer in str

Finding length of str

Finding capacity of str

Output

```
F:\test>javac StringBuffDemo1.java

F:\test>java StringBuffDemo1
The String Buffer is Java
The length is 4
The capacity is 20
```


Program Explanation

Using the StringBuffer/StringBuilder class, we can create a buffer of characters. The length function returns the number of characters in the string and the **capacity** function returns the number of characters in the string +16 additional characters.

Example 2.4.2 *Write a Java program, to convert the string 'Great' to a new string 'God'.*

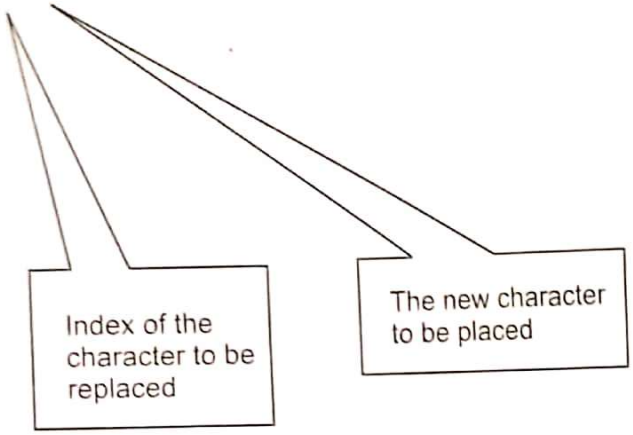
Solution :

```
public class StringBuffDemo2{
 public static void main(String args[])
 {
  StringBuffer str=new StringBuffer("Great");
  System.out.println("The String is: "+str);
  str.setCharAt(1,'o');
  str.setCharAt(2,'d');
  str.setLength(3);
  System.out.println("Now the String is: "+str);

 }
}
```

Output

```
F:\test>javac StringBuffDemo2.java

F:\test>java StringBuffDemo2
The String is: Great
Now the String is: God
```

```
setChar(1,'o');
```



Similarly, there is a **setLength** function which sets the length of the string. The str 'Great' has a length 5. To convert this string to a new string say 'God', the length is to 3. The above given output simplifies the program.

Example 2.4.3 *Write a simple Java program to append the string "God" by the string " Great" and then append an exclamation mark to the complete string.*

Solution :

Java Program [StringBuffDemo3.java]

```
public class StringBuffDemo3{
 public static void main(String args[])
 {
  StringBuffer str=new StringBuffer("God");
  System.out.println("Initially the String is:  "+str);
  str.append(" is Great");
  str.append("!");
  System.out.println("Now the String is: "+str);
 }
}
```

Output

```
F:\test>javac StringBuffDemo3.java

F:\test>java StringBuffDemo3
Initially the String is: God
Now the String is: God is Great!
```

Example 2.4.6 *Write a simple Java program to change the String "Impossible" to "po*

olution :

ava Program [StringBuffDemo6.java]

```
ublic class StringBuffDemo6{
public static void main(String args[])
{
 StringBuffer str1=new StringBuffer("Impossible");
 System.out.print("The word "+str1);
 str1.delete(0,2);
   System.out.println(" is changed to "+str1);
 }
}
```

Output

```
F:\test>javac StringBuffDemo6.java
F:\test>java StringBuffDemo6
The String Impossible is changed to possible
```

```
while(i<args.length)
{
  str=args[i];
  i=i+1;
  System.out.println(str);
}
System.out.println("\n The total number of arguments are..."+i);
}
}
```

Output

```
D:\>java CommandDemo aaa bbb ccc ddd eee fff

Following are the strings entered at command prompt
aaa
bbb
ccc
ddd
eee
fff
The total number of arguments are...6
```

Review Question

1. *Explain the concept of command line argument with illustrative Java program.*

2.6 Use of Wrapper Class

- Wrapper classes are those classes that allow primitive data types to be as objects.
- The wrapper class is one kind of wrapper around a primitive data ty

long	java.lang.Long
short	java.lang.Short
void	java.lang.Void

- Methods to handle wrapper classes are enlisted in the following table -

Method used	Description
Integer val=new Integer(int_var)	An object is created for the integer variable int_var.
Float val=new Float(f_var)	An object is created for the Float variable f_var.
Double val=new Double(d_var)	An object is created for the double variable d_var.
Long val=new Long(Long_var)	An object is created for the Long variable Long_var.

- Suppose an object for holding an integer value is created then we can retrieve the integer value from it. For instance the object **obj** contains an integer value then we can obtain the integer value from **obj.** It is as follows -

```
int num=obj.intValue();
```

Similarly we can use floatValue(), doubleValue(), and longValue().

- Similarly in order to convert the numerical value to string the **toString()** method can be used. For instance

```
str=Integer.toString(int_val)
```

The varibale **int_val** can be converted to string **str.**

- For converting the string to numerical value the **parseInt** or **parseLong** methods can be used.

```
class WrapperDem
{
 public static void main(String[] args)
 {
 System.out.println("Creating an object for value 10");
 Integer i=new Integer(10);
 System.out.println("Obtaining the value back from the object: "+i.intValue());
 String str="100";
 System.out.println("The string is: "+str);
 System.out.println("Obtaining the numeric value from the string: "+Integer.parse
 }
}
```

Output

```
Creating an object for value 10
Obtaining the value back from the object: 10
The string is: 100
Obtaining the numeric value from the string: 100
```

Review Question

1. *Write a Java program to illustrate Wrapper Class.* GTU : Dec.-10, M

2.7 University Questions with Answers

(Regulation 2008)

Winter 2012

Q.1 *Differentiate between - String class and StringBuffer class.* **[Refer section 2.**

Winter 2013

Q.2 *Compare string with String buffer. Also write a program to count occurren character in a string.* **[Refer section 2.4]**

Summer 2014

Q.3 *Differentiate between - String class and StringBuffer class.* **[Refer section 2.**

3 Classes, Objects and Methods

15

Syllabus

Class, Object, Object reference, Constructor, Constructor overloading, Method overloading, Recursion, Passing and returning object form method, New operator, this and static keyword, finalize() method, Access control, Modifiers, Nested class, Inner class, Anonymous inner class, Abstract class.

Contents

3.1 Introduction
3.2 Class
3.3 Object
3.4 Assessing Class Members
3.5 Object Reference
3.6 Constructor **Nov.-11, Winter-14**
..... **Dec.-11, Summer-13** ..... Marks 7
3.7 Method Overloading
3.8 Recursion
3.9 Passing and Returning Object from Method
3.10 New Operator ..... **Dec.-11, Summer-13** ..... Marks 7
3.11 this Keyword
3.12 Static Keyword
3.13 Finalize() Method
3.14 Access Control and Modifiers ..... **Winter-14** ..... Marks 7
3.15 Nested Class
3.16 Inner Class
3.17 Abstract Class
3.18 University Questions with Answers

```
class classname
{
        type variable1;
        type variable2;
        //...
        type method1(parameter-list)
        {
                ...
                ...
        }
        type method2(parameter-list)
        {
                ...
                ...
        }
        ...
        type methodn(parameter-list)
        {
                ...
                ...
        }
```

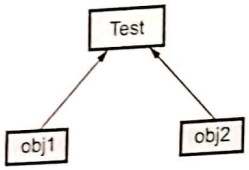
Now if we create two objects then it can be

Test obj1, obj2;

obj1 = new Test () ;

obj2 = new Test () ;

It can be graphically as shown below -



3.4 Assessing Class Members

- There are two types of class members - The data members and the method.
- These members can be accessed using **dot** operator.
- For accessing the data members the syntax is -

  name_of_object.variable_name=value;
- For accessing the method of the class the syntax is -
  name_of_object.methodname(parameter list);
- For example -

  obj.name="XYZ"

  obj.display();
- Suppose we wish pass the parameters to the method then first we will create object for the class as follows -

  Test obj1=new Test();

  Test obj2=new Test();

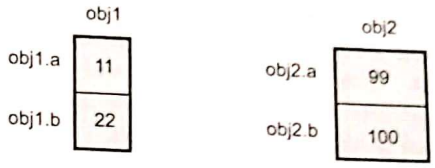
Now two objects are created namely obj1 and obj2.

obj1.get_val(11,22);

obj2.get_val(99,100);

Suppose by this method we assign values to two variables a and b then -

It can be graphically represented as -



Let us see the simple Java program which makes use of classes and objects -

Java Program [Rectangle.java]

```
/*This is a Java program which shows the
use of class in the program
*/
class Rectangle
{
 int height;
 int width;
 void area()
 {
  int result=height*width;
  System.out.println("The area is "+result);
 }
}
class classDemo
{
 public static void main(String args[])
 {
  Rectangle obj=new Rectangle();
  obj.height=10;//setting the attribute values
  obj.width=20;//from outside of class
  obj.area();//using object method of class is called
 }
}
```

Class with attributes and methods defined within it.

Another class in which main() function is written.

Output

```
The area is 200
```

In the above program we have used two classes one class is classDemo which is our usual one in which the main function is defined and the other class is Rectangle. In this class we have used height and width as attributes and one method area() for calculating area of rectangle. In the main function we have declared an object of a class as

```
Rectangle obj=new Rectangle();
```

And now using obj we have assigned the values to the attributes of a class.

Let us now understand with the help of programming example two things - 1. How create object references and 2. Even though we assign null one of the object the other py of the object remains.

```
ass Point

    int x;
    int y;

ass PointDemo

    public static void main(String args[])
    {
          Point p1=new Point();
          Point p2=p1;
          p1.x=10;
          p1.y=20;
          System.out.println("\n Point p1 is...");
          System.out.println("x= "+p1.x);
          System.out.println("y= "+p1.y);

          System.out.println("\n Point p2 is...");
          System.out.println("x= "+p2.x);
          System.out.println("y= "+p2.y);

          System.out.println("\n\tAssigning null to object p1...");
          p1=null;
          System.out.println("\tStill p2 is alive!!!");

          System.out.println("\n Point p2 is...");
          System.out.println("x= "+p2.x);
          System.out.println("y= "+p2.y);


     }
```

Output

```
\>javac PointDemo.java
\>java PointDemo

oint p1 is...
= 10
= 20

oint p2 is...
= 10
```

```
    }
}

class Constr_Demo {
public static void main(String args[])
{
    Rectangle1 obj=new Rectangle1();
    obj.area();//call the to method
    Rectangle1 obj1=new Rectangle1(11,20);
    obj1.area();//call the to method

}
}
```

Object created and simple constructor is invoked

Object created and parameterised constructor is invoked

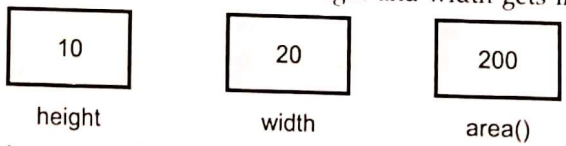
Output

```
F:\test>javac Rectangle1.java
F:\test>java Constr_Demo
Simple constructor: values are initialised...
Now, The function is called...
The area is 200
Parameterised constructor: values are initialised...
Now, The function is called...
The area is 220

F:\test>
```

Note the method of running the program

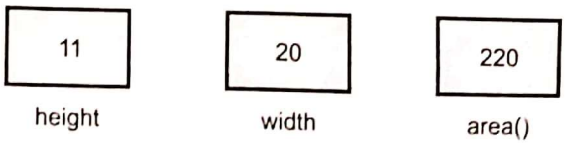
Program Explanation

In above program, there are two classes - **Rectangle1** and **Constr_Demo.** In the **Rectangle1** class, data fields, method and constructor is defined. In the **Constr_Demo** class the objects are created. When an object obj gets created, then the simple constructor **Rectangle1()** gets invoked and the data fields height and width gets initialised



Hence the **area** function will compute the area=10*20=200.

When an object **obj1** gets created, then the parameterised constructor **Rectangle1(11, 20)** gets invoked and the data fields height and width gets initialised.



Hence the area function will compute the area = 11*20 = 220.

The class Constr_Demo is called main class because it consists of main function.

Example 3.6.1 *Differentiate between constructor and method.* **GTU : Nov-11, Marks 3**

Solution :

Sr.No.	Constructor	Method
1.	The name of the constructor must be same as the class name.	The name of the method should not be the class name. It can be any with alphanumeric or alphabetic characters with restricted character length. The name must not start with digit or special symbols. Only underscore is allowed.
2.	It does not return anything hence no return type.	It can return and hence it has a return type. If method does not return anything then the return type is void.
3.	The purpose of constructor is to initialize the data members of the class.	The method is defined to execute the core logic of the class.
4.	The constructor is invoked implicitly at the time of object creation.	The method must be called explicitly using the object name and dot operator.

3.6.1 Properties of Constructor

1. Name of constructor must be the same as the name of the class for which it is being used.
2. The constructor must be declared in the **public** mode.
3. The constructor gets invoked automatically when an object gets created.
4. The constructor should not have any return type. Even a **void** data type should not be written for the constructor.
5. The constructor can not be used as a member of union or structure.
6. The constructors can have default arguments.
7. The constructor **can not be inherited.** But the derived class can invoke the constructor of base class.
8. Constructor can make use of **new** or **delete** operators for allocating or releasing memory respectively.
9. Constructor can not be **virtual.**
10. Multiple constructors can be used by the same class.
11. When we declare the constructor explicitly then we must declare the object of the class.

Example 3.6.2 *There is no destructor in Java. Justify* **GTU : Winter-14, Marks 3**

Solution : There are two purposes of destructor - 1. Freeing up the memory allocated for the objects. 2. Cleaning up resources like closing the open file stream. The garbage collection mechanism of Java takes care of these two tasks automatically. Hence there is no need for having destructor in Java.

3.6.2 Constructor Overloading

Overloading is one of the important concept in Object Oriented Programming. Similar to methods the constructors can also be overloaded.

Constructor overloading in Java allows to have *more than one constructor inside one class.* Multiple constructor with different signature with only difference that Constructor doesn't have return type in Java. Those constructor will be called as **overloaded constructor.**

Following is a simple Java program that illustrates the concept of constructor overloading.

```
public class Rectangle2 {
    int height,width;
    double ht,wd;
    Rectangle2(int h,int w)//constructor with two integer values
    {
        height=h;
        width=w;
    }
    Rectangle2(double h,double w)//constructor with two double values
    {
        ht=h;
        wd=w;
    }
    Rectangle2(int val)//constructor with single integer value
    {
        height=val;
    }
    void area1()
    {
        System.out.println("Now, The function is called...");
        int result=height*width;
        System.out.println("The area is "+result);
    }
    void area2()
    {
        System.out.println("Now, The function is called. .");
        double result=ht*wd;
        System.out.println("The area is "+result);
```

```
    }
    void area3()
    {
        System.out.println("Now, The function is called...");
        int result=height*height;
        System.out.println("The area is "+result);
    }

}
class OverLoadConstr
{
public static void main(String args[])
{
    Rectangle2 obj1=new Rectangle2(11,20);
    obj1.area1();//call the to method
    Rectangle2 obj2=new Rectangle2(11.33,20.22);
    obj2.area2();//call the to method
    Rectangle2 obj3=new Rectangle2(10);
    obj3.area3();//call the to method


}
}
```

Output

```
D:\>javac Rectangle2.java
D:\>java OverLoadConstr
Now, The function is called...
The area is 220
Now, The function is called...
The area is 229.08259999999998
Now, The function is called...
The area is 100
```

Program Explanation

In above program we have defined three constructors; there are two parameters that are passed to the first constructor. This constructor invokes the area1. To the second constructor the two double values are passed as arguments constructor invokes the method area2. Then the third constructor is defined which only one argument passed to it.

This constructor invokes the method area3. Depending upon the parameters the appropriate constructor gets invoked. This mechanism is called overloading.

Example 3.8.3 *Create a class called math techniques that supports the following constant and methods : i) Pi - 3.14159 ii) Area of rectangle that is passed a height and a width iii) Perimeter of rectangle that is passed a height and a width iv) Area of circle that is passed a radius v) Perimeter of circle that is passed a radius*

Create a separate class to test out the math technique class. (Make use of Constructor overloading)

Solution :

```
public class MathTechniques
{
    double pi;
    double height,width;
    double radius;
    MathTechniques(double h,double w)
    {
        height=h;
        width=w;
    }
    MathTechniques(double r)
    {
        pi=3.14159;
        radius=r;
    }
    void Rect_Area()
    {
        System.out.println("The height of rectangle is: "+height);
        System.out.println("The width of rectangle is: "+width);
        double area=height*width;
        System.out.println("The area of rectangle is: "+area);

    }
    void Rect_Peri()
    {
        double p=2*(height+width);
        System.out.println("The perimeter of rectangle is: "+p);
    }
    void Circle_Area()
    {
        System.out.println("The radius of circle is: "+radius);
        double area=pi*radius*radius;
        System.out.println("The area of circle is: "+area);

    }
    void Circle_Peri()
    {
        double p=2*pi*radius;
```

```
            System.out.println("The perimeter of circle is: "+p);
        }
}
class MainClass
{
    public static void main(String args[])
    {
        MathTechniques obj1=new MathTechniques(5,3);
        MathTechniques obj2=new MathTechniques(10);
        obj1.Rect_Area();
        obj1.Rect_Peri();
        System.out.println("----------------------------------------");
        obj2.Circle_Area();
        obj2.Circle_Peri();
    }
}
```

Output

```
The height of rectangle is: 5.0
The width of rectangle is: 3.0
The area of rectangle is: 15.0
The perimeter of rectangle is: 16.0
----------------------------------------
The radius of circle is: 10.0
The area of circle is: 314.159
The perimeter of circle is: 62.8318
```

Example 3.6.4 *Design a class named Fan to represent a fan. The class contains:*

- *Three constants named SLOW, MEDIUM and FAST with values 1,2 and 3 to deno the fan speed.*
- *An int data field named speed that specifies the speed of the fan (default SLOW).*
- *A boolean data field named f_on that specifies whether the fan is on(default false).*
- *A double data field named radius that specifies the radius of the fan (default 4).*
- *A data field named color that specifies the color of the fan (default blue).*
- *A no-arg constructor that creates a default fan.*
- *A parameterized constructor initializes the fan objects to given values.*
- *A method named display() will display description for the fan. If the fan is on, t display() method displays speed, color and radius. If the fan is not on, the method retur fan color and radius along with the message "fan is off".*

Write a test program that creates two Fan objects. One with default values and the oth with medium speed, radius 6, color brown, and turned on status true. Display t descriptions for two created Fan objects. **GTU : Dec.-11, Marks**

Solution :

```
class Fan
{
    public static final int SLOW = 1;
    public static final int MEDIUM = 2;
    public static final int FAST = 3;
    int speed;
    boolean f_on;
    double radius;
    String color;
    Fan()
    {
        speed = SLOW;
        f_on = false;
        radius = 4;
        color = "blue";
    }
    Fan(int s,boolean f, double r, String c)
    {
        speed = s;
        f_on = f;
        radius = r;
        color = c;
    }
    public void display()
    {
        System.out.println("\n Description For the Fan...");
        if(f_on)
        {
            System.out.println("Speed : "+speed+"(MEDIUM)");
            System.out.println("Radius : "+radius);
            System.out.println("Color : "+color);
        }
        else
        {
            System.out.println("Radius : "+radius);
            System.out.println("Color : "+color);
            System.out.println("Fan is Off.");
        }
    }
}
class testclass
{
    public static void main(String args[])
    {
        Fan obj1 = new Fan();
        obj1.display();
```

```
        Center_X=x;
        Center_Y=y;
        height=h;
        width=w;
    }
    double getArea()
    {
        return (height*width);
    }
    double getPerimeter()
    {

        return ( 2*(height+width) );
    }
    boolean contains(double x,double y)
    {
        X_Left=( Center_X - (width/2) );
        X_Right=( Center_X + (width/2) );
        Y_UP=(Center_Y + (height/2));
        Y_Down=(Center_Y - (height/2));
        if( (x > X_Left && x < X_Right) && (y < Y_UP && y > Y_Down))
            return true;     //point is inside rect.
        else
            return false;    //point is outside rect.
    }
}
class testclass
{
    public static void main(String args[])
    {
        Rectangle obj1 = new Rectangle();
        Rectangle obj2 = new Rectangle(300,300,100,200);
        System.out.println();
        System.out.println("\t\t Rectangle 1" );
        System.out.println(" Area "+obj1.getArea());
        System.out.println(" Perimeter "+obj1.getPerimeter());
        if(obj1.contains(10,20))
            System.out.println(" Contains (10,20) is inside the Rectangle ");
        else
            System.out.println(" Contains (10,20) is outside the Rectangle ");
        System.out.println("----------------------------------------------");
```

```
    else
        System.out.println(" Contains (250,310) is outside the Rectangle ");

    }
}
```

Example 3.6.6 *State whether any error exists in the following code. If so, correct the error a give output.*

```
class Test {
public static void main(String args[]) {
A a = new A();
a.print();
}
}
class A {
String s;
A(String s) {
this.s=s;
}
public void print() {
System.out.println(s);
}
}
```

GTU : Dec.-11, Marks

Solution : The error at the line where the instance a for the class A is created. S string must be passed as an argument to the constructor. The corrected code is as g below -

```
class Test {
public static void main(String args[]) {
A a = new A("JAVA");
a.print();
}
}
class A {
String s;
A(String s) {
this.s=s;
}
public void print() {
System.out.println(s);
}
```


Example 3.6.7 *Declare a class called coordinate to represent 3 dimensional cartesian coordinates (x, y and z). Define following methods :*

-constructor

-display, to print values of members

-add_coordinates, to add three such coordinate objects to produce a resultant coordinate object. Generate and handle exception if x, y and z coordinates of the result are zero.

-main, to show use of above methods. **GTU : Summer-13, Marks 7**

Solution :

```
import java.lang.Exception;

class Cartesian extends Exception
{
   double result;
   double x,y,z;
   Cartesian()
   {

   }
   Cartesian(double x,double y,double z)
   {
      this.x=x;
      this.y=y;
      this.z=z;
   }

   void display()
   {
      System.out.println("value of x = "+x);
      System.out.println("value of y = "+y);
      System.out.println("value of z = "+z);
   }

   Cartesian add_coordinates(Cartesian obj1,Cartesian obj2,Cartesian obj3) throws
Cartesian
   {
      Cartesian obj4=new Cartesian();
      obj4.x=obj1.x+obj2.x+obj3.x;
      obj4.y=obj1.y+obj2.y+obj3.y;
      obj4.z=obj1.z+obj2.z+obj3.z;
      if((obj4.x == 0)||(obj4.y == 0)||(obj4.y == 0))
      {
         throw new Cartesian();
      }
      return obj4;
   }
```

```
int sum(int a,int b);
double sum(double a,double b);
int sum(int a,int b,int c);
```

That means, by overloading mechanism, we can handle different number o parameters or different types of parameter by having the same method name Following Java program explains the concept overloading -

Java Program [OverloadingDemo.java]

```
public class OverloadingDemo {
public static void main(String args[]) {
 System.out.println("Sum of two integers      ");
 Sum(10,20); <------------- line A
 System.out.println("Sum of two double numbers      ");
 Sum(10.5,20.4); <------------- line B
 System.out.println("Sum of three integers     ");
 Sum(10,20,30); <------------- line C
}
public static void Sum(int num1,int num2)
{
  int ans;
  ans=num1+num2;
  System.out.println(ans);
}
public static void Sum(double num1,double num2)
{
  double ans;
  ans=num1+num2;
  System.out.println(ans);
}
public static void Sum(int num1,int num2,int num3)
{
  int ans;
  ans=num1+num2+num3;
  System.out.println(ans);
}
}
```

Output

```
F:\test>javac OverloadingDemo.java

F:\test>java OverloadingDemo
Sum of two integers
30
Sum of two double numbers
30.9
```

```
Sum of three integers
60
```

Program Explanation

In above program, we have used three different methods possessing the same n

Note that,

on line A

We have invoked a method to which two integer parameters are passed. T compiler automatically selects the definition **public static void Sum(do num1,double num2)** to fulfil the call. Hence we get the output as 30 which is act the addition of 10 and 20.

on line B

We have invoked a method to which two double parameters are passed. T compiler automatically selects the definition **public static void Sum(do num1,double num2)** to fulfil the call. Hence we get the output as 30.9 which is act the addition of 10.5 and 20.4.

on line C

We have invoked a method to which the three integer parameters are passed. T compiler automatically selects the definition **public static void Sum(int num num2,int num3)** to fulfil the call. Hence we get the output as 60 which is actually addition of 10, 20 and 30.

Review Question

1. *Explain the concept of method overloading.*

3.8 Recursion

Recursion is a programming technique in which the method calls itself repeatedly. The parameters passed to a particular method allow to call itself. Such a method is called **recursive method.**

Following example illustrates the idea of recursion.

A most common example of recursion is use of **factorial,** which is usually used mathematics.

One can define the factorial of some number n as a product of all the integers from n to 1.

For example : If the 5 factorial has to be calculated then, it will be = 5*4*3*2*1 =12

Similarly 3! = 3*2*1 = 6 and the 0! = 1. The exclamation mark is used to denote the factorial. We may write the definition of factorial function as -

```
n! = 1 if n==0
Otherwise,  n! = n*(n-1)*(n-2)*...*1 if n>0
```

If the value of n is any of the 0, 1, 2, 3 or 4 then the definition will be -

```
0! = 1
1! = 1
2! = 2*1
3! = 3*2*1
4! = 4*3*2*1
```

Let us see how the recursive definition of the factorial function is used to evaluate the 5!

```
Step 1 :  5! = 5*4!
Step 2 :      4! = 4*3!
Step 3 :          3! = 3*2!
Step 4 :              1! = 1*0!
Step 5 :                  2! = 2*1!
Step 6 :                      0! = 1.
```

Actually the step 6 is the only step which is giving the direct result. So to solve 5! we have to backtrack from step 6 to step 1, collecting the result from each step. Let us see how to do this.

```
Step 6' : 0! =1
Step 5' : 1! = 1* 0! = 1          from step 6'
Step 4' :  2! = 2*1! = 2          from step 5'
Step 3' :  3! = 3*2! = 6          from step 4'
Step 2' :  4! = 4*3! = 24         from step 3'
Step 1' :  5! = 5*4! = 120        from step 2'
```

Example 3.8.1 *Write a program to find factorial of a given number.*

GTU : May-10, Marks 7

Solution :

```
//////////////////////////////////////////////////////////////////////////////
//Program for finding out the factorial for any given number.
//////////////////////////////////////////////////////////////////////////////
import java.io.*;
import java.util.*;
class Factorial
{
 public int fact(int n)
 {
  int x,y;
```

Solution :

```
////////////////////////////////////////////////////////////////////////////////
/Program to find out the greatest common divisor of the two
/integers.
////////////////////////////////////////////////////////////////////////////////
mport java.io.*;
mport java.util.*;
lass GCD

public int gcd(int a,int b)
{
 int temp,ans;
  if( b<=a && a%b ==0)
        return b;
  else
  {
      if(a<b)
        return(gcd(b,a));
 /*the divisior should be less than divident*/
 else
      ans = a%b;
     return(gcd(b,ans));   //call to recursive function
  }
 }//end of function
 public static void main(String[] args)throws IOException
{
 int ans;
 GCD g=new GCD();
 System.out.println("\n\t\t GCD Of two integers");
 System.out.println("\n Enter first number: ");
 BufferedReader buff=new BufferedReader(new InputStreamReader(System.in));
 int a=Integer.parseInt(buff.readLine());
 System.out.println("\n Enter second number: ");
 int b=Integer.parseInt(buff.readLine());
 ans=g.gcd(a,b);
 System.out.println("\n The gcd of "+a+" and "+b+" is = "+ans);
 }
}
```

Output

```
        GCD Of two integers

Enter first number:
15
Enter second number:
18
The gcd of 15 and 18 is = 3
```

Comparison between Recursion and Iteration

Sr. No.	Iteration	Recursion
1.	Iteration is a process of executing certain set of instructions repeatedly, without calling the self function	Recursion is a process of executing certa set of instructions repeatedly by calling self function repeatedly.
2.	The iterative functions are implemented with the help of for, while, do-while programming constructs.	Instead of making use of for, while, do-while the repetition in code executio obtained by calling the same function a and again over some condition.
3.	The iterative methods are more efficient because of better execution speed.	The recursive methods are less efficient.
4.	Memory utilization by iteration is less.	Memory utilization is more in recursive functions.
5.	It is simple to implement.	Recursive methods are complex to implement.
6.	The lines of code is more when we use iteration.	Recursive methods bring compactness in the program.

3.9 Passing and Returning Object from Method

3.9.1 How to Pass Objects to the Method ?

The object can be passed to a method as an argument. Using dot operator the ob value can be accessed. Such a method can be represented syntactically as -

```
Data_Type name_of_method(object_name)
{
    //body of method
}
```

Following is a sample Java program in which the method **area** is defined parameter passed to this method is an **object.**

Java Program[ObjDemo.java]

```
public class ObjDemo {
    int height;
    int width;
    ObjDemo(int h,int w)
    {
        height=h;
        width=w;
    }
    void area(ObjDemo o)
    {
        int result=(height+o.height)*(width+o.width);
        System.out.println("The area is "+result);
    }
}
class Demo {
 public static vo' main(String args[])
 {
    ObjDemo obj1=new ObjDemo(2,3);
    ObjDemo obj2=new ObjDemo(10,20);
    obj1.area(obj2);
 }
}
```

Method with object as parameter

height=2 and o.height=10
width=3 and o.width=20
Hence,
result=(2+10)*(3+20) =12*23 = 276

Output

```
F:\test>javac ObjDemo.java
F:\test>java Demo
The area is 276
```

Program Explanation

The method, **area** has object **obj2** as argument. And there is another object named **obj1** using which the method area is invoked. Inside the method **area,** the height and width values of invoking object are added with the height and width values of the object to be passed.

When an object needs to be passed as a parameter to the function then constructor is built. Hence we have to define **constructor** in which the values of the object are used for initialization.

3.9.2 How to Return an Object from a Method ?

We can return an object from a method. The data type such method is a class type.

Java Program[ObjRetDemo.java]

```
public class ObjRetDemo {
    int a;
    ObjRetDemo(int val)
    {
        a=val;
    }
    ObjRetDemo fun()
```

height=2 and o.height=10
width=3 and o.width=20
Hence,
result=(2+10)*(3+20) =12*23 = 276

Now using the object of the class one can ccess the members of that class with the help f do operator.

The object has **physical reality** that means bject occupies space in memory.

Consider a case of reference creation. onsider following statements

```
est obj1=new Test();
est obj2;
bj2=obj1;
```

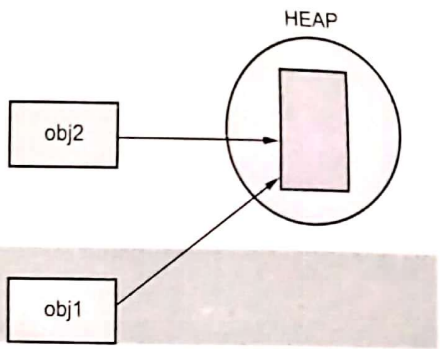


Fig. 3.10.1

Here **obj2** is a reference variable which oints to same object which is pointed by bj1.

Now if we assign

```
obj1=null;
```

Then it does not mean **obj2** is destroyed. This means that obj2 will point to the same bject but **obj1** will not point to any object.

Example 3.10.1 *Give output of the following program :*

```
public class Test {
public static void main(String args[]) {
Count myCount = new Count();
int times=0;
for(int i=0;i<100;i++)
increment(myCount,times);
System.out.println("count is "+myCount.count);
System.out.println("times is "+times);
}
public static void increment(Count c,int times) {
c.count++;
times++;
}
}
class Count {
public int count;
Count(int c){ count=c; }
Count(){ count=1; }
}
```

GTU : Dec.-11, Marks 7

```
}
void calculate_spi()
{
    for(int i=0;i<no_of_sub_registered;i++)
    {
    if(grade_obtained[i].equals("AA"))
            gradePoint[i]=10;
        else if(grade_obtained[i].equals("AB"))
            gradePoint[i]=9;
        else if(grade_obtained[i].equals("BB"))
            gradePoint[i]=8;
        else if(grade_obtained[i].equals("BC"))
            gradePoint[i]=7;
        else if(grade_obtained[i].equals("CC"))
            gradePoint[i]=6;
        else if(grade_obtained[i].equals("CD"))
            gradePoint[i]=5;
        else if(grade_obtained[i].equals("DD"))
            gradePoint[i]=4;
        else if(grade_obtained[i].equals("FF"))
            gradePoint[i]=0;
    sum=sum+sub_credit[i]*gradePoint[i];
    Total_credit=Total_credit+sub_credit[i];
    }
     spi= sum/Total_credit;
     System.out.println("\n SPI= "+spi);
    }
public static void main(String [] args)
{
    Scanner s=new Scanner(System.in);
    Student obj=new Student();
     String str;
    int n = Integer.parseInt(args[0]);
    if(n <= 0)
    {
        System.out.println("No student is registered!!!");
    }
    for(int j=0;j<n;j++)
    {
      System.out.println("\n Enter The record for Student#"+(j+1));
      for(int i=0;i<obj.no_of_sub_registered;i++)
      {
       System.out.println("Subject Code: "+obj.sub_code[i]);
       System.out.println("Credit points: "+obj.sub_credit[i]);
        System.out.println("Enter the Grade: ");
       str=s.nextLine();
       obj.grade_obtained[i]=str;
```

```
    }
    obj.calculate_spi();
  }
 }
}
```

Review Question

1. *What is the use of new operator. Explain the method to create an object using new operator.*

3.11 this Keyword

When a calling object wants to refer its own values then the **this reference** is u
The **this** is a keyword used for making the **this reference.** Using this reference we
refer to class's hidden data fields. For example

```
this.a=a
```

means assign the value of **a** to the data field **a** of calling object.

Java Program [thisRefDemo.java]

```
public class thisRefDemo {
    int height;
    int width;
    thisRefDemo(int h,int w)
    {
        this.height=h;
        this.width=w;
    }
    thisRefDemo()
    {
        this(10,20);
    }
    void area()
    {
        System.out.println("Now, The function is called...");
        int result=height*width;//here this.height and height values are the same
                                //here this.width and width values are the same
        System.out.println("The area is "+result);
    }
}
```

Method1: this reference

Method2: this reference


```
class thisDemo {
public static void main(String args[])
{
    thisRefDemo obj=new thisRefDemo(10,10);
    obj.area();//call the to method
    thisRefDemo obj1=new thisRefDemo();
    obj1.area();//call the to method


}
}
```

Output

```
F:\test>javac thisRefDemo.java

F:\test>java thisDemo
Now, The function is called...
The area is 100
Now, The function is called...
The area is 200
```

Program Explanation

In above program, we have written a simple method **area** for computing the area of rectangle. For calculating the area of rectangle we need the **height** and **width** values. Each time an object is created to invoke the method **area.** While using the first object **obj** the parameterised constructor is used. The values passed as parameter are assigned to the data fields **height** and **width** using **this reference.** Similarly the simple constructor is used to create another object **obj1.** In this constructor the **this** reference is used to pass the values to **height** and **width.**

when we create object **obj1** then

```
    thisRefDemo obj1=new thisRefDemo();

    thisRefDemo()
    {

        this(10,20);

    }

    void area()
    {
        System.out.println("Now, The function is called...");
        int result=height*width;//here this.height and height values are
                                the same
                                //here this.width and width values are the same
```

While creating the object the constructor is invoked

Invoking another constructor

10 gets assigned to height and 20 gets assigned to width

Finalization is the facility provided by the Java for the classes for cleaning up the native resources before the objects are garbage collected. The garbage collector is unable to control the cleaning up of native resources which are used earlier. Then the responsibility of cleaning up those native allocations falls on the object's finalization code.

Thus the purpose of finalization is to clean up the native resources used earlier. The finalize() method must be run before invoking the garbage collector.

3.14 Access Control and Modifiers

GTU : Winter-14, Marks 7

Access modifiers control access to data fields, methods and classes. There are three modifiers used in Java -

- public
- private
- default modifier

public allows classes, methods and data fields accessible from any class.

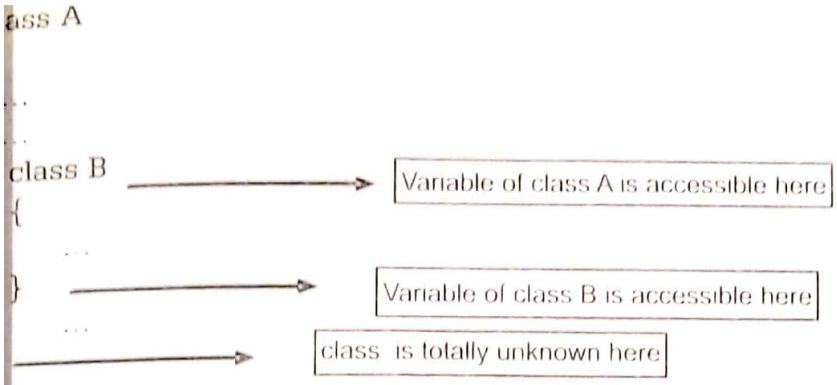
private allows classes, methods and data fields accessible only from within the own class.

If public or private is not used then by default the classes, methods, data fields are assessable by any class in the same package. This is called **package-private** or **package-access.** A package is essentially grouping of classes.

For example :

```
package Test;
public class class1
{
        public int a;
        int b;
        private int c;
        public void fun1() {
        }
        void fun2() {
        }
        private void fun3() {
        }
}
public class class2
{
        void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;//allowed
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//allowed
        obj.fun3();//error:cannot access
        }
}
```

```
package another_Test
public class class3
{
        void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;// error:cannot
access
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2()//error:cannot
access
        obj.fun3()//error:cannot
access
        }
}
```



va Program

```
////////////////////////////////////////////////////////////////////////////////
Demonstrating the concept of nested class
////////////////////////////////////////////////////////////////////////////////
nport java.io.*;
nport java.util.*;
ass A

t a=10;
oid show()

 B obj_B=new B();
 obj_B.display();

lass B

oid display()

 System.out.println("The value of a is = "+a);


ublic static void main(String[] args)

 A obj_A=new A();
 obj_A.show();
```

Output

```
e value of a is = 10
```

Program Explanation

In above program, the class B is within the class A. The class A has one fu
show() which invokes the function of class B. The class B defines a function dis
which invokes the variable **a** of class **A**. That means inner class can access the c
of the outer class.

The access chain will be

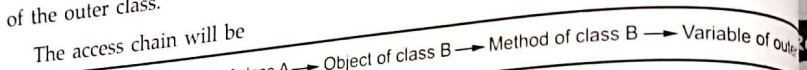


Fig. 3.15.1 Access chain

However if we make some changes in above program as given below then raise compiler error

```
//////////////////////////////////////////////////////////////////
//Demonstrating the concept of nested class
//////////////////////////////////////////////////////////////////
import java.io.*;
import java.util.*;
class A
{
 int a=10;
 void show()
 {
   B obj_B=new B();
   obj_B.display();
 }
 class B
 {
  void display()
  {
    int b=20;
    System.out.println("The value of a is = "+a);
  }
 }
  void find()
  {

    System.out.println("The value of b is = "+b);
  }

  public static void main(String[] args)
  {
   A obj_A=new A();
   obj_A.show();
  }
}
```


The boldfaced code indicates those changes. The compiler error "cannot find symbol" ror will occur for variable b. That means the inner class is not accessible from the uter class (after the closing bracket of inner class)

Thus the nested class B can be defined within the class A.

eview Question

1. *Write Java program to illustrate the concept of nested classes.*

3.16 Inner Class

Inner classes are the nested classes. That means these are the classes that are defined ide the other classes. The syntax of defining the inner class is -

```
ccess_modifier class OuterClass
{
   //code
Access_modifier class InnerClass
{
   //code
}
```

llowing are some properties of inner class -

- The outer class can inherit as many number of inner class objects as it wants.
- If the outer class and the corresponding inner class both are public then any other class can create an instance of this inner class.
- The inner class objects do not get instantiated with an outer class object.
- The outer class can call the private methods of inner class.
- Inner class code has free access to all elements of the outer class object that contains it.
- If the inner class has a variable with same name then the outer class's variable can be accessed like this -

  outerclassname.this.variable_name

There are **four types of inner classes -**

1. Static Member classes 2. Member Inner Classes 3. Local Inner Classes
d 4. Anonymous Inner Classes

3.16.1 Static Member Classes

- This inner class is defined as the static member variable of another class.
- Static members of the outer class are visible to the static inner class.

```
ode
ublic static class InnerClass
{
    //code
}
```

2 Member Inner Classes

- This type of inner class is non-static member of outer class.

3.16.3 Local Inner classes

- This class is defined within a Java code just like a local variable.
- Local classes are never declared with an access specifier.
- The scope inner classes is always restricted to the block in which they declared.
- The local classes are completely hidden from the outside world.

Syntax

```
abstract class A
{
abstract void fun1();
void fun2()
{
System.out.println("A:Infun2");

}
}
class B extends A
{
void fun1()
{
        System.out.println("B:In fun1");
}
}
class C extends A
{
void fun1()
{
        System.out.println("C:In fun1");
}

}
```

This method is so abstract that it has no definition body. This is an **abstract method**.

```
abstract class shape
{
  int base,height;
  double a;
  void initFun()
  {
    base=5;height=6;
  }
  abstract void compute_area(); //pure virtual function

}
class triangle extends shape
{
  public void compute_area()
  {

    a=(base*height)/2;
    System.out.println("\n Area of triangle is "+a);

  }
}
class Rectangle extends shape
{
  public void compute_area()
  {

    a=(base*height);
    System.out.println("\n Area of rectangle is "+a);

  }
}

public class Test
{
  public static void main(String[] args)
  {

    triangle obj1=new triangle();
    obj1.initFun();
    obj1.compute_area();
    Rectangle obj2=new Rectangle();
    obj2.initFun();
    obj2.compute_area();
```

- *A no-arg constructor that creates a default fan.*
- *A parameterized constructor initializes the fan objects to given values.*
- *A method named display() will display description for the fan. If the fan is on, the display() method displays speed, color and radius. If the fan is not on, the method returns fan color and radius along with the message "fan is off".*

Write a test program that creates two Fan objects. One with default values and the other with medium speed, radius 6, color brown, and turned on status true. Display the descriptions for two created Fan objects. **[Refer example 3.6.4]** [7]

Q.3 *Define the Rectangle class that contains :*

Two double fields x and y that specify the center of the rectangle, the data field width and height ,A no-arg constructor that creates the default rectangle with (0,0) for (x,y) and 1 for both width and height.

A parameterized constructor creates a rectangle with the specified x, y, height and width.

A method getArea() that returns the area of the rectangle.

A method getPerimeter() that returns the perimeter of the rectangle.

A method contains(double x, double y) that returns true if the specified point (x, y) is inside this rectangle.

Write a test program that creates two rectangle objects. One with default values and other with user specified values. Test all the methods of the class for both the objects. **[Refer section 3.6.5]** [7]

Q.4 *State whether any error exists in the following code. If so, correct the error and give output.*

```
class Test {
public static void main(String args[]) {
A a = new A();
a.print();
}
}
class A {
String s;
A(String s) {
this.s=s;
}
public void print() {
System.out.println(s);
}
```

} **[Refer section 3.6.6]** [7]

Q.5 *Give output of the following program :*

```
public class Test {
public static void main(String args[]) {
Count myCount = new Count();
int times=0;
for(int i=0;i<100;i++)
increment(myCount.times);
System.out.println("count is "+myCount.count);
System.out.println("times is "+times);
}
public static void increment(Count c,int times) {
c.count++;
times++;
}
}
class Count {
public int count;
Count(int c){ count=c; }
Count(){ count=1; }
}
```

[Refer example 3.10.1]

Summer 2013

Q.6 *Declare a class called coordinate to represent 3 dimensional carte coordinates (x, y and z). Define following methods :*

-constructor

-display, to print values of members

-add_coordinates, to add three such coordinate objects to produce a result coordinate object. Generate and handle exception if x, y and z coordinates of result are zero.

-main, to show use of above methods. **[Refer example 3.6.7]**

Q.7 *It is required to compute SPI (Semester Performance Index) of n students of a college for their registered subjects in a semester.*

Declare a class called student having following data members :

id_no, no_of_subjects_registered, subject_code, subject_credits, grade_obtai and spi.

-Define constructor and calculate_spi methods.

-Define main to instantiate an array for objects of class student to process data of n students to be given as command line arguments. **[Refer example 3.10.2]** [7]

Winter 2014

Q.8 *There is no destructor in Java. Justify.* **[Refer example 3.6.2]** [3]

Q.9 *Discuss public, private,protected and default access modifier with example.* **[Refer section 3.14]** [7]

❑❑❑

4 Inheritance and Interfaces

10

Syllabus

Use of inheritance, Inheriting data members and methods, Constructor in inheritance, Multilevel inheritance – Method overriding handle multilevel constructors – Super keyword,Stop Inheritance - Final keywords, Creation and implementation of an interface, Interface reference, Instance of operator, Interface inheritance, Dynamic method dispatch ,Understanding of java object class,Comparison between abstract class and interface, Understanding of system.out.println – statement.

Contents

4.1 Use of Inheritance

4.2 Inheriting Data Members and Methods

4.3 Constructor in Inheritance

4.4 Multilevel Inheritance ..... **Dec.-10** ..... Marks 7

4.5 Method Overriding ..... **May-11, 12, Winter-13,14** .. Marks 7

4.6 Handle Multilevel Constructor

4.7 Super Keyword

4.8 Final Keywords

4.9 Concept of Interface ..... **Summer-13** ..... Marks 7

4.10 Extending Interface

4.11 Creation and Implementation of an Interface

4.12 Interface Inheritance ..... **Dec.-10, 11,Winter-12,14** .. Marks 7

4.13 Instanceof Operator

4.14 Polymorphism **May-12,Summer-13,14**
..... **Dec.-11** ..... Marks 7

4.15 Dynamic Binding

4.16 Understanding of Java Object Class

4.17 Comparison between Abstract Class and Interface ..... **Winter-14** ..... Marks 7

4.18 Understanding of System.out.println Statement

4.19 University Questions with Answers

```
class A

    int a;
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}
class B extends A

    int b;
    void set_b(int i)
    {
        b=i;
    }
    void show_b()
    {
        System.out.println("The value of b= "+b);
    }
    void mul()
```

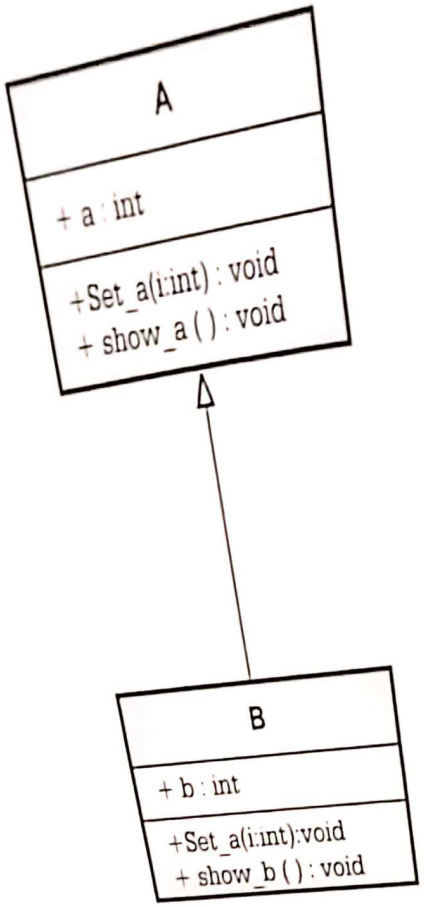


Fig. 4.2.1 Single Inheritance

In some programming languages, we can derive many classes from a single b class. This type of inheritance is called **multiple inheritance.** But in Java, mult inheritance is not allowed. In other words, Java class may inherit directly from only o superclass. This type of inheritance is called as **single inheritance.** That means only o parent is allowed to have in Java.

4.3 Constructor in Inheritance

GTU : Dec.-11, Marks 7

The constructor in inheritance can be used using the keyword **super**. A subclass c invoke the constructor method defined by the superclass. The syntax for the **super** w be

GTU : Dec.-10, M

4.4 Multilevel Inheritance

The multilevel inheritance is a kind of inheritance in which the derived cla derives the subclasses further.

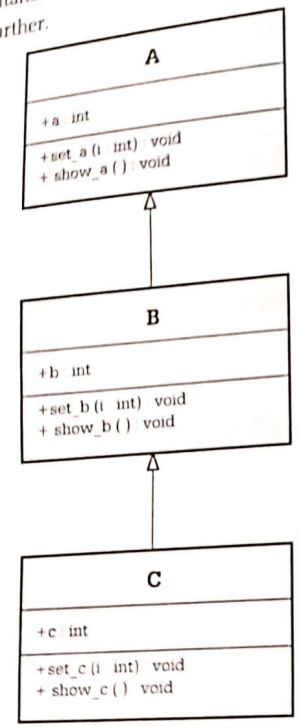


Fig. 4.4.1 Multilevel inheritance

In the following program, we have created a base class A from which the su B is derived. There is a class C which is derived from class B. In the function ma can access any of the field in the class hierarchy by creating the object of class C.

Java Program[MultiLvlInherit.java]

```
class A
{
 int a;
 void set_a(int i)
 {
  a=i;
 }
 void show_a()
 {
 System.out.println("The value of a= "+a);
```



```
 }
}
class B extends A
{
 int b;
 void set_b(int i)
 {
  b=i;
 }
 void show_b()
 {
  System.out.println("The value of b= "+b);
 }
}
class C extends B
{
 int c;
 void set_c(int i)
 {
  c=i;
 }
 void show_c()
 {
  System.out.println("The value of c= "+c);
 }
 void mul()
 {
  int ans;
  ans=a*b*c;
  System.out.println(" The value of ans= "+ans);
 }
}

class MultiLvlInherit
{
 public static void main(String args[])
 {
  A obj_A=new A();
  B obj_B=new B();
  C obj_C=new C();
  obj_C.set_a(10);
  obj_C.set_b(20);
  obj_C.set_c(30);
  obj_C.show_a();
  obj_C.show_b();
  obj_C.show_c();
```

```
  obj_C.mul();
 }
}
```

Output

```
The value of a= 10
The value of b= 20
The value of c= 30
The value of ans= 6000
```

Example 4.4.1 *Explain single level and multiple inheritances in java. Write a program to demonstrate combination of both types of inheritance as shown in Fig. 4.4.2. i.e hybrid inheritance.* **GTU : Dec.-10, Marks 7**

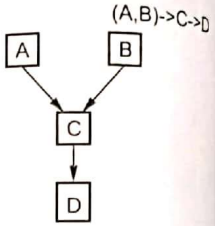


Fig. 4.4.2

Solution :

```
class A
{
 int a=10;

}
interface B
{
 int b=20;

}
class C extends A implements B
{
 int c;
 int mul()
 {
  c=a*b;
  return c;
 }

}
class D extends C
```



```
{
 void sum()
 {
  System.out.println("Adding all the three variables");
  int d=a+b+mul();
  System.out.println(d);
 }

}
class Demo1
{
 public static void main(String[] args)
 {
  C obj1=new C();
  D obj2=new D();
  System.out.println("Multiplying two variables");
  System.out.println(obj1.mul());
  obj2.sum();

 }
}
```

Output

```
Multiplying two variables
200
Adding all the three variables
230
```

Example 4.4.2 *Declare a class called employee having employee_id and employee_name as members. Extend class employee to have a subclass called salary having designation and monthly_salary as members. Define following :*

- *Required constructors.*
- *A method to find and display all details of employees drawing salary more than 20000/-.*
- *Method main for creating an array for storing these details given as command line arguments and showing usage of above methods.* **GTU : May-11, Marks 7**

Solution :

```
class employee
{
  int employee_id;
  String employee_name;
}
class salary extends employee //derived class
{
  String designation;
  double monthly_salary;
```

```
salary()        //default constructor
{
}
salary(int employee_id,String employee_name,String designation,double
monthly salary) //parameterised construct
{
    this.employee_id = employee_id;
    this.employee_name = employee_name;
    this.designation = designation;
    this.monthly_salary = monthly_salary;
}
void fun(String emp[][]) //Function displaying salary > 20000
{
    for(int i=0;i<emp.length;i++)
    {
        if( Double.parseDouble(emp[i][3]) > 20000 )
        {
            System.out.println("\nEmployee Drawing Salary More than    20000/- ");
            System.out.println("Id = "+emp[i][0]);
            System.out.println("Name = "+emp[i][1]);
            System.out.println("Designation = "+emp[i][2]);
            System.out.println("Salary = "+emp[i][3]);
            System.out.println("-------------------------------");
        }
    }
}
public class employeetest
{
public static void main(String args[])
{
    salary obj[] = new salary[5];
    String Details[][];
    obj[0] = new salary(1,"AAA","Accountant",12000);
    obj[1] = new salary(2,"BBB","Manager",31000);
    obj[2] = new salary(3,"CCC","Executive",25000);
    obj[3] = new salary(4,"DDD","CEO",50000);
    obj[4] = new salary();
    try
    {
        obj[4].employee_id = Integer.parseInt(args[0]);
        obj[4].employee_name = args[1];
        obj[4].designation = args[2];
        obj[4].monthly_salary = Double.parseDouble(args[3]);
    }
    catch(ArrayIndexOutOfBoundsException e)
    {
        System.out.println("Exception "+e);
    }
    Details = new String[obj.length][4];
    for(int i=0;i<obj.length;i++)
```



```
        Details[i][0] = String.valueOf(obj[i].employee_id);
        Details[i][1] = obj[i].employee_name;
        Details[i][2] = obj[i].designation;
        Details[i][3] = String.valueOf(obj[i].monthly_salary);
    }
    obj[4].fun(Details);
}
}
```

Output

```
E:\Mest>javac employeetest.java

E:\Mest>java employeetest 5 EEE Manager 20000

Employee Drawing Salary More than    20000/-
Id = 2
Name = BBB
Designation = Manager
Salary = 30000.0
-------------------------------

Employee Drawing Salary More than Rs. 20000/-
Id = 4
Name = DDD
Designation = CEO
Salary = 50000.0
-------------------------------
```

4.5 Method Overriding

GTU : Nov-11, 12, Winter-13,14,

Method overriding is a mechanism in which a subclass inherits the meth superclass and sometimes the subclass modifies the implementation of a defined in superclass.

The method of superclass which gets modified in subclass has the same n type signature. The overridden method must be called from the subclass following Java Program, in which the method named as **fun** ( ) in which a p with some value is modified in the derived class. When an overridden method from within a subclass, it will always refer to the version of that method re-de the subclass. The version of the method defined by the superclass will be hidden

Java Program[OverrideDemo.java]

```
class A
{
    int a=0;
    void fun(int i)
    {
```

2. ... overridden only if it is accessible. Hence private method cannot be overridden.

3. The static method can be inherited but cannot be overridden.

4. Method overriding occurs only when the name of the two methods and their type signatures is same.

Example 4.5.1 *Describe abstract class called Shape which has three subclasses say Triangle,Rectangle,Circle. Define one method area() in the abstract class and override this area() in these three subclasses to calculate for specific object i.e. area() of Triangle subclass should calculate area of triangle etc. Same for Rectangle and Circle.*

GTU : May-12, Marks 7

Solution :

```
abstract class Shape
{
   double Length,Breadth,radius;
   abstract double area();
}
class Triangle extends Shape
{
   Triangle(double l,double b)
   {
      Length=l;
      Breadth=b;
   }
   double area()
   {
      System.out.print("Area Of Triangle : ");
      return ((Length*Breadth)/2);
   }
}
class Rectangle extends Shape
{
   Rectangle(double l,double b)
   {
      Length=l;
      Breadth=b;
   }
   double area()
   {
      System.out.print("Area Of Rectangle : ");
      return (Length*Breadth);
   }
}
class Circle extends Shape
{
   Circle(double r)
   {
```

```
    radius = r;
  }
  double area()
  {
    System.out.print("Area Of Circle : ");
    return (3.14*radius*radius);
  }
}
class testclass
{
  public static void main(String args[])
  {
    Triangle tri = new Triangle(10,20);
    Rectangle rect = new Rectangle(50,60);
    Circle cir = new Circle(10);
    Shape obj;
    obj = tri;
    System.out.println(obj.area());
    System.out.println();
    obj = rect;
    System.out.println(obj.area());
    System.out.println();
    obj = cir;
    System.out.println(obj.area());
  }
}
```

Output

```
Area Of Triangle : 100.0

Area Of Rectangle : 3000.0

Area Of Circle : 314.0
```

Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
The method overloading occurs at **compile time.**	The method overriding occurs at the run time **execution time.**
In case of method overloading **different number of parameters** can be passed to the function.	In function overriding the **number of parameters** that are passed to the function are the **same.**
The overloaded functions may have **different return types.**	In method overriding all the methods will have **the same return type.**
Method overloading is performed within a class.	Method overriding is normally performed between two classes that have inheritance relationship.
Example - Refer section 3.7	Example - Refer section 4.5

Review Questions

1. *Explain method overriding and method overloading with the help of examples*
   GTU : May-11, May-12,Winter-13, Marks 7
2. *Differentiate between method overloading and method overriding with example*
   GTU : Winter-14, Marks 7

4.6 Handle Multilevel Constructor

A constructor invokes its superclass's constructor explicitly and if such explicit call to superclass's constructor is not given then compiler makes the call using **super()** as a first statement in constructor. Normally a superclass's constructor is called before the subclass's constructor. This is called **constructor chaining.** Thus the calling of constructor occurs from **top to down.**

In the following Java program, there are three classes namely: A, B and C.

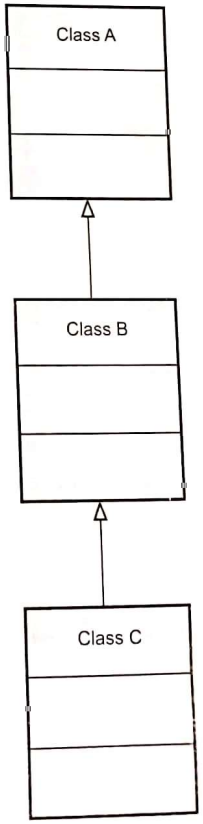


Fig. 4.6.1 Constructor chaining

```
lass A

public A()
        {
                System.out.println("1. Statement in class A");
        }
        public B(int i)
        {
        System.out.println(i+". Statement in class B");
        }
}
public class C extends B
{
        public static void main(String arg[])
        {
         new C();
        }
        public C()
        {
                System.out.println("4. Statement in class C");
        }
```

Output

```
:\test>javac C.java

:\test>java C
. Statement in class A
. Statement in class B
. Statement in class B
. Statement in class C
```

Program Explanation

In the above program, we have used the keyword **super** for invoking the me **show_a()** which belongs to the superclass A. For that purpose we have **super.show_a()**. But if class A contains some method *something()*, the class C is inhe from class B, class B is inherited from class A, then to invoke *something()* we cannot **super.super.something()** because this chain is supposed to be illegal in Java.

Review Question

1. Explain super keyword by giving examples. **GTU : Dec.-10, Marks**

4.8 Final Keywords

The final keyword can be applied at three places -

- For declaring variables
- For declaring the methods
- For declaring the class

4.8.1 Final Variables and Methods

A variable can be declared as final. If a particular variable is declared as final the cannot be modified further. The final variable is always a constant. For example -

```
final int a=10;
```

The final keyword can also be applied to the method. When final keyword is appl to the method, the method overriding is avoided. That means the methods those declared with the keyword **final** cannot be overridden.

Consider the following Java program which makes use of the keyword **final** declaring the method -

```
class Test
{
final void fun()
{
  System.out.println("\n Hello, this function declared using final");
}
}
class Test1 extends Test
{
final void fun()
{
  System.out.println("\n Hello, this another function");
}
}
```


Output

```
D:\>javac Test.java
Test.java:10: fun() in Test1 cannot override fun() in Test, overridden method is final
 final void fun()
            ^
1 error
```

Program Explanation

The above program, on execution shows the error. Because the method **fun** is declared with the keyword **final** and it cannot be overridden in derived class.

4.8.2 Final Classes to Stop Inheritance

GTU : Dec.-11, Marks 7

If we declare particular class as final, no class can be derived from it. Following Java program is an example of final classes.

```
final class Test
{
 void fun()
 {
  System.out.println("\n Hello, this function in base class");
}
}
class Test1 extends Test
{
 final void fun()
 {
  System.out.println("\n Hello, this another function");
 }
}
```

Output

```
D:\>javac Test.java
Test.java:8: cannot inherit from final Test
class Test1 extends Test
                    ^
1 error
```

4.9 Concept of Interface

GTU : Summer-13, Marks 7

An interface is similar to a class but there lies some difference between the two.

Class	Interface
The class is denoted by a keyword **class**	The interface is denoted by a keyword **interface**
The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.	The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.

```
interface Interface_name2 extends interface_name1
{
        } Body of interface
}
```

- For example

```
interface A
{
   int val=10;
}
interface B extends A
{
  void print_val();
}
```

That means in **interface B** the **display** method can access the value of variable **val.**

Similarly more than one interfaces can be extended.

```
interface A
{
   int val=10;
}
interface B
{
   void print_val();
}
interface C extends A,B
{
     ...
     ...
}
```

Even-though methods are declared inside the interfaces and sub-interfaces, these methods are not allowed to be defined in them. Note that methods are defined only in the classes and not in the interfaces.

4.11 Creation and Implementation of an Interface

- It is necessary to create a class for every interface.
- The class must be defined in the following form while using the interface

```
lass Class_name extends superclass_name
mplements interface_name1,interface_name2,...

  //body of class
```

In above program, the interface **my_interface** declares only one method i.e. **my_method.** This method can be defined by class **A**. There is another method which is defined in class **A** and that is **another_method.** Note that this method is not declared in the interface **my_interface.** That means, a class can define any additional method which is not declared in the interface.

- The design various ways by which the interface can be implemented by the class is represented by following Fig. 4.11.1.

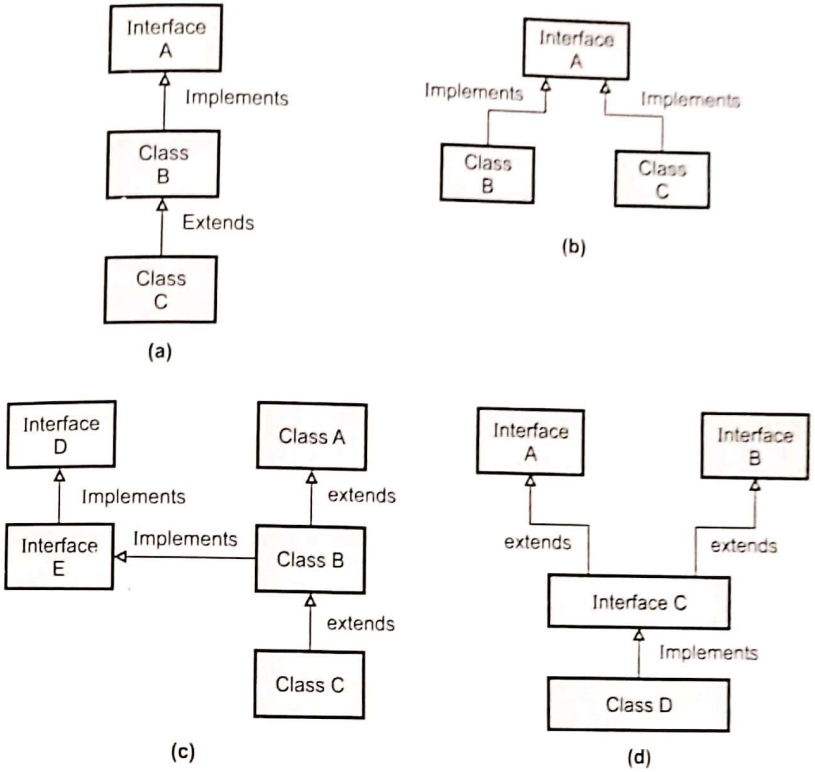


Fig. 4.11.1 Design of ways of interface implementation

4.12 Interface Inheritance

GTU : Dec.-10, 11,Winter-12, 14, Marks 7

Multiple inheritance is a kind of inheritance in which more then one classes are derived from the same base class

Multiple inheritance is **not possible in Java.** But still it can be achieved with the help of **interface**. Following Program illustrates it.

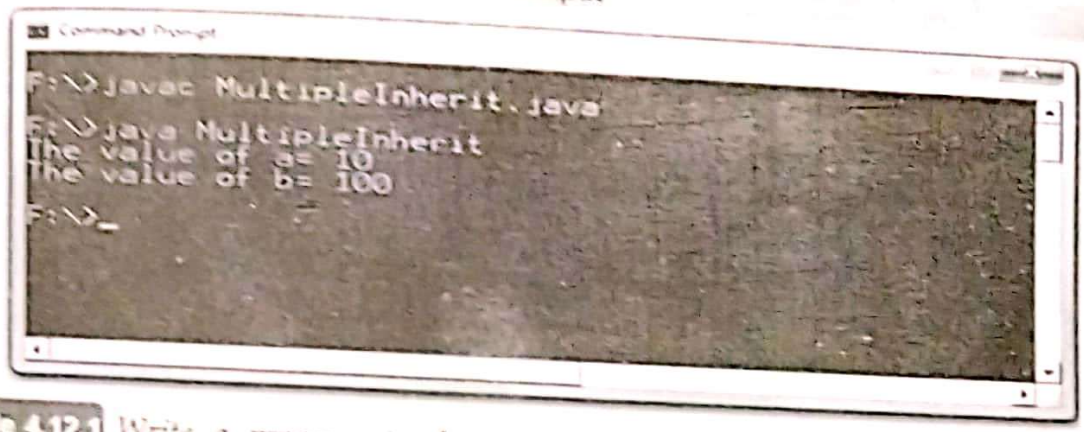
Step 1 : Create an interface in a separate java file. Name it as **interface1.java**, write the following code in it and just save it.

```
public static void main(String args[])
{
  interface1 obj_A = new A();
  interface1 obj_B = new B();
  obj_A.set_val(10);
  obj_B.set_val(20);
  obj_A.show_val();
  obj_B.show_val();
}
}
```

Create the object instances using the interface

Using the objetcs of derived class the method defined in base class is used

Output



Example 4.12.1 *Write a program to demonstrate the multipath inheritance for the classes having relations as shown in Fig. 4.12.2. A->(B,C)->D*

GTU : Dec.-10, Winter-14, Marks 7

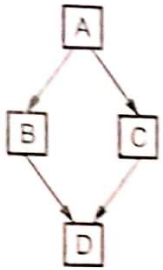


Fig. 4.12.2

Solution :

```
interface A
{
  int a=10;
}
```

```
int a=10;
int b=20;
int c;
abstract void addition(int x,int y);
void subtraction()
{
c=b-a;
System.out.println("Subtraction of 20 and 10: "+c);
}
}
class B extends A
{
void addition(int x,int y)
{
int z=x+y;
System.out.println("addtion function in class B: "+z);
}
}
class C extends A
{
void addition(int x,int y)
{
int z=x+y;
System.out.println("addtion function in class C: "+z);
}
}
public class AbstractDemo
{
public static void main(String[] args)
{
B b=new B();
C c=new C();
System.out.println("----------------------------------------");
b.addition(10,20);
b.subtraction();
System.out.println("----------------------------------------");
c.addition(100,200);
c.subtraction();
System.out.println("----------------------------------------");
}
}
```

Output

```
----------
addtion function in class B: 30
Subtraction of 20 and 10: 10
----------
```

...ction function in class C. 300
...traction of 20 and 10. 10

...ple 4.12.3 *Write a program that illustrates interface inheritance. Interface P is ex... by P1 and P2. Interface P12 inherits from both P1 and P2.Each interface declares ... constant and one method. Class Q implements P12.Instantiate Q and invoke each ... methods. Each method displays one of the constants.* **GTU : Winter-14, Mar...**

Solution :

```
interface P
{
    public static final int p = 10;
    void Method_P();
}
interface P1 extends P
{
    public static final int p1 = 20;
    void Method_P1();
}
interface P2 extends P
{
    public static final int p2 = 30;
    void Method_P2();
}
interface P12 extends P1,P2
{
    public static final int p12 = 40;
    void Method_P12();
}
class Q implements P12
{
    public void Method_P12()
    {
        System.out.println("In the method of P12 and Constant is: "+p12);
    }
    public void Method_P1()
    {
        System.out.println("In the method of P1 and Constant is: "+p1);
    }
    public void Method_P2()
    {
        System.out.println("In the method of P2 and Constant is: "+p2);
    }
    public void Method_P()
    {
```

```
class LivingBodies extends Object
{
}
class Human extends LivingBodies
{
}
class Boy extends Human
{

}
public class instanceofDemo
{
      public static void main(String[] args)
      {
       Boy B1=new Boy();
       System.out.println("The value of object B1: "+B1);
       Human H=B1;
       if(H instanceof Boy)
       {
       System.out.println("It is now safe to downcast the object");
       Boy B2=(Boy)H;
       System.out.println("The value of object B2: "+B2);
       }
      }
}
```

Output

```
F:\test>javac instanceofDemo.java

F:\test>java instanceofDemo
The value of object B1: Boy@42e816
It is now safe to downcast the object
The value of object B2: Boy@42e816
```

Note that **instanceof** is a keyword in which every letter is in lowercase.

: May-12,Summer-13,14, Dec.-11, M

ws to have many forms of the
A() may take an instance of an
he same method A() may take
ethod belonging to another class.
e superclass A and build a

x = new A( )

fun (object x)

x = new C( )

olymorphism

responding method is invoked by

Java program -

```
    }
}
class B extends A
{
    public String toString()
    {
        return "B";
    }
}
class C extends B
{
    public String toString()
    {
        return "C";
    }
}
public class PolymorphismDemo
{
    public static void main(String[] args) {
    fun(new C());//invokes the method toString() of class C
    fun(new B());//invokes the method toString() of class B

    fun(new A());//invokes the method toString() of class A

    }
    public static void fun(Object x) {
    System.out.println(x.toString());
    }
}
```

Output

```
F:\test>javac PolymorphismDemo.java

F:\test>java PolymorphismDemo
C
B
A
```

Program Explanation

The same function **fun** is used to execute the methods of various class. The selection of the method of the class (i.e. toString method) depends upon the instance of the object which is passed to the function **fun.** Which implementation is to be used is determined dynamically by the Java Virtual Machine. This mechanism is called **dynamic binding**.

```
nterface MyInterface

    void display();

lass A implements MyInterface

    public void display()  //display method defined in class A
    {
            System.out.println("Display Method:In class A");
     }

lass B implements MyInterface

    public void display()//display method defined in class B
    {
            System.out.println("Display Method:In class B");
    }

ublic class PolyInterfaceDemo

    public static void main(String args[])
    {
        MyInterface obj;
        obj = new A();
        obj.display();
        obj = new B();
        obj.display();
     }
}
```

Output

```
Display Method:In class A
Display Method:In class B
```

```
class book
{
  private String[] author_name = {"Puntambekar","Godse"};
  void display()
  {
    System.out.println("Author names ");
    for(int i=0;i<author_name.length;i++)
    {
      System.out.println("\t" + author_name[i]);
    }
  }

class book_publication extends book
{
  private String[] title = {"Java Programming","Compiler Design","DAA"},{"Digital
Electronics","Computer Graphics","Microprocessor"}};
  void display()
  {
    System.out.println("Books published by Given Author are...");
    for(int i=0;i<3;i++)
    {
      System.out.println(title[][i]);
    }
  }
}

class paper_publication extends book
{
  private String[][] title ={{"AAA","BBB"},{"XXX","YYY"}};
  void display()
  {
    System.out.println("Papers published by Given Author are...");
```

```
class A extends Object
  {
  }
 class B extends A
  {
  }
class ObjectClassDemo
  {
public static void main(String args[])
    {
     A obj=new A();
   System.out.println("obj: "+obj);
System.out.println("obj.toString(): "+obj.toString());
    }
  }
```

Output

```
F:\ test>javac ObjectClassDemo.java

F:\ test>java ObjectClassDemo
obj: A@3e25a5
obj.toString(): A@3e25a5
```

Here A denotes the class of the object

This part denotes the hexadecimal address.

```
class A extends Object
  {
 public String toString()// Method is overridden
    {
    String str="Hello Friends!!!";
      return str;
    }
  }
 class B extends A
  {
  }
class ObjectClassDemo
  {
public static void main(String args[])
    {
     A obj=new A();
   System.out.println("obj: "+obj);
   System.out.println("obj.toString():"+obj.toString());
    }
  }
```

Output

```
F:\test>javac ObjectClassDemo.java

F:\ test>java ObjectClassDemo
obj: Hello Friends!!!
obj.toString(): Hello Friends!!!
```

Fig. 4.16.1 Invoking toString() method

```
class A extends Object
{
        int a=10;
        public boolean equals(Object obj)
        {
          if(obj instanceof B) { return a==((B)obj).b; }
          else return false;
        }

class B extends A

int b=10;

class ObjectClassDemo1

        public static void main(String args[])
        {
                A obj1=new A();
                B obj2=new B();
                System.out.println("The two values of a and b are equal: "+obj1.equals(x
        }
```

Output

```
\test>javac ObjectClassDemo1.java

\test>java ObjectClassDemo1
The two values of a and b
```

Contents

5.1 Defining a Package ..... **May-11** ..... Marks 7

5.2 Creating and Accessing Package ..... **Winter-13** ..... Marks 7

5.3 Use of Package

5.4 Adding class to a Package

5.5 CLASSPATH

5.6 Import Statement

5.7 JAVA API Package ..... **May-12** ..... Marks 7

5.8 Static Import

5.9 Access Control

5.10 University Questions with Answers

5.1 Defining a Package

Creating a package is very simple. Just include the command **package** at beginning of the program. The syntax for declaring the package is

```
package name_of_package
```

This package statement defines the name space in which the classes are stored. If omit the package then the default classes are put in the package that has no name.

Basically Java creates a directory and the name of this directory becomes the pack name.

For example - In your program, if you declare the package as -

```
package My_Package;
```

then we must create the directory name **My_Package** in the current work directory and the required classes must be stored in that directory. Note that the na the package and the name of the directory must be the same. This name is ca nsitive.

We can create hierarchy of packages. For instance if you save the required class fi in the subfolder **MyPkg3** and the path for this subfolder C:\MyPkg1\MyPkg2\MyPkg3 then the declaration for the package in your ja program will be -

```
package MyPkg1.MyPkg2.MyPkg3;
```

Review Question

1. *Explain packages and interfaces by giving examples.* **GTU : May-11, Marks 7**

5.2 Creating and Accessing Package

GTU : Winter-1

Following are the steps which show ***how to create a package in Java.***

Step 1 : Create a folder named My_Package and write the following code in a file.

```
package My_Package; //include this package at the beginning
public class A
{
int a;
void set_val(int n)
{
    a=n;
}
isplay()
```



```
System.out.pri
}
}
class PackageDemo
{
public static void main(String args[]) throws NoClassDefFoundError
{
A obj=new A();
obj.set_val(10);
obj.display();
}
}
```

Save it by the name **PackageDemo.java.** This file must be saved within the directory **My_Package.**

Step 2 : Compile the above java program as follows -

```
F:\test>cd My_Package

F:\test\My_Package>javac A.java
```

Now, make sure that the class file **A.class** and **PackageDemo.class** gets generated in the folder **My_Package.**

Step 3 : Now execute the program using the command to get an output. Using the package name and dot operator we can execute the program.

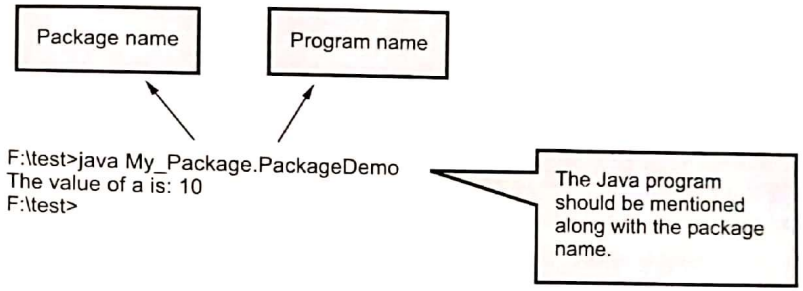


Fig. 5.2.1

This helps to find the CLASSPATH for the executable file. If there is an hierarchy of packages then it can be mentioned by names of packages separated by dots. For example

```
package.my_package.another_package;
```

Note that one of the class in the java package must be **public**.

The main **advantage** of using packages is that we can **arrange** the **systematically.**

Review Question

1. *What is package ?Explain steps to create a package with example* **GTU : Winter-13, Mark**

5.3 Use of Package

In this section we discuss **how to develop a program** which makes use the from other package.

Step 1 : Create a folder named **My_Package.**

Step 2 : Create one class which contains two methods. We will store this class in named **A.java**. This file will be stored in a folder **My_Package**. The code for this will be as follows-

Java Program[A.java]

```
package My_Package; //include this package at the beginning
public class A
{
int a;
public void set_val(int n)
{
    a=n;
}
public void display()
{
System.out.println("The value of a is: "+a);
}
}
```

Note that the class,and the methods defined must be **public**

Note that this class contains two methods namely- **set_val** and **display**. By the method we can assign some value to a variable. The **display** function displays stored value.

Step 3 : Now we will write another java program named **PackageDemo.java** program will use the methods defined in class **A**. This source file is also stored subdirectory **My_Package**. The java code for this file is-

Java Program[PackageDemo.java]

```
import My_Package.A;  //The java class A is referenced here by import statement
class PackageDemo
{
public static void main(String args[]) throws NoClassDefFoundError
{
 A obj=new A(); //creating an object of class A
 obj.set_val(10); //Using the object of class A, the methods present
 obj.display();  //in class A are accessed
}
}
```

Step 4 : Now, open the command prompt and issue the following commands in orde to run the package programs

```
D:\>set CLASSPATH=.;D:\;
D:\>cd My_Package
D:\My_Package>javac A.java
D:\My_Package>javac PackageDemo.java
D:\My_Package>java PackageDemo
The value of a is: 10
D:\My_Package>
```

Setting the class path

Creating the classes A.class anf package Demo.Class files

Running the test program which uses the class **A.Class** stored in another file

5.4 Adding class to a Package

We can add some class in the existing package. For that purpose following steps ar followed-

Step 1 : Write a class (in a separate java file)which is to be added in the existin package. Suppose I want to add the class **B** in the existing package **My_Package**. I wil create a **B.java** file and store the code in it as follows-

Java Program[B.java]

```
package My_Package; //include this package at the beginning
public class B
{
 public void show()
{
 System.out.println("class B is now added in the package");
}
}
```

```
D:\My_Package>javac A.java

D:\My_Package>javac B.java

D:\My_Package>javac PackageDemo.java

D:\My_Package>java PackageDemo
The value of a is: 10
class B is now added in the package
```

Some methods
char readChar() - It reads input character and returns the character value.
boolean readBoolean()- It reads input byte if the input is non zero then it returns true and if input is zero then it returns false.
string readLine()- It reads the next line of the input text
readInt(),readDouble(),readFloat() - These functions are for reading the input and returning int,double and float values respectively.
Void write(byte bt[])- This method is for writing the bytes in the array 'bt' to output stream. *void writeBoolean(boolean val)* -This method is for writing the boolean value 'val' to output stream Similarly methods writeChar(),writeInt, writeDouble() methods are for writing the corresponding type of data to the output stream.
int read()-It reads the byte of data
int read(byte bt[])-It reads into array of bytes 'bt'

Classes

Name	Description
BufferedInputStream	For creating buffered input stream
BufferedOutputStream	For creating buffered output stream
BufferedReader	Reads the text from character input stream by buffering th characters.
BufferedWriter	Writes the text to character output stream by buffering the characters.
ByteArrayInputStream	For storing the bytes read from the stream into the interna buffer this class is used.
ByteArrayOutputStream	For writing the bytes into the buffer as an output stream t class is useful.
DataInputStream	This class helps to read data of primitive data types from underlying input stream.
DataOutputStream	This class helps to write data of primitive data types to an output stream.
FileReader	This class is for reading the character files.
FileWriter	This class is for writing the character files.
InputStream	This class is for providing the input stream of bytes.
InputStreamReader	This class is responsible for reading the bytes and to conve to character type.
OutputStream	This class is for providing the input stream of bytes.

Exceptions :

3. java.lang

Purpose : This package provides core classes and interfaces used in design of Java language.

Interfaces :

Name	Description	Some methods
charSequence	This interface is for getting sequence of characters. The read-only access to character sequence is provided by this method.	int charAt(*int i*)- It returns the position of the character at specified index. String toString()- This method returns the sequence of characters. int length() -It returns length of string
Runnable	A class whose instance can be executed by a thread implements a Runnable interface.	

Classes :

Class	For representing classes and interfaces [illegible] application this Class is used.
ClassLoader	For handling loading of the classes this object is used.
Compiler	This class is for providing support compiler related activities.
Double	It's a wrapper class for Double values.
Float	It's a wrapper class for Float values.
Integer	It's a wrapper class for integer values.
Long	It's a wrapper class for Long values.
Math	This class provides the support for the operations such as square root, exponentiation, and logarithm.
Object	The class Object is at the top of class hierarchy.
Runtime	For linking the Java application program with its runtime environment the instance of class Runtime is used.
StringBuffer	For handling the dynamic sequence of characters StringBuffer class is used
String	It's a wrapper class for String values.
System	This is the most commonly used class in which many important fields and methods are defined. For example err, in and out are the most commonly used fields in this class.
Thread	A Thread class creates a thread for a separate execution of the program.

Exceptions :

Object Oriented		
Adjustable	This interface is for adjustable numerical value which is contained within some range.	void *addAdjustmentListener(AdjustmentListeneL)* -For listening adjustment events the listener is added by this method. *int getMinimum()*- To obtain the minimum value of adjustable object this method is used. *int getValue()*-To obtain the value of adjustable object this method is used. *void setMaximum(int Max)* - This method is for setting maximum value to adjustable object. *void setMinimum(int Min)* - This method is for setting minimum value to adjustable object.
LayoutManager	For the container class which is for layout this interface is used.	void *layoutContainer(Container Root)* - For laying out the specific container this method is used. Similarly *addLayoutComponent()* and *removeLayoutComponent()* methods are used for adding or removing specific components from the layout.
Paint	This interface is for deciding the color pattern for Graphics2D operations.	
PrintGraphics	Its an abstract class for Print Graphics context.	
Shape	This interface defines some graphical shape of an object.	
Stroke	This interface is used to define the outline of the shape object.	

Classes :

Review Question

1. *Explain package in Java. List out all packages with short description.* **GTU : May-12, Marks 7**

5.8 Static Import

Static import is a new feature added in Java. In order to access static members, it is necessary to **qualify references** with the class they came from. That means in order to access the static members of a class , it is a must to write the member along with its belonging class name. For example -

Java Program[StaticImport.java]

```
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}
class B extends A
{
    int b;
    void set_b(int i)
    {
        b=i;
    }
    void show_b()
    {
        System.out.println("The value of b= "+b);
    }
    void mul()
    {
        int c;
        c=a*b;
        System.out.println(" The value of c= "+c);
    }
}
class InheritDemo2
{
    public static void main(String args[])
    {
        A obj_A=new A();
        B obj_B=new B();
        obj_B.set_a(10);
        obj_B.set_b(20);
        obj_B.show_a();
        obj_B.show_b();
        obj_B.mul();
    }
}
```

a cannot be accessible

In the

Contents

6.1 Concept of Exception ..... **Summer-14,** ..... Marks

6.2 Exception and Error

6.3 Use of Try, Catch, Throw, Throws and Finally .. **Dec.-10, 11 June-11, May-12, Winter-12, 14** ..... Marks

..... Marks

6.4 Built in Exception ..... **Dec.-11,** ..... Marks

6.5 Custom Exception

6.6 Throwable Class

6.7 University Questions with Answers

```
ss ExceptionDemo

ublic static void main(String args[])

{
try
{
 int a,b;
 a=5;
 b=a/0;
 }
 catch(ArithmeticException e)
 {
  System.out.println("Divide by Zero\n");
 }
 System.out.println("...Executed catch statement");
 }
 }
```

A try block which includes the lines for which exception must be raised

Exception is caught by catch block.A try-catch pair must exist

Output

```
Divide by Zero

...Executed catch statement
```

5. Use of undeclared variables.
6. Use of = instead of ==
7. Incompatible types in assignment statement.
8. Illegal reference to the object.

For Example -

```
class CompileErrDemo
{
public static void main(String[] args)
{
System.out.println("Hello Java programmers);
}
}
```

Output

```
CompileErrDemo.java:5: unclosed string literal
System.out.println("Hello Java programmers),
^
CompileErrDemo.java:6: ')' expected
}
^
2 errors
```

6.3.2 Uncaught Exception

If there exists some code in the source program which may cause an exception and if the programmer does not handle this exception then java runtime system raises the exception.

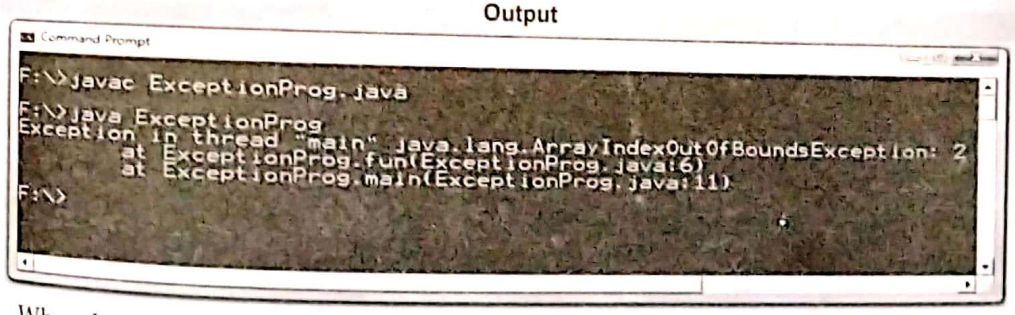
Following example illustrates how Java runtime system deals with an **uncaught exception.**

When we use an index which is beyond the range of index then **ArrayIndexOutOfBoundsException** occurs. Following Java program illustrates it

Java Program [ExceptionProg.java]

```
class ExceptionProg
{
 static void fun(int a[])
 {
  int c;
  c=a[0]/a[2];
}
public static void main(String args[])
{
 int a[]={10,5};
 fun(a);
}
}
```

Output



When Java runtime system detects an attempt to use an array index which is out of bound then it constructs the new exception object and throws the corresponding exception. The default handler displays the description of exception, prints the stack trace from the point at which the exception occurred and then terminates the program.

Note that in the above output, the name of the method as **ExceptionProg.fun** and the line number 6 is displayed to represent the position problematic statements. It also displays the name of the exception as **ArrayIndexOutOfBounds** to represent the type of exception that is caused by the above code.

Caught Exception

iii) Demonstrating NumberFormatException

When we try to convert invalid string to number then **NumberFormatException** occurs. Following is a simple Java program that shows illustration of this type of exception.

```
public class NumberFormatDemo
{
public static void main(String args[])
{
try
{
String str="Hello";
int num=Integer.parseInt(str);//Exception gets raised due to this line
System.out.println(num);
}
catch(NumberFormatException e)
{
System.out.println("Caught Exception "+e);
}
}
```

Output

```
D:\test>javac NumberFormatDemo.java
D:\test>java NumberFormatDemo
```

```
   try
{
    ...
    ...//exception occurs
}
catch(Exception_type e)
{
  ...//exception is handled here
}
   catch(Exception_type e)
{
  ...//exception is handled here
}
   catch(Exception_type e)
{
  ...//exception is handled here
}
```

```
class
{
public static void main(String args[])
{
int a=10,b=-1;
try
{
  b=a/0;
}
catch(ArithmeticException e)
{
System.out.println("In catch block: "+e);
}
finally
{
if(b!=-1)
  System.out.println("Finally block executes without occurrence of exception");
else
  System.out.println("Finally block executes on occurrence of exception");
}
}
}
```

Output

```
In catch block: java.lang.ArithmeticException: / by zero
Finally block executes on occurrence of exception
```

```
*/
class ExceptionThrows
{
static void fun(int a,int b) throws ArithmeticException
 {
 int c;
  try
 {
  c=a/b;
 }
 catch(ArithmeticException e)
 {
   System.out.println("Caught exception: "+e);
  }
 }

 public static void main(String args[])
 {
  int a=5;
  fun(a,0);
 }
}
```

Output

```
Caught exception: java.lang.ArithmeticException: / by zero
```

In above program the method *fun* is for handling the exception *divide by zero*. This is an arithmetic exception hence we write

```
int a[]={10,5};
fun(a);
}
}
```

Example 6.3.2 *Enlist and explain the difference between error and exception. Write a program to handle InterruptedException, IllegalArgumentException using try-cat-finally and throw.*

GTU : Dec.-10, Marks 7

Solution :

Exceptions : Exceptions are thrown if any kind of unusual condition occurs that can be caught. Sometimes it also happens that the exception could not be caught and the program may get terminated.

Errors : When any kind of serious problem occurs which could not be handled easily like OutOfMemoryError then an error is thrown.

i) Program for InterruptedException

```
class MyThread implements Runnable
{
 public void run()
 {
  System.out.println(Thread.currentThread().getName() + " started");
  try
  {
   Thread.sleep(10000);
   for (int i = 1; i < 10; i++)
   {
    if (Thread.interrupted())
    {
     System.out.println("interrupted");
     break;
    }
   }
  }
  catch (InterruptedException ex)
  {
   System.out.println(Thread.currentThread().getName() + " interrupted");
  }
  System.out.println(Thread.currentThread().getName() + " stopped");
 }

public class InterruptDemo
{
 public static void main(String args[]) throws InterruptedException
 {
  Thread t1 = new Thread(new MyThread(), "Thread #1");
  Thread t2 = new Thread(new MyThread(), "Thread #2");
  Thread t3 = new Thread(new MyThread(), "Thread #3");
  t1.start();
  Thread.sleep(1000);
  t1.interrupt();
  t2.start();
  Thread.sleep(3000);
  t2.interrupt();
  t3.start();
  Thread.sleep(2000);
  t3.interrupt();

 }
}
```

Output

```
Thread #1 started
Thread #1 interrupted
Thread #1 stopped
Thread #2 started
Thread #2 interrupted
Thread #3 started
Thread #2 stopped
Thread #3 interrupted
Thread #3 stopped
```

Program explanation

When a thread starts its execution, on interrupting that thread during the sleep, an exception **InterruptedException** gets invoked and the catch block gets executed.

ii) Program for IllegalArgumentException

```
class StringDemo
{
public static void main(String args[]) throws Exception
 {
  String T = "People love to program in Java";
  String P = "Java";
  String result = T.replaceAll(P, "$PHP");
  System.out.println(result);
 }
}
```

Output

```
Exception in thread "main" java.lang.IllegalArgumentException: Illegal group reference
    at java.util.regex.Matcher.appendReplacement(Unknown Source)
```

```
public class Test {
public static void main(String args[]) {
try {
System.out.println("calling method a");
a();
System.out.println("return from method a");
} catch(ArithmeticException e) {
System.out.println("main: catch");
} finally {
System.out.println("main: finally");
}
}
public static void a() {
try {
int x=8,y=0;
int z=x/y;
System.out.println("value of z="+z);
} catch(NumberFormatException e) {
System.out.println("method a:catch");
} finally {
System.out.println("method a:finally");
}}
```

GTU : Dec.-11, Marks 4

common exception types and causes are enlisted in the following t

Exception	Description
ArithmeticException	This is caused by error in Math operations. For e.g. Divide by zero.
NullPointerException	Caused when an attempt to access an object with a Null reference is made.
IOException	When an illegal input/output operation is performed then this exception is raised
IndexOutOfBoundsException	An index when gets out of bound, this exception will be caused
ArrayIndexOutOfBoundsException	Array index when gets out of bound, this exception will be caused.
ArrayStoreException	When a wrong object is stored in an array this exception must occur
EmptyStackException	An attempt to pop the element from empty stack is made then this exception occurs
NumberFormatException	When we try to convert an invalid string to number this exception gets caused

RuntimeException	To show general run time error this exception must be raised.
Exception	This is the most general type of exception
ClassCastException	This type of exception indicates that one tries to cast an object to a type to which the object can not be casted.
IllegalStateException	This exception shows that a method has been invoked at an illegal or inappropriate time. That means when Java environment or Java application is not in an appropriate state then the method is invoked.

6.4.1 Checked and Unchecked Exceptions

- There are two type of exceptions in Java
  - **Checked Exception :** These types of exceptions need to be handled explicitly by the code itself either by using the **try-catch** block or by using **throws**. These exceptions are extended from the **java.lang.Exception** class. **For example :** IOException which should be handled using the try-catch block.
  - **UnChecked Exception :** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions These exceptions are extended from **java.lang.RuntimeException** class **For example :** ArrayIndexOutOfBounds, NullPointerException, RunTimeException.

Review Question

1. *Differentiate between checked and unchecked exception* **GTU : Dec.-11, Marks 3**

6.5 Custom Exception

- We can throw our own exceptions using the keyword **throw.**
- The syntax for throwing out own exception is -

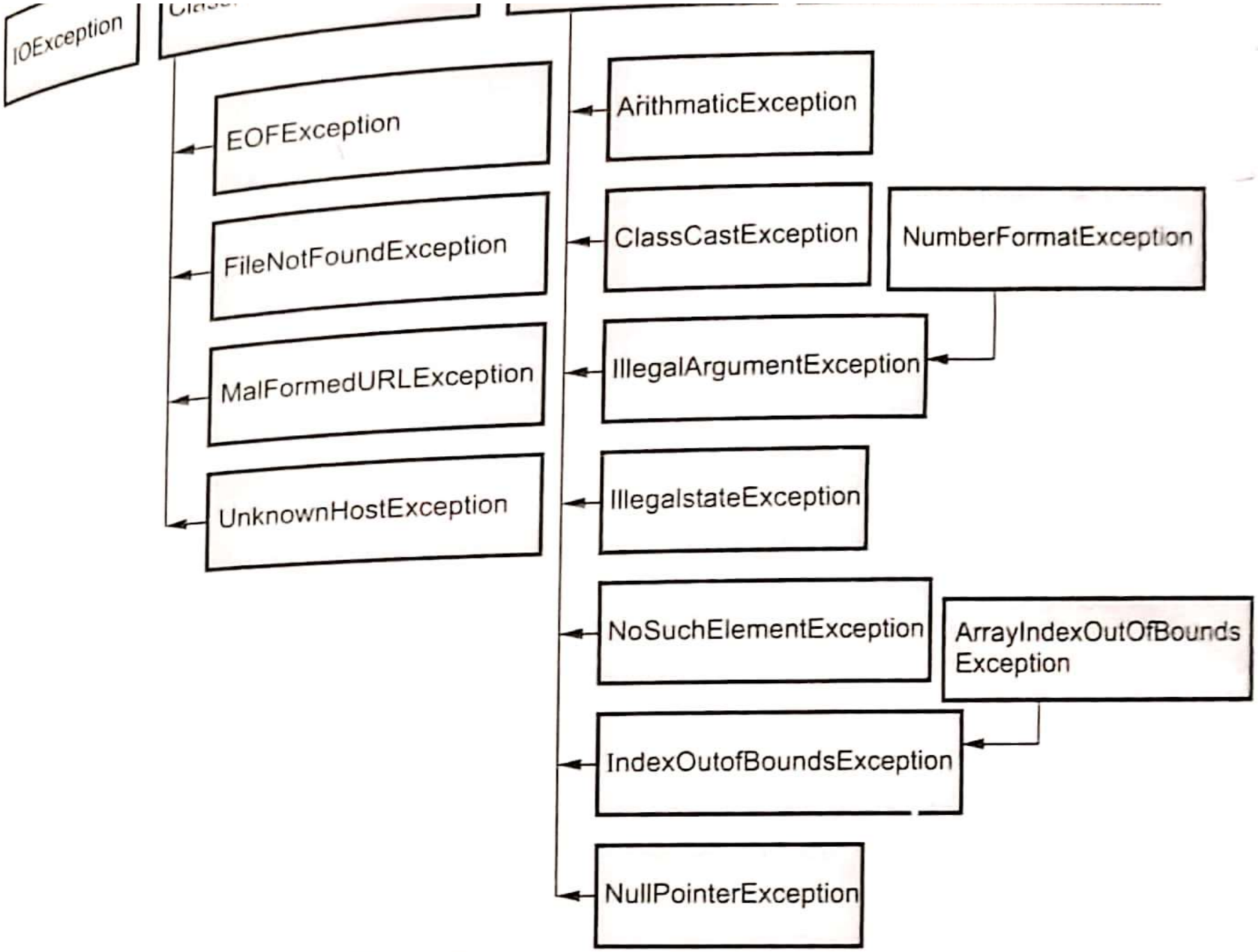


Fig. 6.5.1 Exception hierarchy

6.7 University Questions with Answers

(Regulation 2008)

7 Multithreaded Programming

05

Syllabus

Use of multithread programming, Thread class and runnable interface, Thread priority, Thread synchronization, Thread communication, Deadlock

Contents

7.1 Use of Multithread Programming

7.2 Thread States . . . . . **Dec.-11** . . . . . Marks 4

7.3 Thread Class and Runnable Interface . . . . . **May-12, Winter-14** . . . . . Marks 7

7.4 Extending Thread Class

7.5 Implementing Runnable Interface

7.6 Thread Priority

7.7 Multi-threading . . . . . **June -11, Summer-13**

. . . . . **Winter -12, 13** . . . . . Marks 7

7.8 Thread Synchronization **June-11,May-12**

. . . . . **Summer-14, Winter-12** . . . Marks 7

7.9 Thread Communication . . . . . **June -11,Summer-13**

. . . . . **Dec.-11** . . . . . Marks 7

7.10 Deadlock

7.11 University Questions with Answers

Multiprocessing
Program or process is a fundamental unit of multiprocessing environment.
Multiple programs get executed in multiprocessing environment.
During multiprocessing the processor switches between multiple programs or processes.
It is expensive because when a particular process uses CPU other processes has to wait.
It is less efficient in comparison with multithreading.
It helps in developing efficient operating system programs.

thread programs one is by extending **thread class** and the other is by implementing the **Runnable interface.**

- The **run()** method is the most important method in any threading program. By using this method the thread's behaviour can be implemented. The run method can be written as follows -

```
public void run()
  {
    //statements for implementing thread
  }
```

- For invoking the thread's run method the object of a thread is required. This object can be obtained by creating and initiating a thread using the start() method.

Review Question

1. *Explain thread life cycle in detail. Write a code to create thread in JAVA.*

GTU : May-12, Winter-14, Marks 7

7.4 Extending Thread Class

The procedure to extend a thread class is as given below -

Step 1 : The main class must extend the **Thread** class.

Step 2 : A thread is created in the constructor using the **start()** method. This method initiates the execution of thread.

Step 3 : Override the method **run()** by writing the code required to execute a thread.

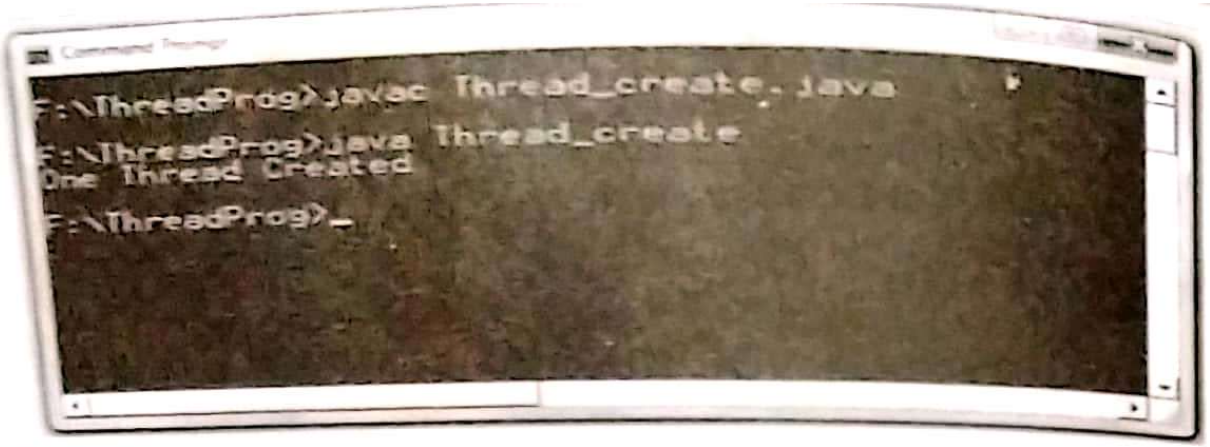
The illustrative Java program is as given below -

Java Program[Thread_create.java]

```
/**************************************************
This is a Java program which creates a single thread
by extending the Thread class
**************************************************/
class ThreadDemo extends Thread
{
  String str=" ";
  //in the constructor the thread is created
  ThreadDemo(String s)  //constructor
  {
    str=s;
    start();
  }
```

→ Extends the **thread** class

Start the thread executation



Review Questions

1. *List and explain the various methods defined by the thread class with examples of each.*

GTU : Dec.2010, Marks

2. *Define a thread. Explain the hierarchy of thread class. Discuss about the various methods in thread class with examples.*

7.5 Implementing Runnable Interface

Implementing thread program using **Runnable interface** is **preferable** to implementing it by extending the **thread class** because of the following two reasons:

1. If a class extends a **thread class** then it can not extends any other class which may be required to extend.
2. If a class **thread** is extended then all its functionalities get inherited. This is expensive operation.

Following Java program shows how to implement **Runnable interface** for creating single thread.

Program[RunnableTh.java]

```
{
  System.out.println("Thread #2");
  for(int k=1;k<=5;k++)
  {
    System.out.println("\tB: "+k);
  }
  System.out.println("\n-----End of Thread#2------");
 }
}
class Thread_PR_Prog
{
 public static void main(String[] args)
 {
  A obj1=new A();
  B obj2=new B();
  obj1.setPriority(1);
  obj2.setPriority(10);//highest priority
  System.out.println("Strating Thread#1");
  obj1.start();
  System.out.println("Strating Thread#2");
  obj2.start();
 }
}
```

Output

```
Strating Thread#1
Strating Thread#2
```

Thread 1 and Thread 2 starts

```
Thread #1
Thread #2
    B: 1
    B: 2
    B: 3
    B: 4
    B: 5
```

Thread 2 preempts the execution of thread 1

```
-----End of Thread#2-------
    A: 1
    A: 2
    A: 3
    A: 4
    A: 5
```

Then Thread 1 Continues its execution

```
-----End of Thread#1
```

[illegible]

```
{

Thread t;
MyClass()
{
  t = new Thread(this);
  t.setDaemon(true);
  t.start();

}
public void run()
{
  try
  {
    while(true)
    {
      System.out.print("a");
      Thread.sleep(10);
    }
  }
  catch(Exception e)
  {
```

```
    thread.setUncaughtExceptionHandler(
    new Thread.UncaughtExceptionHandler()
    {
      public void uncaughtException(Thread t, Throwable e) //handler for uncaught
exception
       {
        e.printStackTrace();
       }
     }
    );
    thread.start();//thread execution starts
   }

   public void run()
   {
    throw new NullPointerException();//throwing an unchecked exception
   }
}
```

Output

```
C >javac ExHandler.java
C  >java ExHandler

java.lang.NullPointerException
        at ExHandler run(ExHandler.java:21)
        at java.lang.Thread.run(Unknown Source)
```

```
Second Thread
Second Thread
Second Thread
Second Thread
Third Thread
Third Thread
Third Thread
Third Thread
Third Thread
```

Program explanation

In the **main** function we have created three threads namely t1,t2 and t3 having the strings "First Thread", "Second Thread" and "Third Thread" respectively. The execution of these threads will be started in **run()** method. We have invoked the **Thread.sleep** method in order to send a thread in **timed waiting state.** The value passed to the method as an argument denotes the time in milliseconds. This statement must be within **try-catch** block by throwing the exception **InterruptedException.**

If you execute the program repeatedly then you may notice that the threads t1,t2 and t3 can execute in any order.

7.7.2 Creation of Multiple Threads by Implementing Runnable Interface

In the following Java program, three threads are created and handled.

Java Program[Many_Thread_runnable.java]

```
class Thread_demo implements Runnable
{
 String str,msg;
 Thread t;
 Thread_demo(String str)
 {
 msg=str;
 t=new Thread(this,str);
 t.start();//directs to the run method
}
public void run()//thread execution starts
{
 for(int i=1;i<=5;i++)
 {
  System.out.println(msg);
 }
 try
 {
   Thread.sleep(1000);
 }
```



```
catch(InterruptedException e)
 {
 System.out.println("Exception in Thread handling");
 }
 }
}
public class Many_Thread_runnable
{
public static void main(String args[])
{
 Thread_demo t1=new Thread_demo("First Thread");
 Thread_demo t2=new Thread_demo("Second Thread");
 Thread_demo t3=new Thread_demo("Third Thread");
}
}
```

Output

```
First Thread
First Thread
First Thread
Second Thread
Second Thread
Second Thread
Second Thread
Second Thread
First Thread
First Thread
Third Thread
Third Thread
Third Thread
Third Thread
Third Thread
```

Example 7.7.1 *Write a Java application program for generating four threads to perform the following operations - i) Getting N numbers as input ii) Printing the numbers divisible by five iii) Printing prime numbers iv) Computing the average.*

Solution :

```
import java.io.*;
import java.util.*;
class FirstThread extends Thread
{
 public void run() //generating N numbers
 {

  int i;
```

```
for (i=1;i<=10;i++)
{
  System.out.println(i);
}
}

lass SecondThread extends Thread

public void run() //Displaying the numbers divisible by five
{
   int i;
   System.out.println("\nDivisible by Five: ");
   for (i=1;i<=10;i++) //10 can be replaced by any desired value
   {
      if (i%5==0)
         System.out.println(i);

   }

 }
}
class ThirdThread extends Thread
{

  public void run() //generating the prime numbers
  {
     int i;
     System.out.println("\nPrime Numbers: ");
     for (i=1;i<=10;i++) //10 can be replaced by any desired value
     {
       int j;
       for (j=2; j<i; j++)
       {
         int n = i%j;
         if (n==0)
             break;
       }
       if(i == j)
            System.out.println(i);

     }

  }
}
```

Example 7.7.2 *Write a multi-threaded program to meet following requirements :*

- *Read matrix[A] mXn*
- *Create m number of threads*
- *Each thread computes summation of elements of one row, i.e. i^{th} row of the matrix is processed by i^{th} thread, where 0<=i<m*
- *Print the results.*

GTU : June - 11, Summer-13, Marks 7

Solution :

```
import java.lang.*;
import java.io.*;
import java.util.*;
class testclass extends Thread
{
   static int A[][];
   static int sum[];
   static int m;
   static int n;
   int row;
   testclass(int i)
   {
      row=i;
      System.out.println("Thread# "+(i+1)+" is executing...");
      this.start();
   }
   public void run()
   {
      int j;
      sum[row]=0;
      for(j=0;j<n;j++)
      {
         sum[row]=sum[row]+A[row][j];
      }
   }
   public static void main(String args[])
   {
      int i,j;
```

```
Thread# 2 is executing...
Thread# 3 is executing...
-----------------
Sum of Row 1 is 10
Sum of Row 2 is 26
Sum of Row 3 is 20
```

Example 7.7.3 *What is multithreading? Why is it required ? Write a program that creates three threads make sure that the main thread executes last.* **GTU : Winter -12, Marks 7**

Solution : Refer section 7.7.

```
public class SynThDemo
{
   public static void main(String[] args)
   {
      B s=new B();
      A t1=new A(s," child 1");
      A t2=new A(s," Main thread");
      A t3=new A(s," child 2 ");

   }
class A extends Thread
{
   B s;
   A(B s,String str)
   {
      super(str);
      this.s=s;
      start();
   }
   public void run()
   {
      s.display(Thread.currentThread().getName());
   }
}
class B
{
   public synchronized void display(String str)
   {
      for(int i=0;i<3;i++)
      {
         System.out.println(str);  //displaying the message three times
         try
         {
            Thread.sleep(100);
         }catch(Exception e){ }
      }
```


Output

```
}
child 1
child 1
child 1
child 2
child 2
child 2
Main thread
Main thread
Main thread
```

Example 7.7.4 *Write an application that creates and starts three threads. Each thread is instantiated from the same class. It executes a loop with 10 iterations. Each iteration displays string "HELLO", sleeps for 300 milliseconds. The application waits for all the threads to complete and displays the message "Good Bye...".* **GTU : Winter-15, Marks 7**

Solution :

```
class MultiThread extends Thread
{
   public void run()
   {
      try
      {
         for(int i=0;i<10;i++)
         {
            System.out.println("HELLO");
            Thread.sleep(300);
         }
      }
      catch(Exception ex)
      {
         System.out.println(ex);
      }
   }
}
class MultiThreadProg
{
   public static void main(String[] args)
   {
      MultiThread t1=new MultiThread();
      MultiThread t2=new MultiThread();
      MultiThread t3=new MultiThread();
      t1.start();
      t2.start();
      t3.start();
```

```
        try
        {
            t1.join();
            t2.join();
            t3.join();

        }
        catch(InterruptedException ex)
        {
          System.out.println(ex);
        }
        System.out.println("Good Bye...");
    }
}
```

Output

```
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
ELLO
ELLO
ELLO
ELLO
ELLO
ELLO
ELLO
ELLO
ELLO
ELLO
ELLO
ELLO
od Bye
```

7.8 Thread Synchronization

GTU : June-11,May-12, Summer-14, Winter-12, Marks 7

When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization. The synchronization is the concept which is based on **monitor.** Monitor is used as mutually exclusive lock or **mutex.** When a thread owns this monitor at a time then the other threads can not access the resources. Other threads have to be there in **waiting** state.

In Java every object has implicit monitor associated with it. For entering in object's monitor, the method is associated with a keyword **synchronized.** When a particular method is in synchronized state then all other threads have to be there in waiting state.

There are two ways to achieve the synchronization -

1. Using Synchronized Methods
2. Using Synchronized Blocks (Statements).

Let us make the method synchronized to achieve the synchronization by using following Java program -

Use of Synchronization Method

ava Program[SynThProg1.java]

Points to remember about synchronization

1. **Only methods** can be synchronized but the variables and classes can not be synchronized.
2. Each **object** has **one lock.**
3. A class contains several methods and all methods need not be synchronized.
4. If two threads in a class wants to execute synchronized methods and both the methods are using the same instance of a class then only one thread can access the synchronized method at a time.
5. We **can not synchronize** the **constructors.**
6. A thread can acquire more than one lock.

2. Use of Synchronized Blocks

When we want to achieve synchronization using the synchronized block then create a block of code and mark it as synchronized. Synchronized statements must specify the object that provides the intrinsic lock. In the following program using this pointer the instance of an object is specified.

The syntax for using the synchronized block is -

```
synchronized(object reference )
{
  statement;
  statement;   //block of code to be synchronized


}
```

Java Program[SynThProg2.java]

```
public class SynThProg2
{
 public static void main(String[] args)
 {
```

Solution : **wait and sleep** - The wait and sleep methods are used in Java thread programming. They are basically used to make the thread to wait for some predefined time. Although both are meant for same thing there some differences between wait and sleep.

1. Wait is called from synchronized context only while sleep can be called without synchronized block.
2. Wait is called on Object while sleep is called on Thread.
3. Wait release lock on object while waiting while sleep doesn't release lock while waiting.
4. For wait, the thread waits until a condition is true while sleep is just to put your thread on sleep.

Synchronization - Refer section 7.8.

Example 7.12 *Explain thread synchronization with join() and is Alive() methods.*

GTU : Summer-14, Marks 7

Solution : The isAlive() method returns true if the invoking thread is alive or not. The join() method is used to wait for a thread to finish.

```
public class ThreadDemo1 extends Thread
{
 public void run()
 {
  System.out.println("Thread1");
  try
  {
   Thread.sleep(100);
  }catch(InterruptedException e){}
  System.out.println("Thread2");
 }
public static void main(String[] args)
{
  ThreadDemo1 t1 =new ThreadDemo1();
  ThreadDemo1 t2 = new ThreadDemo1();
  t1.start();
  t2.start();
  System.out.println("Thread One is alive: " + t1.isAlive());
  System.out.println("Thread Two is alive: " + t2.isAlive());
  try
  {
   System.out.println("Waiting for threads to finish.");
   t1.join();
   t2.join();
  }catch (InterruptedException e)
  { System.out.println("Main thread Interrupted"); }
  System.out.println("Thread One is alive: " + t1.isAlive());
  System.out.println("Thread Two is alive: " + t2.isAlive());
 }
}
```

Review Questions

1. *Why synchronization is required in multithreaded programming and how can implement it in program ?*

   GTU : May-12, Marks 7

2. *Why synchronization is required in multithreaded programming ? Write a program that uses thread synchronization to guarantee data integrity in a multithreaded application.*

   GTU : Winter-12, Marks 7

7.9 Thread Communication

GTU : June -11, Summer-13, Dec.-11, Marks 7

Two or more threads communicate with each other by exchanging the messages. This mechanism is called **inter-thread communication. Polling** is a mechanism generally implemented in a loop in which certain condition is repeatedly checked. To better understand the concept of polling, consider producer-consumer problem in which producer thread produces and the consumer thread consumes whatever is produced. Both must work in co-ordination to avoid wastage of CPU cycles. But there are situations in which the producer has to wait for the consumer to finish consuming of data. Similarly the consumer may need to wait for the producer to produce the data. In polling system either consumer will waste many CPU cycles when waiting for producer to produce or the producer will waste CPU cycles when waiting for the consumer to consume the data. In order to avoid polling there are three in-built methods that take part in inter-thread communication -

notify()	If a particular thread is in the sleep mode then that thread can be resumed using the notify call.
notifyall()	This method resumes all the threads that are in suspended state.
wait()	The calling thread can be send into a sleep mode.

Following is a simple Java program in which two threads are created one for producer and another is for consumer. The producer thread produces(writes) the numbers from 0 to 9 and the consumer thread consumes(reads) these numbers. The wait and notify methods are used to send particular thread to sleep or to resume the thread from sleep mode respectively.

```
class MyClass
{
  int n;
  boolean flag = false;
  synchronized int get() //by toggling the flag synchronised read and write is performed
  {
    if(!flag)
    try
    {
      wait();
    }
    catch(InterruptedException e)
    {
      System.out.println("InterruptedException!!!");
    }
    System.out.println("Consumer consuming: " + n);
    flag = false;
    notify();
    return n;
  }

  synchronized void put(int n)
  {
    if(flag)
    try
    {
      wait();
    }
    catch(InterruptedException e)
    {
      System.out.println("InterruptedException!!!");
    }
    this.n = n;
    flag = true;
    System.out.println("Producer producing " + n);
    notify();
  }
```

```
Producer producing 2
Consumer consuming: 2
Producer producing 3
Consumer consuming: 3
Producer producing 4
Consumer consuming: 4
Producer producing 5
Consumer consuming: 5
Producer producing 6
Consumer consuming: 6
Producer producing 7
Consumer consuming: 7
Producer producing 8
Consumer consuming: 8
Producer producing 9
Consumer consuming: 9
```

Program Explanation

Inside get( ) -

The **wait( )** is called in order to suspend the execution by that time the producer writes the value and when the data gets ready it notifies other thread that the data is now ready.

Similarly when the consumer reads the data execution inside get( ) is suspended. After the data has been obtained, get( ) calls notify( ). This tells producer that now the producer can write the next data in the queue.

Inside put( ) -

The wait( ) suspends execution by that time the consumer removes the item from the queue. When execution resumes, the next item of data is put in the queue and notify( ) is called. When the notify is issued the consumer can now remove the corresponding item for reading.

Example 7.9.1 *It is required to have total two threads, both capable of acting as a produce as well as a consumer. If first thread acts as a producer then, the second thread becomes the consumer and vice-versa. They communicate with each other through a buffer, storing one integer number. One of the threads initiates the communication by sending 1 to the other thread. The second thread, on receiving 1 sends 2 to the first thread. On receiving 2, the first thread sends three integer numbers, one by one to the second thread. The second thread consumes the numbers by displaying them. Both threads terminate after that. Note that both threads must be capable of initiating the communication. Write complete multi-threaded program to meet above requirements.*

GTU : June -11, Summer-13, Marks 7


```
class MyClass
{
 int n;
 boolean flag = false;
 synchronized int get()
 {
   while(!flag)
     try
     {
       wait();
     }
     catch(InterruptedException e)
     {
       System.out.println("InterruptedException caught");
     }
   System.out.println("Consumed :(Press Control-C to Stop.) " + n);
   flag = false;
   notify();
   return n;
 }
 synchronized void put(int n)
 {
   while(flag)
     try
     {
       wait();
     }
     catch(InterruptedException e)
     {
       System.out.println("InterruptedException caught");
     }
   this.n = n;
   flag = true;
   System.out.println("Produced : "+n);
   notify();
 }
}
class Producer implements Runnable
{
 MyClass obj;
 Producer(MyClass obj)
 {
   this.obj = obj;
   new Thread(this,"Producer").start();
 }
```

```
        {
            obj.put(i++);
        }
    }
}
class Consumer implements Runnable
{
    MyClass obj;
    Consumer(MyClass obj)
    {
        this.obj = obj;
        new Thread(this,"Consumer").start();
    }
    public void run()
    {
        while(true)
        {
            obj.get();
        }
    }
}
class testclass

    public static void main(String args[])
    {
        MyClass obj = new MyClass();
        new Producer(obj);
        new Consumer(obj);
    }
}
```

Review Question

1. *Explain interprocess communication mechanism wait(), notify() and notifyall() being used by Java to avoid polling.* **GTU : Dec.-11, Marks 7**

7.10 Deadlock

Deadlock is a situation in which when two or more threads are waiting for each other to release the objects they are holding. For example - Consider the scenario as if thread **A** is waiting for the **obj2** which is hold by thread **B** and if thread **B** is waiting for the **obj1** which is hold by thread **A**, this situation causes deadlock !!


```
import java.util.*;

public class DeadLockDemo extends Thread
{
    public static one obj1=new one();
    public static two obj2=new two();
```

Defining two objects

```
    public static class one
    {
        String a="One";
    }
    public static class two
    {
        String b="two";
    }
```

Classes that are used to define the obj1 and obj2

```
    public static void main(String[] a)
    {
        Thread t1 = new A();
        Thread t2 = new B();
        t1.start();
        t2.start();
    }
```

main method in which the two threads are started

```
    private static class A extends Thread
    {
        public void run()
        {
            synchronized (obj1)
            {
                System.out.println("Thread 1 Holds object obj1");
                try
                {
                    Thread.sleep(1000);
                }
                catch (InterruptedException e)
                {}
                System.out.println("Thread 1 is waiting for object obj2");
            }//synchronised block
        }//end of run
```

Class for Thread 1. This class shows that the thread 1 is holding obj1 and waiting for obj 2

8 Input Output Programming

10

Syllabus

Introduction to stream, Byte stream, Character stream, Readers and writers, File class, File InputStream, File Output Stream, InputStreamReader, OutputStreamWriter, FileReader, FileWriter, Buffered Reader.

Contents

8.1 Introduction to Stream

8.2 Types of Streams

8.3 Readers and Writers

8.4 File Class ..... **June -11,Dec.-10**

..... **Winter -12** ..... Marks 7

8.5 InputStream and OutputStream

8.6 File Input Stream and File Output Stream

8.7 Input Stream Reader and Output Stream Writer

8.8 FileReader and FileWriter

8.9 Buffered Reader

8.10 Programming Examples on I/O ..... **Dec.-10,11,14, Winter-12,13,14** Marks 7

Reader Classes

FileReader

PipeReader

FilterReader

BufferedReader

DataReader

LineNumberReader

PushbackReader

ByteArrayReader

SequenceReader

StringBufferReader

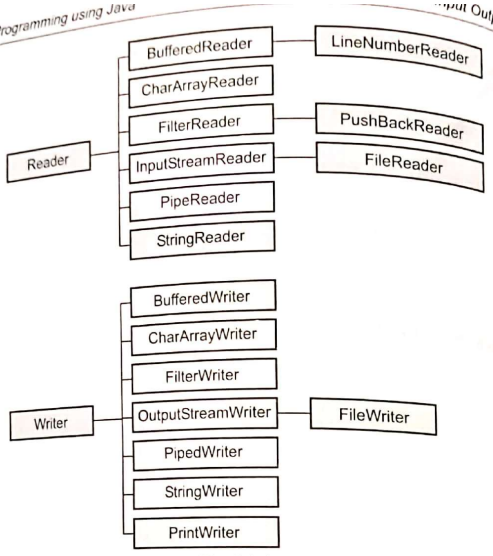


Fig. 8.3.1

Reader and **InputStream** define similar APIs but for different data types. example, Reader contains these methods for reading characters and arrays of characte

```
int read()
int read(char buf[])
int read(char buf[], int offset, int length)
```

InputStream defines the same methods but for reading bytes and arrays of bytes

```
int read()
int read(byte buf[])
int read(byte buf[], int offset, int length)
```

Writer and **OutputStream** are similarly parallel. **Writer** defines these method writing characters and arrays of characters :

```
int write(int c)
int write(char buf[])
int write(char buf[], int offset, int length)
```

And **OutputStream** defines the same methods but for bytes :

```
int write(int c)
int write(byte buf[])
int write(byte buf[], int offset, int length)
```


8.4 File Class

GTU : June -11,Dec.-10, Winter -12, Marks 7

Various classes that work on the Input/Output system makes use of the Java package java.io, however the File class works on file and file system. That means the File class **does not contain methods for reading and writing the file contents,** but it describes the properties of file itself. A **File object** is used to manipulate the information present in the disk file. Files are useful for storing the persistent and shared information.

What is absolute filename ?

The file is typically specified by the filename. The filename can be specified with its complete path and drive letter. Such a specification of filename is called absolute filename. For example - **c:\test\myprogram.html** where **test** is a current directory in which **myprogram.html** lies.

What is relative filename ?

The file name is a filename relative to the current directory. That means while specifying the relative file name we should not mention the complete path of the corresponding file.

For example - **new File("myprogram.html")**

Example 8.4.1 *Explain file constructors, any two methods of class file and method seek ?*

GTU : June -11, Marks 7

Solution : The file constructor is used in Java while handling the file I/O. These constructors are there in java.io.File. For example -

```
File f1=new File("input.dat");
```

Methods in class File :

1. **Public String getPath()** - This method returns a string that contains the path being used for the file.
2. **Public String getParent()** - This method returns a string that contains the name of the single directory which contains this file in the hierarchy.
3. **Public boolean exists()** - This method indicates whether or not a particular file exists where you expect it to be.
4. **Public boolean canWrite()** - This method indicates whether you have write access to this file.

Seek Method

This method belongs to the stream **RandomAccessFile.** The seek() that allows you to specify the index from where the read and write operations will start. The syntax of seek method is

```
import java.io.File;
class FileProperties {
public static void main(String args[])
{
    File FileObj=new File("test1/fun.txt");
    System.out.println("-----------------------------------------");
    System.out.println("File Properties are as follows...");
    System.out.println("-----------------------------------------");
    System.out.println("File Name: " + FileObj.getName());
    System.out.println("Path: " + FileObj.getPath());
    System.out.println("Abs Path: " + FileObj.getAbsolutePath());
    System.out.println("Parent: " + FileObj.getParent());
    System.out.println("This file exists: "+FileObj.exists());
    System.out.println(FileObj.canWrite() ? "is writeable" : "is not writeable");
    System.out.println(FileObj.canRead() ? "is readable" : "is not readable");
```

```
{
 System.out.println("...
 }
catch (Exception e)
{
 System.err.println("Error: " + e.getMessage());
}
File dir = new File("home");
String[] children = dir.list();
if (children == null)
{
 // Directory does not exist
}
//now inside the subdirectory home/abc create some file with some contents in it
//so that the file name along with its size and file permissions can be displayed
else
{
System.out.println("The contents under the directory "+dir+" are...");
for (int i=0; i<children.length; i++)
{
  // Get filename of file or directory
  String filename = children[i];
  File file = new File(filename);
if(!file.isFile())
  System.out.println(filename);
  else
  {  //displaying the file name, file size and permissions
  System.out.print(filename+","+file.length()+" bytes"+",");
  System.out.print(file.canRead()?"R":"-");
  System.out.print(file.canWrite()?"W":"-");
  System.out.print(file.canExecute()?"X":"-");
  }
}

System.out.println("\n...Destructing the class data");
r.delete();
```

Solution :

```
import java.io.*;
class FileORDirDemo
{
 public static void main(String args[]) throws IOException
 {
   File fobj = new File(args[0]);
   boolean isFile = fobj.isFile();
   boolean isDirectory = fobj.isDirectory();
   if(isFile)
   {
     System.out.println(fobj.getPath() + " is a file.");
     long length=fobj.length();
     System.out.println("\n The size of the file is: "+length+" Bytes");
   }
   else if(isDirectory)
   {
     System.out.println(fobj.getPath() + " is a directory.");
     String[] s = fobj.list();
     System.out.println("\n It Contains...");
     for (int i=0; i < s.length; i++)
     {
       File f = new File(fobj.getAbsolutePath() + "/" + s[i]);
       if (f.isDirectory())
       {
         System.out.println(s[i] + " as a directory");
       } else
       {
         System.out.println(s[i] + " as a file");
       }
     }
   }
   else
     System.out.println("It is not present as File or Directory.");


   }
}
```

Method	Purpose
void close()	This method closes the output stream and if we try to write further after using this method then it will generate **IOException**
void flush()	This method clears output buffers.
void write(int val)	This method allows to write a single byte to an output stream.
void write(byte buffer[ ])	This method allows to write complete array of bytes to an output stream.
void write(byte buffer[ ],int offset, int n)	This method writes n bytes to the output stream from an array *buffer* which is beginning at *buffer[offset]*

8.6 File Input Stream and File Output Stream

The **FileInputStream** class creates an **InputStream** using which we can read bytes from a file. The two common constructors of FileInputStream are -

```
FileInputStream(String filename);
FileInputStream(File fileobject);
```

In the following Java program, various methods of **FileInputStream** are illustrated -

Java Program[FileStreamProg.java]

The **FileOutputStream** can be used to write the data to the file using the **OutputStream.** The constructors are -

```
FileOutputStream(String filename)
FileOutputStream(Object fileobject)
FileInputStream(String filename,boolean app);
FileInputStream(String fileobject,boolean app);
```

The *app* denotes that the append mode can be true or false. If the append mode is true then you can append the data in the file otherwise the data can not be appended in the file.

In the following Java program, we are writing some data to the file.

Java Program[FileOutStreamProg.java]

```
{
    public static void main(String[] args)
    {
        InputStreamReader in = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(in);
        int num=0;
        try
        {
            System.out.println("Enter some number..");
            num = Integer.parseInt(br.readLine());

        }
        catch (NumberFormatException e)
        {
                e.printStackTrace();
        }
        catch (IOException e)
        {
                e.printStackTrace();
        }
        System.out.println("you entered "+num);
    }
}
```

Output

```
Enter some number..
100
you entered 100
```

Program Explanation

When we want to read some character from the console we should make use of **system.in**. The character stream class **BufferedReader** is used for this purpose. Here we should pass the variable **System.in** to **BufferedReader** object. Along with this we should also mention the abstract class of BufferedReader class which is **InputStreamReader**.

Example Program for OutputStreamWriter

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;
public class OutputStreamWriterDemo
{
    public static void main(String[] args)
    {
        String str = "Hello Friends!!!";
```



```
        BufferedWriter bw = null;
        try
        {
            Writer w = new OutputStreamWriter(System.out);
            bw = new BufferedWriter(w);
            System.out.println("\n The message written as output...");
            bw.write(str);
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                bw.close();
            }
            catch (IOException ex) {ex.printStackTrace();}
        }
    }
}
```

Output

```
The message written as output...
Hello Friends!!!
```

Program explanation

For writing the data to the console we need to make use of **BufferedWriter** class. The System.out is a stream created for writing the contents to the console. And for writing the contents to the console we pass the instance of OutputStreamWriter class as an input to **BufferedWriter**. Using the object of **BufferedWriter** and **Write** method the contents can be written to the console.

8.8 FileReader and FileWriter

For reading the simple text files the FileReader is used. The instance is then wrapped in **BufferedReader** class.

Constructors for FileReader

```
FileReader(File file)
FileReader(FileDescriptor fd)
FileReader(String Filename);
```

The methods used by this class are

Method	Description
public int read()	Reads a single character. Returns an int, which represents the character read.
public int read(char [] c, int offset, int length)	Reads characters into an array. Returns the number of characters read
public void close()	closes FileReader

Example for FileReader

Step 1: Create a file named **temp.txt** using notepad.And write something to it I
Written following contents to this file -

```
Hello everybody
Enjoy Java programming
Bye!!See You.
```

Step 2 : Now create a Java program that uses **FileReader** class for reading the conte
of the file created in step 1. The code is as follows -

```
import java.io.*;
public class FileReaderDemo

  public static void main(String [] args)
  {
    String fileName = "d:\\temp.txt";
    String line = null;
    try
    {
      // FileReader reads text files in the default encoding.
      FileReader fileReader = new FileReader(fileName);
      BufferedReader bufferedReader = new BufferedReader(fileReader);
      while((line = bufferedReader.readLine()) != null)
      {
        System.out.println(line);
      }
      bufferedReader.close();
    }
    catch(FileNotFoundException ex)
    {
      System.out.println("Unable to open file");
    }
    catch(IOException ex)
    {
      System.out.println("Error reading file");
```



```
    }
  }
}
```

Output

```
Hello everybody
Enjoy Java programming
Bye!!See You.
```

For writing the character oriented data to the file the FileWriter class is used.

Constructors for FileWriter

FileWriter(File file)
FileWriter(String File)

Methods for FileWriter

Method	Description
public void write(String text)	writes the string into FileWriter
public void write(char c)	writes the char into FileWriter
public void write(char[] c)	writes char array into FileWriter
public void flush()	flushes the data of FileWriter
public void close()	closes FileWriter

Example for FileWriter

Following is a Java program that makes use of FileWriter for writing the contents to the text file.

The instance of FileWriter class is wrapped in bufferedWriter and then using the write method of bufferedWriter class the contents can be written. Following program illustrates it.

```
import java.io.*;
public class FileWriterDemo
{
    public static void main(String [] args)
    {
        String fileName = "d:\\temp.txt";
        try
        {
            FileWriter fileWriter = new FileWriter(fileName);
            BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
                bufferedWriter.write("Hello everybody");
            bufferedWriter.write("Writing Programs in Java is a fun!!");
            bufferedWriter.newLine();
```

Note : After executing the above program, you just open the temp.txt file and read that the above specified contents are written in this file.

8.9 Buffered Reader

In this section, we will understand how to read the input from console? In Java System is a class defined in java.lang and in, out and err are the variables of the class System. Hence System.out is an object used for standard output stream and System.err are the objects for standard input stream and error.

When we want to read some character from the console we should make use of **system.in.** The character stream class **BufferedReader** is used for this purpose. In fact we should pass the variable **System.in** to **BufferedReader** object. Along with it we should also mention the abstract class of BufferedReader class which is **InputStreamReader**. Hence the syntax is,

```
import java.io.*;
import java.util.*;
class FileDemo
{
   public static void main(String[] args)throws Exception
   {
      int count=0;
      File fin = new File("C:/lab/word.txt");
      BufferedReader br = new BufferedReader(new FileReader(fin));

      String FullStr="";
        String TempStr;
      while((TempStr=br.readLine())!=null)//if multi-line input is present in the file
      {                    //then it is collected in one string separated by spaces
        FullStr+=TempStr+" ";
      }
      Scanner input=new Scanner(System.in);
      System.out.print("Enter the word to be searched from the file: ");
      String word1=input.next();

      String Word_List[]=FullStr.split(" ");//Extract words from string for counting no of
occurences
```

Java is fun. Programming in Java is interesting. People love Java

Example 8.10.2 *Write a program to replace all "word1" by "word2" to a file without using temporary file and display the number of replacement.* **GTU : Dec.-10, Marks 7**

Solution :

```
import java.io.*;
import java.util.*;
class FileDemo
{
  public static void main(String[] args)throws Exception
  {
    int count=0;
    File fname=new File("C:/lab/word.txt");
    BufferedReader br=new BufferedReader(new FileReader(fname));

    String FullStr="";
      String TempStr;
    while((TempStr=br.readLine())!=null)//if multi-line input is present in the file
    {               //then it is collected in one string separated by spaces
      FullStr+=TempStr+" ";
    }
    Scanner input=new Scanner(System.in);
    System.out.print("Enter the word to be searched from the file: ");
    String word1=input.next();

    String Word_List[]=FullStr.split(" ");//Extract words from string for counting
//no.of occurrences

    //Searching for the word1
    for(int i=0;i<Word_List.length;i++)
    {
      if(Word_List[i].equals(word1))
      {
        count++; //counting no. of occurrences
      }
    }
    System.out.print("\nEnter new Word for replacement: ");
    String word2=input.next();

    String New_Str=FullStr.replace(word1,word2);
    //File is opened for writing purpose
    BufferedWriter bw=new BufferedWriter(new FileWriter(fname));
    //Modified contents are written to the file
    bw.write(New_Str);
    System.out.println("\n The replacement is done and file is rewritten");
    System.out.println("\nThe number of replacements are "+count);
    bw.close();

    br=new BufferedReader(new FileReader(fname));
    System.out.println("\n Now,the Contents of the file are...\n ");
    while((TempStr=br.readLine())!=null)
      System.out.println(TempStr);
    br.close();
  }
}
```


Example 8.10.3 *Write a program that takes input for filename and search word command-line arguments and checks whether that file exists or not. If exists, the pr will display those lines from a file that contains given search word.*

GTU : Dec.-11, Ma

Solution :

```
import java.io.*;
class WordSearchDemo
{
  public static void main(String args[]) throws IOException
  {
    boolean status;
    boolean WordOccur=false;
    File fobj = new File(args[0]);
    if(fobj.exists())
      status = true;
    else
    {
      status=false;
      System.out.println("\n This file does not exist!!!");
      return;
    }
    if(status)
    {
      BufferedReader in = new BufferedReader(new FileReader(fobj));
      String Line;
      String key = args[1];
      while ((Line = in.readLine()) != null)
      {
        String[] Data = Line.split(" ");
        for(String Element:Data)
        {
          if(Element.equalsIgnoreCase(key))
          {
            System.out.println("" + Line+"")
```

```
import java.io.*;
import java.util.*;

public class WordCountDemo
{
   public static void main(String[] args) throws NullPointerException,IOException
   {


      try
      {
         BufferedReader in = new BufferedReader(new FileReader(args[0]));
         String New_Line = "", Word = "";
         int Word_Count = 0;
         while ((New_Line = in.readLine()) != null)
            Word += New_Line + " ";
         StringTokenizer Token = new StringTokenizer(Word);
         while (Token.hasMoreTokens())
         {
            String s = Token.nextToken();
            Word_Count++;
         }
         System.out.println("Text file contains " + Word_Count + " words.");
      }
      catch (NullPointerException e)
```

```
        for(int i = 0; i < temp_str.length(); i++)
        {
          if(Character.isWhitespace(temp_str.charAt(i)))
            wspace_count++;
          if(Character.isDigit(temp_str.charAt(i)))
            digit_count++;
          if(Character.isLetter(temp_str.charAt(i)))
            alphabet_count++;
        }

        token=new StringTokenizer(temp_str," ");
        while(token.hasMoreTokens())
        {
          word_count++;
          String s=token.nextToken();
          char_count+=s.length();
        }

      }
      System.out.println("Character Count : "+char_count);
      System.out.println("Word Count : "+word_count);
      System.out.println("Line Count : "+line_count);
      System.out.println("Aplphabet Count : "+alphabet_count);
      System.out.println("Digit Count : "+digit_count);
      System.out.println("White Space Count : "+wspace_count);

      br.close();
    }
}
```

Sample input file: **inputfile.txt**

```
Hello123
How are u?
I am fine
Enjoying my job
```

Output

```
Enter filename : inputfile.txt
Character Count : 36
Word Count : 10
Line Count : 4
Aplphabet Count : 32
Digit Count : 3
White Space Count : 6
```


9 Collection Classes

02

Syllabus

List, AbstractList, ArrayList, LinkedList, Enumeration, Vector, Properties, Introduction to Java.util package.

Contents

9.1 *Introduction* ..... **Summer-14** ..... Marks 3
9.2 *Commonly used Collection Classes*
9.3 *List*
9.4 *Abstract List*
9.5 *Array List*
9.6 *Linked List*
9.7 *Enumeration*
9.8 *Vector Properties* ..... **Winter-14** ..... Marks 7
9.9 *Introduction to Java Util Package*
9.10 *University Questions with Answers*

Method	Description
boolean add(Object *obj*)	Objects are added using this method. This method takes arguments of type object.
boolean addAll(Collection *collection*)	Entire content of one collection can be added to another using this method.
void clear()	In order to clear the collection, this method is used.
boolean contains(Object *obj*)	For checking whether the collection contains specific object or not this method is used.
boolean containsAll(Collection *collection*)	If all the elements of the collection are present in the collection then this method returns true.
boolean isEmpty()	It's a Boolean method which determines whether collection is empty or not. If a collection is empty then it return true otherwise false.
Iterator iterator()	Returns iterator to the collection.
boolean remove(Object *obj*)	An object can be removed by this method.
boolean removeAll(Collection *collection*)	It helps in removing a group of objects.

Description

This method inserts the object obj at specified location in the list. This location is denoted by index i.

This method inserts elements of collection collection at specified location in the list. This location is denoted by index i.

To get the object stored at specified location in the list, this method is used. This location can be obtained with the help of index i.

The value of the element can be set using this method

If object is an element of the list then its index will be returned by this method

This method returns the index of the last occurrence present in the list

```
 System.out.println(list.size());
 return 0;
}

public boolean add(Object e)
{

 list.add(e);
 System.out.println(list);
 return false;
}

public static void main(String[] args)
{
 AbstractListDemo obj= new AbstractListDemo();
 System.out.println("Adding the elements in the List");
 for(int i=0; i<=3; i++)
 {
    obj.add(i);
 }
 System.out.print("Size of list = ");
 obj.size();
 System.out.println("Element at 3rd position");
 obj.get(3);
 System.out.println("Adding few more elements");
 obj.add(4);
 obj.add(5);
 obj.add(6);
 obj.add(7);
 System.out.print("Size of list =");
 obj.size();

 }
}
```

Output

```
Adding the elements in the List
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
Size of list = 4
Element at 3rd position
3
Adding few more elements
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5, 6]
[0, 1, 2, 3, 4, 5, 6, 7]
Size of list =8
```

Review Question

1. *Explain AbstractList with supporting operations on it.*

9.5 ArrayList

The **ArrayList** class implements the List interface. It is used to implement the dynamic array. The dynamic array means the size of array can be created as per requirement. Hence **ArrayList** is a variable length array of object references. Initially ArrayList is created with some initial size and then as per requirement we can extend the size of array. When the objects are removed then the size of array can be reduced. The syntax of using ArrayList is as given below -

ArrayList() ← Creates an empty list

ArrayList(Collection *collection*) ← Creates a list in which the collection elements are added

ArrayList(int *c*) ← Creates a list with specified capacity *c*, the capacityrepresents the size of the underlying array

Let us see simple Java program which demonstrates the ArrayList

Java Program [ArrayListProg.Java]

```
/*********************************************************************
Program uses the ArrayList collection class to
 implement ArrayList data structure
*********************************************************************/
import java.util.*;
class ArrayListProg
{
 public static void main(String[] args)
 {
  System.out.println("\n\t\t Program for Implementing Array List");
  ArrayList obj=new ArrayList();              //creation of ArrayList
  System.out.println("\n Inserting some elements in the array");
  obj.add(10);
  obj.add(20);
  obj.add(30);
  obj.add(40);
```

```
bj.add(50);
ystem.out.println("The array elements are... "+obj);
ystem.out.println("\n Inserting some elements in the array in between");
bj.add(4,45);                                    //inserting after element 40
System.out.println("The array elements are... "+obj);
System.out.println("\n Removing some elements from the array");
bj.remove(1);                                    //array index is passed to remove method
System.out.println("The array elements are... "+obj);
System.out.println("The array size is... "+obj.size());
```

Output

Program for Implementing Array List

```
nserting some elements in the array
he array elements are... [10, 20, 30, 40, 50]

nserting some elements in the array in between
he array elements are... [10, 20, 30, 40, 45, 50]

Removing some elements from the array
The array elements are... [10, 30, 40, 45, 50]
```

```
case '3': System.out.println("\n Enter the element to be inserted in the list");
          val=getInt();
           obj.addFirst(val);
           System.out.println("\n The element inserted!!!");
          break;
case '4': System.out.println("\n Enter the element to be inserted in the list");
          val=getInt();
           obj.addLast(val);
           System.out.println("\n The element inserted!!!");
          break;
case '5': obj.removeFirst();
           System.out.println("\n The element deleted!!!");
          break;
case '6': obj.removeLast();
           System.out.println("\n The element deleted!!!");
          break;
case '7':System.out.println("\n Enter the element to be inserted in the list");
          val=getInt();
           System.out.println("\n Enter the position at which the element is to be
nserted");
          position=getInt();
           obj.add(position,val);
           System.out.println("\n The element inserted!!!");
          break;
case '8': System.out.println("\n Enter the position of element to be deleted");
           position=getInt();
           obj.remove(position);
           System.out.println("\n The element deleted!!!");
          break;

System.out.println("\n Do u want to go to main menu?");
ch=getChar();
}while(ch=='y');
 }
////////////////////////////////////////////////////////////////////////////////////////////////
/Following functions are used to handle the inputs entered
/by the user using keyboard
////////////////////////////////////////////////////////////////////////////////////////////////
ublic static String getString() throws IOException
{
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader b = new BufferedReader(input);
    String str = b.readLine();//reading the string from console
    return str;
}

ublic static char getChar() throws IOE
{
  String str = getString();
  return str.charAt(0);//reading first char of console string
}
public static int getInt() throws IOException
{
  String str = getString();
  return Integer.parseInt(str);//converting console string to numeric value
}
}//end of class
```

Output

Program for Implementing Linked List

```
1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
1

 Enter the element to be inserted in the list
10
 Do u want to insert more elements?
y
 Enter the element to be inserted in the list
20
 Do u want to insert more elements?
y

 Enter the element to be inserted in the list
30
 Do u want to insert more elements?
y

 Enter the element to be inserted in the list
40
 Do u want to insert more elements?
n

 Do u want to go to main menu?
y
```

Program f...

1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
2

The List elements are... [10, 20, 30, 40]
The size of linked list is... 4

Do u want to go to main menu?
y

Program for Implementing Linked List

1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
3

Enter the element to be inserted in the list
9

The element inserted!!!

Do u want to go to main menu?
y

Program for Implementing Linked List

1.Create
2.Display
3.Insert First

```
Do u want to go to main menu?
y
                Program for Implementing Linked List


1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
5

 The element deleted!!!

 Do u want to go to main menu?
y
                Program for Implementing Linked List

1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
2
        The List elements are... [10, 20, 30, 40, 45]
        The size of linked list is... 5

 Do u want to go to main menu?
y
                Program for Implementing Linked List

1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
```



```
7.Insert At any Position
8.Delete From any Position
Enter Your choice
6

The element deleted!!!

Do u want to go to main menu?
y
                Program for Implementing Linked List


1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
2
        The List elements are... [10, 20, 30, 40]
        The size of linked list is... 4

Do u want to go to main menu?
y
                Program for Implementing Linked List

1.Create
2.Display
3.Insert First
4. Insert Last
5.Delete First
6.Delete Last
7.Insert At any Position
8.Delete From any Position
Enter Your choice
7

 Enter the element to be inserted in the list
15

 Enter the position at which the element is to be inserted
1
```

9.7 Enumeration

Enumeration generates series of elements one at a time. This interface is used to access the elements defined in some collection like **Vector** or **ArrayList.** Generally such collection is of unknown size.

The enumeration makes use of two methods -

1. **nextElement()** : It returns the next object in the list.
2. **hasMoreElements()** : If the next element is present in the collection then it returns true otherwise returns false.

The enumeration does not allow modification of the collection being traversed.

Following is a simple Java Program which makes use of enumeration -

Java Program

```
}
}
```

Output

```
One
Two
Three
Four
Five
```

Review Question

1. *Explain the concept of Enumeration*

9.8 Vector Properties

GTU : Winter-14, Marks 7

Vector is similar to **Array List** class which implements the **dynamic array**. It implements the List interface. This class contains various additional methods which are enlisted below -

Method	Description
void addElement(object)	For adding some element in the vector this method is used.
void insertElementAt(object obj,int pos)	For inserting the element in the vector specified by its position.
boolean removeElement(object ele)	This method removes the specified element.
void removeAllElements()	This method is for removing all the elements from the vector.
void removeElementAt(int pos)	The element specified by its position gets deleted from the vector.
int capacity()	It returns the capacity of the vector.
int size()	It returns the total number of elements present in the vector.
boolean isEmpty()	It returns true if the vector is empty.
object firstElement()	Returns the first element of the vector.
int indexOf(object ele)	It returns the index of corresponding element in the vector.
void setSize(int size)	This method is for setting the size of the vector.

Following program illustrates the implementation of vector class.

Java Program [VectorProg.Java]

```
/**********************************************
Program for implementing vector class
**********************************************/
import java.io.*;
import java.util.*;
class VectorProg
{
public static void main(String[] args)throws IOException
{
Vector obj=new Vector(3);
int ival;
double dval;
char ans='y';
System.out.println("The capacity of vector is  "+obj.capacity());
System.out.println("The total number of elements are  "+obj.size());
System.out.println("\t Inserting the integers");
do
{
System.out.println("\n Enter some integer value");
ival=getInt();
obj.addElement(ival);
System.out.println("\n Do u want to enter more?");
ans=getChar();
}while(ans=='y');
System.out.println("The elements in the vector are "+obj);
System.out.println("\t Inserting the double values");
do
{
System.out.println("\n Enter some double value");
dval=getDouble();
obj.addElement(dval);
System.out.println("\n Do u want to enter more?");
ans=getChar();
}while(ans=='y');
System.out.println("The elements in the vector are "+obj);
System.out.println("\t The size of vector is  "+obj.size());
System.out.println("\t The first element of the vector is "+(Integer)obj.firstElement());
System.out.println("\t The last element of the vector is "+(Double)obj.lastElement());
System.out.println("Removing 2 the elements");
obj.remove(0);
obj.remove(2);
System.out.println("Now The elements in the vector are "+obj);
```

```
}
//////////////////////////////////////////////////////////////////////////
Following functions are used to handle the inputs entered
//by the user using keyboard
//////////////////////////////////////////////////////////////////////////
public static String getString() throws IOException
 {
    InputStreamReader input = new InputStreamReader(System.in);
    BufferedReader b = new BufferedReader(input);
    String str = b.readLine();//reading the string from console
    return str;
 }

public static char getChar() throws IOException
 {
   String str = getString();
   return str.charAt(0);//reading first char of console string
 }
public static int getInt() throws IOException
 {
    String str = getString();
    return Integer.parseInt(str);//converting console string to numeric value
  }
public static double getDouble() throws IOException
 {
    String str = getString();
    return Double.parseDouble(str);//converting console string to numeric value
  }
}
```

Output

```
The capacity of vector is  3
The total number of elements are  0
       Inserting the integers

 Enter some integer value
11

 Do u want to enter more?
y

 Enter some integer value
22

 Do u want to enter more?
y
```

Interface/Classes	Description
Collection	This is the root interface used in collection hierarchy. It provides the implementation for more specific sub interfaces such as Set and List
Comparator	The comparison function which provides the total ordering on collect objects.
Iterator	It is an interface over the Collection interface
Map	It is an object that maps the keys to values
Set	It is a collection that contains no duplicate elements
List	It is an interface for ordered collection of elements.
AbstractCollection	This class provides skeletal implementation of the Collection interface
AbstractList	This class provides skeletal implementation of the List interface
AbstractSet	This class provides skeletal implementation of the Set interface
Date	This class interface represents specific instance in time
Dictionary	This class allows to map keys to values.

Syllabus

InetAddress class,Socket class, DatagramSocket class, DatagramPacket class.

Contents

10.1 Introduction

10.2 InetAddress Class

10.3 Inet4Address and Inet6Address

10.4 Socket Class

10.5 Datagram Socket Class and Datagram Packet Class

10.1 Introduction

The **java.net** package is for providing the useful classes and interfaces for networking applications which are used in sockets and URL. These are as given below -

Classes

Name	Description
ContentHandler	This class is a superclass of all the classes that read the data from a class URLConnection. It also builds the appropriate local object based on MIME types.
DatagramSocket	This class represents the Datagram Socket (UDP socket).
DatagramPacket	This class represents the Datagram Packet (UDP packets for containing data).
InetAddress	This class represents the IP address.
InetSocketAddress	The IP socket address is a combination of IP address and port number. To implement such IP socket address this class is used.
ServerSocket	For implementing serverside sockets this class is useful.
Socket	For implementing client side sockets this class is useful.
SocketAddress	This class helps to represent the socket address without specification of protocol.
URL	This class is for creating a reference of Uniform Resource Locator which points to WWW.
URI	This class provides the object of Uniform Resource Identifier.
URLConnection	For establishing a communication between application program and URL this class is used.

Interfaces

Name	Description	Some methods
ContentHandlerFactory	This interface is for defining the factory for content handlers.	*createContentHandler(string MIME_type)*-Creates ContentHandler.
SocketImplFactory	This interface is for defining the factory for implementing the sockets.	*createSocketImpl()*- Creates a new instance for implementing socket.
URLStreamHandlerFactory	For implementing URL stream protocol handlers the factory can be defined by this interface.	*createURLStreamHandler (String p)*- For specific protocol 'p' a new URLStreamHandler instance can be created by this

10.2 InetAddress Class

- The InetAddress class from **java.net** package represents the IP addresses. It work with either **host name** or **numerical IP address** of corresponding host.
- **InetAddress** class offers many useful methods for handling IP addresses and ho names.
- We will now discuss the three important **methods** from InetAddress class a those are
  - getLocalHost
  - getByName
  - getAllByName

Following program illustrates the use of **getLocalHost()** method -

Java Program

```
/*********************************************************************
This program make use of the method getLocalHost()
method for finding the IP address of local host machine.
*********************************************************************/
import java.net.*;
class InetAdd1
{
    public static void main(String args[]) throws UnknownHostException
    {
        InetAddress local_add=InetAddress.getLocalHost();
        System.out.println("Local Host is: "+local_add);
    }
}
```

Output

```
D:\test>javac InetAdd1.java

D:\test>java InetAdd1
Local Host is: aap/192.168.0.166
```

Program Explanation :

When you run the above program you get the IP address of the local machine o which you are running your program along with the host name. In above case, *aap* the name my machine on which I am running this program and 192.168.0.166 is m machine's IP address. Note that while running this program your machine should be network. If it is not in the networking (i.e. standalone machine) then you will get th output in the following manner,

```
D:\test>java InetAdd1
Local Host is: aap/127.0.0.1
```

Here 127.0.0.1 is a reserved IP address corresponding to the host computer which is also known as loop back address. 127.0.0.1 is used whenever program needs to access a network running on the local computer itself.

One more important thing about the above Java program and that is definition of main function. It should be

```
public static void main(String args[]) throws UnknownHostException
```

That means, it is a must to throw an exception for Unknown host in order to handle unknown host situation. Hence precisely UnknownHostException should be thrown.

Our programs which makes use of the methods in InetAddress class must have following statement at the beginning -

```
Import java.net.*
```

Remember that it is java.net package only which deals with InetAddress class functionalities. Let us have fun with another java program.

Java Program

```
/*********************************************************************
This program shows the use of the method getByName
for getting the IP address for the corresponding host name
*********************************************************************/
import java.net.*;
class InetAdd2
{
 public static void main(String args[]) throws UnknownHostException
 {
  InetAddress addr = InetAddress.getLocalHost();
  System.out.println(addr);
  Address=InetAddress.getByName("vtubooks.com");
  System.out.println(addr);
 }
}
```

Output

```
D:\test>java InetAdd2
aap/192.168.0.166
vtubooks.com/202.137.237.142
```

While running the above program if your computer is not connected to internet then it will generate following output [In fact here is the real use of throwing an exception]

```
D:\test>javac InetAdd2.java

D:\test>java InetAdd2
aap/127.0.0.1
Exception in thread "main" java.net.UnknownHostException: vtubooks.com
at java.net.Inet4AddressImpl.lookupAllHostAddr(Native Method)
at java.net.InetAddress$1.lookupAllHostAddr(Unknown Source)
at java.net.InetAddress.getAddressFromNameService(Unknown Source)
at java.net.InetAddress.getAllByName0(Unknown Source)
at java.net.InetAddress.getAllByName(Unknown Source)
at java.net.InetAddress.getAllByName(Unknown Source)
at java.net.InetAddress.getByName(Unknown Source)
at InetAdd2.main(InetAdd2.java:8)
```

To avoid such dirty output we can slightly modify the above program as follows -

Java Program

```
/*********************************************************************
This program shows the use of the method getByName
for getting the IP address for the corresponding host name
*********************************************************************/
import java.net.*;
class InetAdd2
{
public static void main(String args[]) throws UnknownHostException
 {
  try
  {
  InetAddress addr=InetAddress.getLocalHost();
  System.out.println(addr);
  Address=InetAddress.getByName("vtubooks.com");
  System.out.println(addr);
  }
  catch(UnknownHostException e)
  {
   System.out.println(e);
  }
 }
}
```

Output

```
D:\test>javac InetAdd2.java

D:\test>java InetAdd2
aap/127.0.0.1
java.net.Unknown
```

```
InetAddress
    addr=InetAddress.getByName("192.168.0.166");
  System.out.println(addr.getHostName());

}
}
```

Output

```
D:\test>javac InetAdd4.java

D:\test>java InetAdd4
aap
```

Types of addresses

Different types of IP addresses can be identified using various methods as follows -

1. isLoopbackAddress()

This method is a Boolean method which returns true if the address given to it is a loop back address (i.e. 127.0.0.1). The following program shows use of this method.

Java Program

```
/*
This program shows the type of IP address

*/
import java.net.*;
class InetAdd5
{
 public static void main(String args[]) throws UnknownHostException
 {
  InetAddress addr=InetAddress.getByName("127.0.0.1");
  if(addr.isLoopbackAddress())
    System.out.println("Its a Loop back address"+addr);
  else
    System.out.println("Its not a Loop back address" +addr);

 }
}
```

Output

```
D:\test>javac InetAdd5.java

D:\test>java InetAdd5
Its a Loop back address/127.0.0.1
```

2. isAnyLocalAddress()

This is a Boolean method which returns true if the system has a local IP address.

- User **level processes** or services generally use port numbers **>=1024..**
- A **socket** is basically an endpoint of a two-way communication link between two programs running on the network. Typically these programs are **server program and client program.** Thus socket is OS - controlled interface into which the applications can send or receive messages to and fro from another application.
- A socket is bound to a port number so that the TCP/UDP from transport layer can identify the corresponding application at destination.
- There are typically two types of sockets -

TCP sockets : which are denoted by streams and
UDP sockets which are denoted by datagrams.

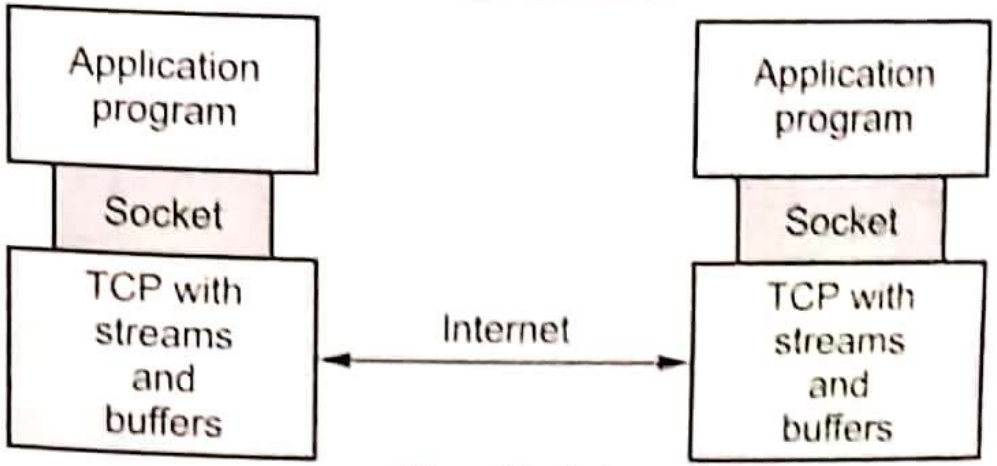


Fig. 10.4.1

- There are two constructors used to create **client socket** -

Constructor	Meaning
Socket(String *hostName*, int *port*)	Creates a socket connecting the local host to the named host and port.
Socket(InetAddress *ipAddress*, int *port*)	Creates a socket using a pre-existing **InetAddress** object and a port.

- The methods used for examining the socket are -

InetAddress getInetAddress( )	Returns the **InetAddress** associated with the **Socket** object.
int getPort( )	Returns the remote port to which this **Socket** object is connected.
int getLocalPort( )	Returns the local port to which socket object is connected .

10.5 Datagram Socket Class and Datagram Packet Class

- A **datagram** is an independent, self-contained message sent over the network whose arrival, arrival time and content are not guaranteed.
- It is a basic transfer unit associated with a packet-switched network. The applications that communicate via datagrams send and receive completely independent packets of information.
- The java.net package contains two classes to help you write Java programs that use datagrams to send and receive packets over the network : **DatagramSocket** and **DatagramPacket** .
- Java **DatagramSocket** and **DatagramPacket** classes are used for **connection-less socket programming.**
- An application can send and receive DatagramPackets through a DatagramSocket.
- **Datagram packets** are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet.
- A **datagram socket** is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed.
- Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

Constructor	Description
DatagramPacket(byte buff[] , int len)	Constructs a DatagramPacket for receiving packets of specific length.
DatagramPacket(byte buff[], int len, InetAddress addr, int port)	Constructs a datagram packet for sending packets of specific length to the specified port number on the specified host.

Methods for DatagramPacket

Constructor	Description
InetAddress getAddress()	Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received.
byte[] getData()	Returns the data buffer.
int getLength()	Returns the length of the data to be sent or the length of the data received.
int getOffset()	Returns the offset of the data to be sent or the offset of the data received.
int getPort()	Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.
SocketAddress getSocketAddress()	Gets the SocketAddress (usually IP address + port number) of the remote host that this packet is being sent to or is coming from
void setAddress(InetAddress iaddr)	Sets the IP address of the machine to which this datagram is being sent.
void setData(byte[] buf)	Set the data buffer for this packet.
void setData(byte[] buf, int offset, int length)	Set the data buffer for this packet.
void setLength(int length)	Set the length for this packet.
void setPort(int iport)	Sets the port number on the remote host to which this datagram is being sent.

11 Introduction to OOMD Techniques

02

Syllabus

Modeling concepts, Abstraction, The three models, Class Model, State model and Interaction model

Contents

11.1 Introduction ..... Summer-13 ..... Marks 4
11.2 Modeling Concepts
11.3 Modeling Design Technique
11.4 Abstraction
11.5 The Three Models
11.6 Class Model
11.7 State Model
11.8 Interaction Model
11.9 Relationship among the Models ..... May-12 ..... Marks 7
11.10 University Questions with Answers

11.1 Introduction

GTU : Summer-13 Marks 4

Object oriented analysis and design is a software engineering approach which models the system as interacting objects. Each object represents a system entity which plays vital role in building of that system. Each object has a state and behavior.

Object Oriented Analysis (OOA) focuses on analysis of functional requirements of the system. The Object Oriented Design (OOD) takes analysis model as input and produces implementation specification.

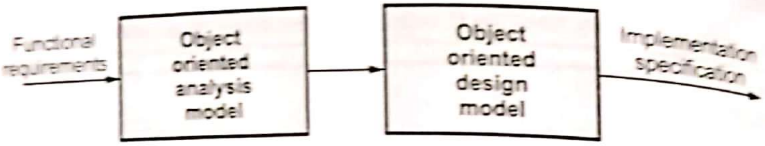


Fig. 11.1.1 Concept of OOAD

11.1.1 What is Object Orientation ?

Object orientation means organization of software as a collection of discrete objects. Each object consists of **data structure** and **behavior** which are closely coupled.

Following are some basic characteristics required by OO approach -

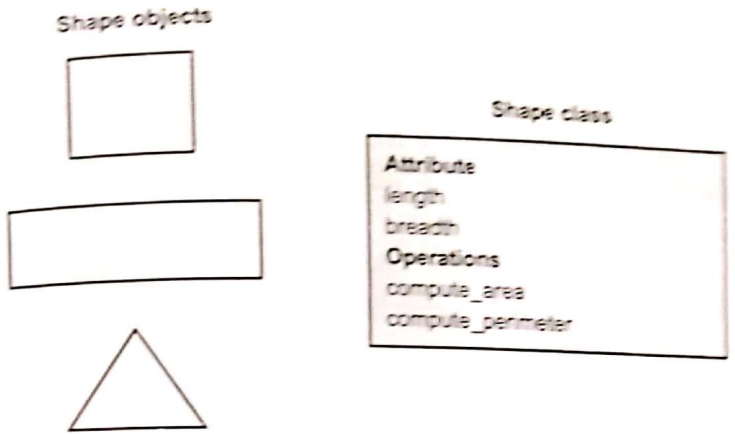
1. **Identity :** The identity means data is arranged in distinguishable entities called **objects**. Each object is treated as an inherent entity that means - each object is distinct even if their attribute values are same. In programming languages an object is referred by the memory references.

Suyash's ball

Neeta's pen

2. **Classification :** Classification is a technique in which objects of same data structure(attribute) and behavior(operations) are grouped into class. Each class can define infinite number of **objects and object is an instance of a class.** Each object can have its own values for attributes and operations but these objects can share the attributes and operations of other objects too.

For example - Shape can be a class and rectangle, triangle and square can be the objects of the class Shape. Following figure illustrates this idea -



3. **Inheritance :** Inheritance allows the class to share the property of another class in hierarchical manner. The **superclass** is a class whose properties are shared by the **subclass**. The **subclass** not only shares the properties of **superclass** but it can add its own unique properties. Thus the **super class** is a general class where as the **subclass** is a more specific class.

For example - **Animal** is a super class but the **Dog**, **Cat**, **Cow** are the subclasses. Each subclass has its own unique features. For example if the sound is an attribute of class **Animal** then the sound of Dog, Cat, Cow are different.

Due to inheritance property the repetition in the design can be avoided. Hence data reusability is the main ability in object oriented programming due to inheritance feature.

4. **Polymorphism :** Poly means many and morphs means forms. Hence polymorphism is a technique in which the same operation can be defined differently for the different classes. For example the operation **clean** can be used to clean the **dish** objects, **car** object or **vegetable** object.

In object oriented programming the language can automatically select the appropriate method to implement the required operation.

Example 11.1.1 *In object orientation if two objects are identical, does it mean they are equal ? Is object concrete or conceptual ?* GTU : Summer-13, Marks 4

Solution : In object orientation, if two objects are identical, that does not mean they are equal. The term **Identity** specifies that object is treated as inherent entity that means each object is distinct even if their attribute values are same. The object is an conceptual entity.

Review Question

1. *Explain following important terms - 1) Identity 2) Classification 3) Inheritance 4) Polymorphism.*

11.2 Modeling Concepts

- In OO development approach, the developer has to think in terms of applicat and therefore the data structures and functions are addressed effectively.
- In OO development the fundamental way of thinking is represented and the focu is not on the programming techniques.
- Using the modeling techniques the system that is to be built can be very well explained to specifiers, developers and the customers.

11.3 Modeling Design Technique

Model is abstraction of the system and is created for the sake of understanding. Only important aspects of the system are represented in the model.

Designers build various types of models - For example - Blueprints can be made before building the house. Drawings to show the machine parts can be created. Following are some purposes - "Why models are created ?"

- **Communication with customer** : Architects and system designers represent the model to their customers in order to give the mock demonstration of the system.
- **Testing of physical entity** : If the actual system is built and then tested then it is costly affair. Some sort of testing can be done by creating a physical model or computer model. The simulation of the system by the model is not only cheaper but it informs the developers about the problems that may come occur in actual system.
- **Visualization** : Using the model system can be represented for visualization. The major elements of the system, some important activities that can be performed by the system, useful events and transitions can be shown by the model. The customer can visualize the system due to model.
- **Complexity reduction** : The complexities of the system can be reduced by modeling the system and separating small number of important things to deal with them separately.

Review Question

1. *Write short note on - Modeling design technique*

11.4 Abstraction

- Abstraction means examining the selective area of the problem.
- The goal of abstraction is to separate out all the important aspects of the specific area of the problem and ignore all unimportant aspects of that area.

- There is always some purpose for the abstraction. Similarly many different abstractions can be possible for the same thing.
- All abstractions are incomplete or unimportant. In fact the abstraction is nothing but the small summary of particular problem. Describing simply summary of the problem is also useful thing. Because by abstracting the problem in this way, it becomes easy for us to understand the problem. Thus purpose of an abstraction is to limit the universe so that we can understand the problem.
- A good model should capture only the crucial aspects of the problem and should omit the others. Sometimes a model may contain certain extra details of the problem because of which developers attention gets diverted from the real issues. In such a sitation abstraction is the only thing which helps the developer to focus the attention on specific goal of the problem.

Review Question

1. *Explain the concept of abstraction.*

11.5 The Three Models

- There are three different viewpoints used to model the system.
- The **class model** represents the **static, structural, data** aspects of the system.
- The **state model** represents the **temporal, behavioral, control** aspect of the system.
- The **interaction model** represents collaboration of objects and **interaction** aspect of the system.
- Normally all these three aspects are **incorporated within a single system**. There is use of data structures (class model), operations of the system are executed in some specific sequence (state model) and data as well as control is passed among the objects (interaction model).
- Using these models **distinct views** of the system can be created.
- These models are not completely independent. There is **limited and explicit interconnection** among these objects. The good design isolates different aspects of the system and have limited coupling.
- These models **evolve** during the development.
- Analysts construct a model without thinking of implementation. The designers add solution constructs to the model. Implementers write the code for model and solution construct.
- Thus the model has **two dimensions** - First dimension includes the **views** such as class model, state model and interaction model. Other dimension includes the **stage of development** such as analysis, design and implementation.

- There is always some purpose for the abstraction. Similarly many different abstractions can be possible for the same thing.
- All abstractions are incomplete or unimportant. In fact the abstraction is nothing but the small summary of particular problem. Describing simply summary of the problem is also useful thing. Because by abstracting the problem in this way, it becomes easy for us to understand the problem. Thus purpose of an abstraction is to limit the universe so that we can understand the problem.
- A good model should capture only the crucial aspects of the problem and should omit the others. Sometimes a model may contain certain extra details of the problem because of which developers attention gets diverted from the real issues. In such a sitation abstraction is the only thing which helps the developer to focus the attention on specific goal of the problem.

Review Question

1. *Explain the concept of abstraction.*

11.5 The Three Models

- There are three different viewpoints used to model the system.
- The **class model** represents the **static, structural, data** aspects of the system.
- The **state model** represents the **temporal, behavioral, control** aspect of the system.
- The **interaction model** represents collaboration of objects and **interaction** aspect of the system.
- Normally all these three aspects are **incorporated within a single system**. There is use of data structures (class model), operations of the system are executed in some specific sequence (state model) and data as well as control is passed among the objects (interaction model).
- Using these models **distinct views** of the system can be created.
- These models are not completely independent. There is **limited and explicit interconnection** among these objects. The good design isolates different aspects of the system and have limited coupling.
- These models **evolve** during the development.
- Analysts construct a model without thinking of implementation. The designers add solution constructs to the model. Implementers write the code for model and solution construct.
- Thus the model has **two dimensions** - First dimension includes the **views** such as class model, state model and interaction model. Other dimension includes the **stage of development** such as analysis, design and implementation.

11.6 Class Model

- The class model describes the structure of the object, the relationship of one object with other objects, attributes and operations of the objects.
- The state and interaction models use the context of the class model.
- The goal of constructing class model is to capture the concepts from the real world that are useful for the application.
- While modeling the particular system the terms that are familiar to that system must be used.
- The analysis model should not contain the **computer constructs** but the design model contains the **computer constructs** and solution to the problem.
- The class model is represented by the **class diagram**. The classes in this diagram define the **attributes and operations** of each object. The **association relationship** is used to link the classes together. The **generalization relationship** is used for a group of classes to share common data structures and operations.

11.7 State Model

- The state model is concerned with the time and sequencing of the operations of the object.
- On occurrence of the events the object change their current state.
- The state model captures the **control** aspect of the system.
- The state model is represented by **state diagram.** Each state diagram represents state and the event sequences.
- Actions and events in the state diagram becomes the operations of the object in the class model. References between the state diagram become interactions in interaction model.

11.8 Interaction Model

- Interaction model describe how one object collaborates with other in order to achieve behavior of the system.
- The overall behavior of the system can be represented with the help of state and interaction model.
- The interaction model includes **use case diagram, sequence diagram and activity diagram**.
- The use case diagrams show how the outsider actors interact with the system to achieve the functionality.
- The sequence diagram represents the objects that interact and the time sequence of their interaction.
- The activity diagram represents flow of control among various objects.

12 Class Modeling

05

Syllabus

Object and class concepts, Link and association, Generalization and inheritance

Contents

12.1 Introduction to Class Modeling

12.2 Object and Class Concepts

12.3 Link and Association Concepts **May-12, Winter-12**

..... **Summer-13** ..... Marks 4

12.4 Generalization and Inheritance..... **Winter-12, May-12**

..... **Summer-13** ..... Marks 7

12.5 University Questions with Answers

12.1 Introduction to Class Modeling

- Class model is a **static model**.
- In this model the objects, the relationship shared by them, the **attributes** and **operations** for each class of the object are represented.
- This model **emphasises on the object** instead of functionalities.
- Class model gives the graphical representation of the system and it is useful for the customer to understand the system.

In this chapter, we will understand many fundamental concepts used in class modelling.

12.2 Object and Class Concepts

12.2.1 Objects

- The **main purpose** of class model is to describe objects.
- Object is an instance or occurrence of a class. It is a concept, abstraction or thing with identity and meaning.
- The objects can be conceptual entities, real-word entities, or important things from implementation point of view.
- The objects are normally nouns. The choice of object is done by judgements.
- All objects are distinguishable entities.
- For example - If **Student** is a class then **Anuja**, **Supriya** and **Shilpa** are the objects of the class students. Each student has its own *name*, *roll number* and *address*. Each object is different from other object. That is every object has its own **identity**.
- **Identity** means the objects are distinct due to their inherent existence and not by their description.

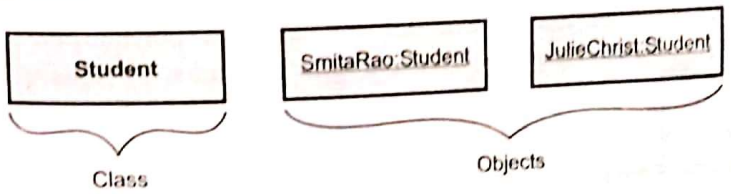
12.2.2 Classes

- Class is a group of objects having same attributes and operations, relationships and semantics.
- For example - Student, Employee, Company, Course all are classes. Each Student has rollnumber, name and address.
- The classes appear as common nouns or noun phrases.
- Objects of particular class have same attributes and behaviour but each object is different from the other due to values of its attribute, different behaviour and relationship with other

- Objects in a class share a common semantic purpose. For example both the **dog** and **cat** have the properties like tail and legs and they belong to the same class **Animal**.
- Each object belongs to some class and that object is aware of its belonging class.
- Grouping the objects into corresponding classes make the design abstract.

12.2.3 Class Diagrams

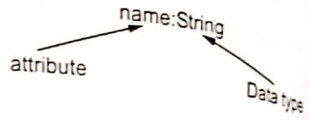
- The class model is represented by two types of diagrams - **Class diagram** and **object diagram**.
- Class diagram is a **graphical representation** used for modelling classes and their relationships. It describes all possible objects belonging to the classes.
- Class diagram is used for abstract modelling and for implementing actual program.
- The class diagram is concise and can be understood easily.
- It is most common to make use of class diagram in the OO modelling.
- The **object diagram** represents the objects and their relationships with each other.
- Using the one class diagram various object diagrams can be created.
- The object diagram is used for documenting the **test cases**.
- The class and object is represented graphically by the box. The class name must be written at the center of the box and the first letter is capitalised. It is written in boldface.
- The object name is followed by the colon and the class name. Everything must be underlined in the box that represents the object.
- The multi-words are written without space having the first letter of the separate word capitalized.



12.2.4 Values and Attributes

- Attribute is a named property of a class and it can hold some value.
- Adjectives in the problems statement generally represent some attributes.

- For example - Student class might have name, address, phone number as the attributes. Each attribute have values. For instance - name can be "AAA" for one object and can be "PPP" for another object.
- Different objects may have same or different values for its attributes.
- The name of the attribute is always unique within the class.
- The values are always assigned to the attributes. The values alone have no identity. They can be recognised with the help of belonging attributes. For instance - If you say "AAA" then it is just meaningless but if "AAA" is assigned as a name of the **Student** then it makes some sense.
- The attributes are always written in the second compartment of the class. It may be written along with its data type and default value. For example -



The default values can be written by using equal to sign and the value.

- Many times objects have **unique identifiers**. For example **studentNumber** is an unique identifier which is an internal identifier and it should not be considered as an attribute. But there are some real-world identifiers such as **RollNumber**, **EMPID**, **TelephoneNumber** and so on these must be mentioned as the attributes. In fact, these kind of attributes are important attributes.

12.2.5 Operations and Methods

- Objects have procedures or functions which are called as **operations**.
- All the objects in the same class share the common set of operations.
- For example - The class **Shape** can have various objects such as **rectangle**, **triangle** or **square** having the common set of operations such as - **move**, **draw**, **get_area** and so on.
- The same operation can be applied to different classes or it can be used for different purposes. Such operations are called as polymorphic operations. For example **get_area** operation can calculate the area of square or triangle or rectangle.
- **Method** is an implementation of operation for a class. Some piece of code is used to implement each method. For example code written for calculating the area of square is the method.
- Any number of arguments can be passed to the operations.
- **Signature** is nothing but the number and types of the arguments and type of the return value of the operation. For example, **int get_area(int l, int b)** - Here the two arguments int l and int b along with the return type int represent the signature of the operation.
- The generic word used for either attribute or an operation is called the **feature**.
- The operations are always written in the third compartment of the class box.
- The names of the operations must be written in regular face. The first letter of the operation must be in lower case. We can mention the names of the arguments along with the data type. The arguments can be specified within the parenthesis. Similarly we can specify the return data type of particular operation by preceding the colon. For example -

  get_area(l:int,b:int):int

12.2.6 Summary of Class Notations

As we know, the class can be represented using the box. The first compartment contains the name of the class, the second compartment contains the attributes and the third compartment contains the operations belonging to that class. We can specify the data types as well as the default values to the attributes and operations. The list of arguments can also be passed to the operations.

Class Name
$attribute_1$:dataType=defaultValue ... ... $attribute_n$:dataType=defaultValue
$operation_1$(argList):returnType ... ... $operation_n$(argList):returnType

Each argument of the operation can have **direction**. The direction can be input(in), output(out) or any modifiable mode(inout). Using the equal to sign the default value can be represented.

The attributes or operation compartments are optional. Missing attribute compartment means the attributes are not specified. Similarly missing operation compartment means that the operations are not specified.

But if we represent the empty compartment then that means the attributes or operations are specified but there are not attributes or operations.

- Due to **encapsulation** the operations are kept private to a class. But the association breaks this encapsulation, in the sense one class can be correlated with the other class. Thus hidden things in the program can be avoided and program can be extended further if needed.

12.3.2 Multiplicity

- Multiple objects can be related to some objects. The multiplicity represents "how many" objects are connected.
- The multiplicity can be written as expression. It describes "one" or "many" objects associated with other object.
- The multiplicity is specified at the end of the association link.
- Following are the notations that can be used to denote the multiplicity of associations -

Notations	Description
1	Only one instance
0..1	Zero or one instance
*	Many instances
0..*	Zero or many instances
1..*	One or many instances
2..10	You can specify the integer range

For example -

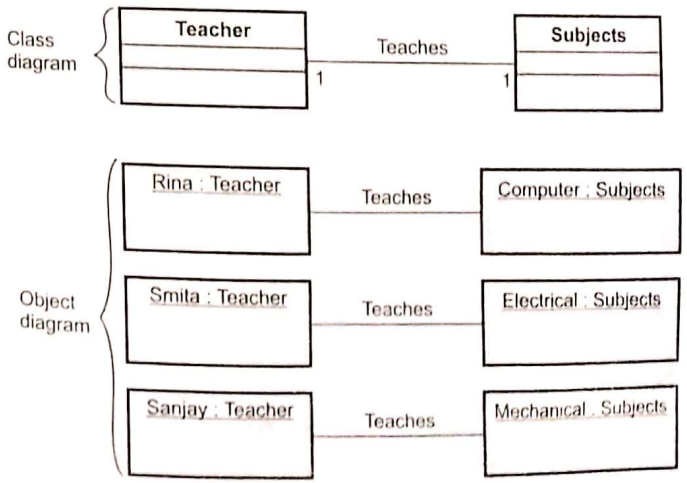


Fig. 12.3.5 Ordering the objects

The web browser displays the web pages in some specific order (first come first served). Hence the {ordered} keyword is used at the association end (the end at which the objects should appear in some specific order).

12.3.5 Bags and Sequences

In association relationship there are some objects that may be duplicated or may appear in some sequence. In these cases the **bags** and **sequences** are used at the association ends.

Bag is a collection of objects at other end that are duplicated. That means if the word bag appears at some end then the object at that end can appear more than once.

Sequence is a collection of objects at other end that are duplicated or appear in some sequence.

The qualifier will be **StudentID**.

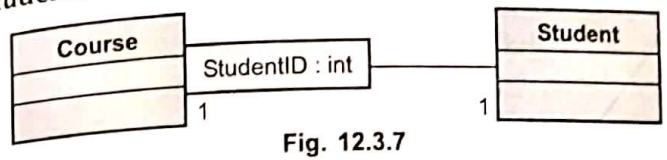


Fig. 12.3.7

A qualifier is a property which defines **selection key.** UML allows having a qualifier on each end of a single association. The multiplicity can be associated with the association link.

Review Questions

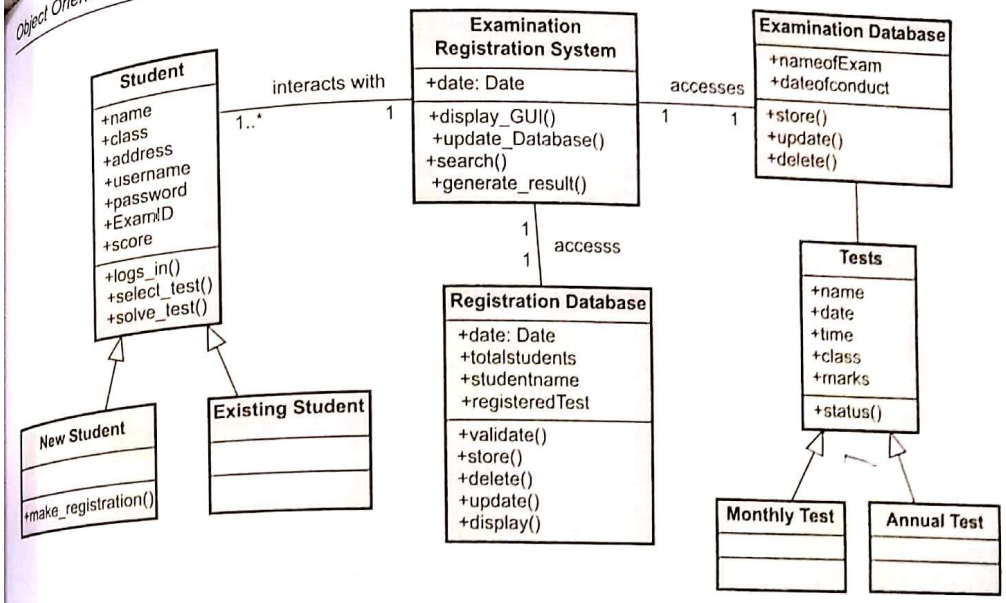
1. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Multiplicity* **GTU : May-12, Mark 1**
2. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Association Class.* **GTU : May-12, Mark 1**
3. *Is association class same as ordinary class ? Explain with example.* **GTU : Winter-12, Marks 4**
4. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Qualified Association.* **GTU : May-12, Mark 1**
5. *What is a qualified association ?* **GTU : Summer-13, Marks 3**

12.4 Generalization and Inheritance

GTU : Winter-12, May-12, Summer-13 Marks 7

12.4.1 Definition

- **Generalization** is the relationship between a superclass and one or more subclasses.
- Each subclass inherits the features of its superclass.
- The **superclass** is more **general** and it holds common attributes, operations and associations. Whereas the **subclass** is more **specific.** It uses the common attributes, operations and association of superclass but adds its specific attributes, operations and associations.
- Generalization is also called as **is-a** relationship. This is because, each instance of subclass is an instance of the superclass.

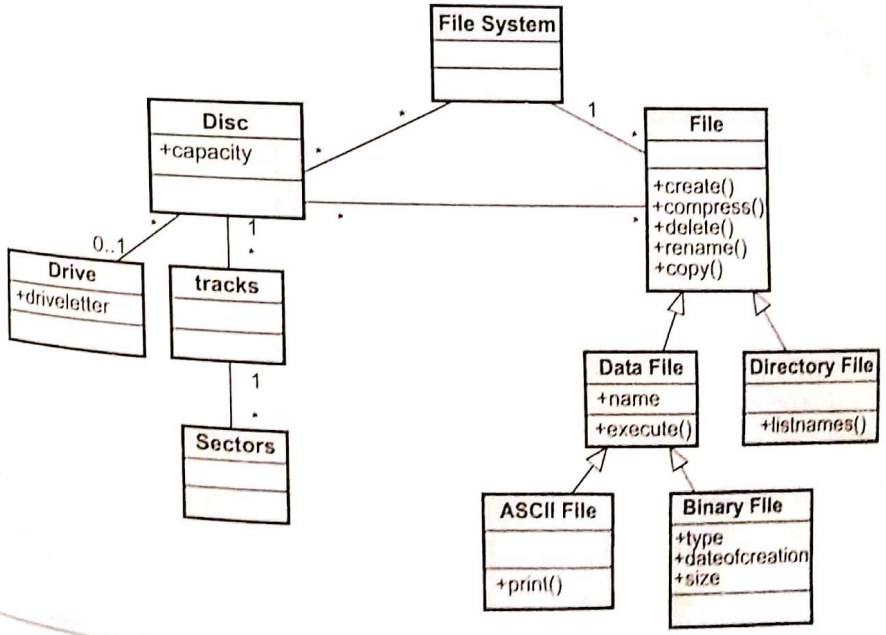


Example 12.4.1 *Prepare a class diagram for each group of classes. Add at least 10 relationships (association and generalizations) to each diagram.*

File system, file, ASCII file, binary file, directory file, disc, drive, track, sector

GTU : Winter-12, Marks 7

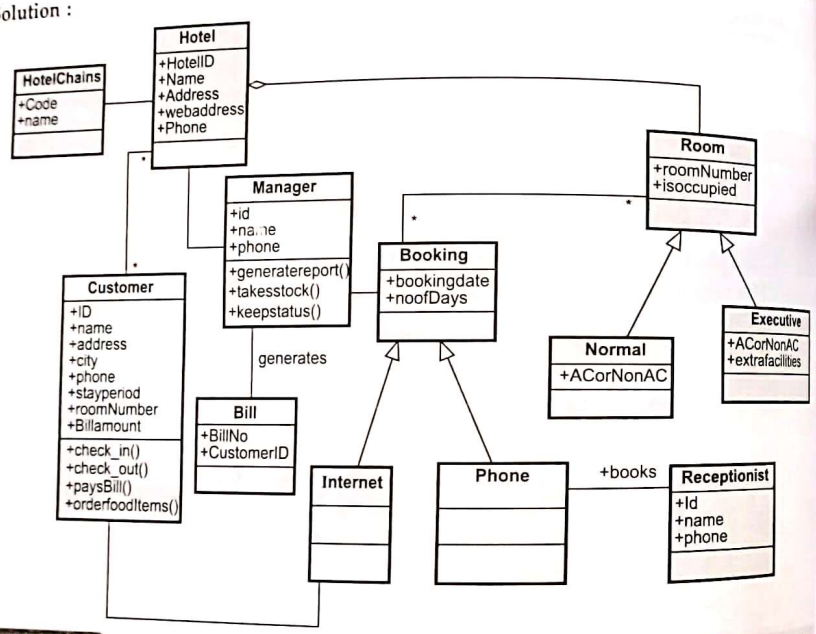
Solution : The required class diagram is given below -



Example 12.4.2 *Prepare a class model for the hotel management system. The system should support chain of hotels. A hotel contains two categories of rooms : executive and normal, both AC and non-AC. The customers of executive rooms can avail extra facilities like games, swimming, food service in rooms, etc. The booking is possible by Internet or by phone. If the booking is through phone, process is done by receptionist and if booking is done through Internet the process is carried out by customer through hotel website. Depending on the number of days customer stays, appropriate bill is generated. The bill also contains amount for transport, food and other facilities enjoyed by the customer along with necessary taxes. The manager should be able to generate reports like list of customers staying in the hotel, list of rooms empty, monthly/yearly income, etc.*

GTU : May-12, Marks 7

Solution :



Example 12.4.3 *Prepare a class model to describe geographical map. Map contains roads, rivers and mountains. All components are described by points representing longitude and latitude.*

GTU : Summer-13, Marks 3

Solution : See Fig. 12.4.2 on next page.

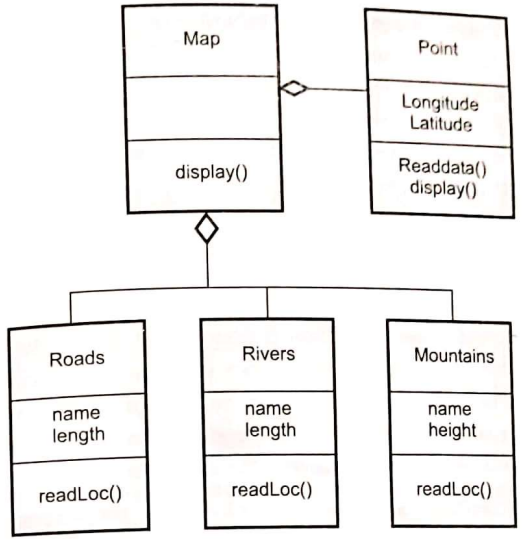


Fig. 12.4.2 Class model for geographical map

12.4.2 Use of Generalization

- There are **three main uses** of the generalization class -

1. **Polymorphism :** Polymorphism is an ability of object oriented design in which the operation can be called at the superclass level, and the same operation with the same name can be defined at the subclass level doing different functionalities. The compiler automatically selects appropriate operation that matches the object's class. Due to this feature the software design becomes flexible. The new subclasses can be added without changing the rest of the code.
2. **Structuring of description of objects :** Due to generalization, the designer can form the taxonomy of objects. Based on similarities and difference of the objects the classes can be arranged. That also means if there are some classes that share a **is-a** kind of relationship then using generalization concept these classes can be structured in hierarchical form.
3. **Reusability of code :** As the common properties defined by the superclass are also used by the subclasses, there is reusability of code. If a library of some useful or most commonly required classes is created then due to generalization, the designer can inherit some classes from the library. This actually brings reusability of the code.

- **Generalization**, **specialization** and **inheritance** are used for in a context reference. The terms generalization and specialization are complement to each other. The generalization refers to some common properties while specialization refers to the specific property or purpose. The inheritance is the technique which attributes, operations and relationships are shared by the set of classes using the generalization/specialization concepts.

12.4.3 Overriding Features

- **Overriding features** include overriding methods or default values of attributes.
- The subclass overrides the superclass features. That means the overriding features are redefined or replaced by some another value in the subclass, but they possess the same name. for instance - the **display** method of superclass can be overridden in subclass. It means we can redefine the **display** method in subclass having different functional body but possessing the same name.
- The overridden features are used in OO design for various reasons - i) For defining the different behaviour in the subclass ii) For having the specific functionality in the subclass.
- The signature or datatype must not be overridden.
- While overriding, attribute type, number of arguments and type of arguments and return type must not be changed.
- The overriding of general method to special method increases the overall performance of the code.

Example -

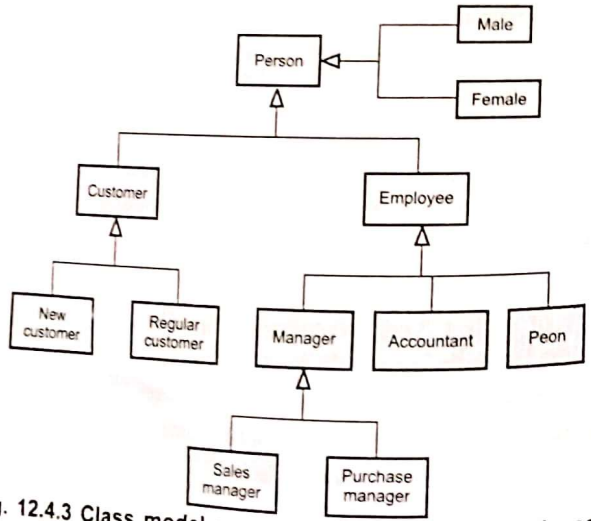


Fig. 12.4.3 Class model representing hierarchies of classes

12.5 University Questions with Answers

(Regulation 2008)

May 2012

Q.1 *Define the peurpose of the following term with suitable example and UML notation with respect to class model - Multiplicity* **[Refer section 12.3]** [1]

Q.2 *Define the purpose of the following term with suitable example and UML notation with respect to class model - Association Class.* **[Refer section 12.3]** [1]

Q.3 *Define the purpose of the following term with suitable example and UML notation with respect to class model - Qualified Association.* **[Refer section 12.3]** [1]

Dec. 2012

Q.4 *Is association class same as ordinary class ? Explain with example.* **[Refer section 12.3]** [4]

May 2013

Q.5 *What is a qualified association ?* **[Refer section 12.3]** [3]

13 Advanced Class Modeling

05

Syllabus

Advanced object and class concepts, Association ends, N-ary associations, Aggregation, Abstract classes, Multiple inheritance, Metadata, Constraints, Derived data, Packages.

Contents

13.1 Advanced Object and Class Concepts
13.2 Association Ends
13.3 N-ary Associations
13.4 Aggregation ..... **Dec.-11, May-12, Winter-12,** Marks 3
13.5 Abstract Classes
13.6 Multiple Inheritance
13.7 Metadata ..... **May-12** ..... Mark 1
13.8 Constraints ..... **May-12** ..... Mark 1
13.9 Derived Data
13.10 Packages ..... **May-12** ..... Mark 1
13.11 University Questions with Answers

13.1 Advanced Object and Class Concepts

13.1.1 Enumerations

The enumeration is a primitive data type which can have finite set of values and can be modelled as classes. For example - Job can have the status such as Processing, Idle and Waiting. These values can be represented by designing **Status** as class of type <<enumeration>>. Note that in the following figure, the keyword **enumeration** is written in Guillemets i.e. << >>. The list of values are given in the second compartment.

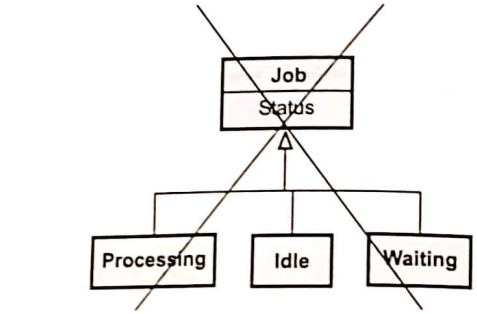
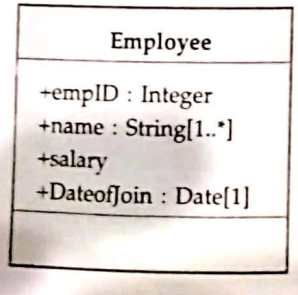


Fig. 13.1.1 Modeling enumeration

13.1.2 Multiplicity

Multiplicity denotes how many objects are associated with particular object. Multiplicity can be associated with the associations as well as attributes. Normally in databases the multiplicity is associated with the attributes. For example -



13.1.3 Scope

Scope represents whether particular feature is available for class or object. In UML the scope can be **static scope** or and **object** or **instance scope.**

1. ***Object scope*** - Each object of a class has its own value. It requires no notation for specifying the instance.
2. **Static scope** - All the instances of a class have just one value. This is referred as static scope. The static scope is mentioned by underlining the attribute.

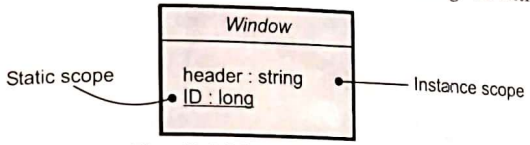


Fig. 13.1.2 Instance and scope

The private attributes of class makes use of static scoped features.

13.1.4 Visibility

The visibility specifies the how the attributes and operations are visible in the system. There are four levels of visibility -

1. **public :** The public element is visible to all elements that can access the classifier. It is denoted by '+' symbol.
2. **protected :** The protected element is visible to all the elements that have a generalization relationship to the namespace that owns it. **Protected** visibility is represented by '#' symbol.
3. **private :** A private element is only visible inside the namespace that owns it. Private visibility is represented by '-' symbol.
4. **package :** Only the elements that are declared in the same package can use this feature. It is specified using the symbol '~'.

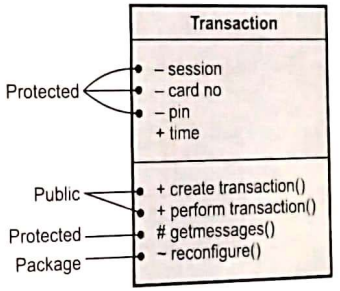
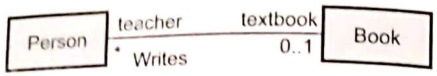


Fig. 13.1.3 Visibility

13.2 Association Ends

- Association end is an end of association.
- Binary association has two ends, ternary association has three ends and so on.
- **Properties of association end** - There are various properties of association end and these are enlisted below -

- **Association End Name :** For clear understanding of the association relationship we can write the names at the association. The traversing direction can be understood due to association end names.



Teacher	textbook
Rita	EVS
Smita	Mathematics

Fig. 13.2.1 Association end names

- **Multiplicity :** Multiple objects can be related to one particular object. It can be denoted by "1", "*", "1...*" and so on.
- **Ordering :** This is a type of association which is used to denote the set of objects at one end must appear in specific order.

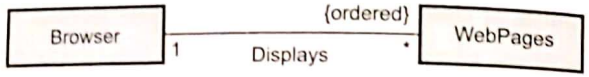


Fig. 13.2.2 Ordering the objects

- **Qualification**

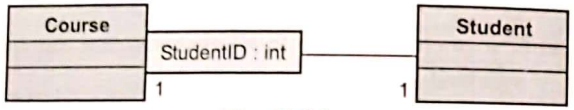
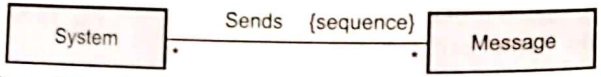


Fig. 13.2.3

- **Bags and Sequences :** In association relationship, there are some objects that may be duplicated or may appear in some sequence. For that purpose bags and sequences are used.



- **Aggregation :** It is a kind of association in which one object is a part of another object.
- **Changeability :** The update status of an association end is denoted by this property. The values for this property can be *changeable* and *readonly*.
- **Navigability :** By default the association can be traversed in both the directions however by showing the arrowhead at one end we can represent the navigability from one class to other.
- **Visibility :** The association ends can be public, private protected or package.

13.3 N-ary Associations

The n-ary association is the association among three or more classes. Normally having n-ary association in the class diagram is not preferred and therefore it is a practice to split n-ary association into binary association.

Following is a simple association that can be seen as n-ary association but it can be represented using binary association.

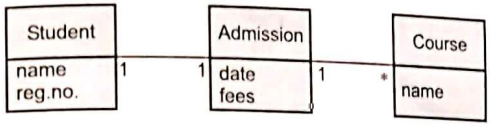


Fig. 13.3.1 n-ary association represented by binary association

Following is a typical example of n-ary association in which three classes take part actively. Note that such an association is represented using a diamond notation. The course management system can maintain the records for the students who are studying the courses in particular academic year.

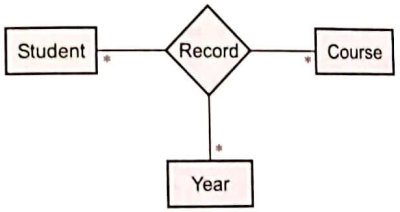


Fig. 13.3.2

Following is an another representation in which four classes are taking part making the association.

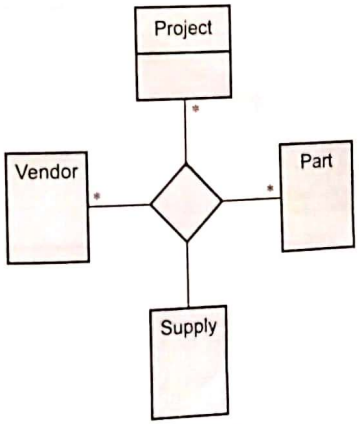


Fig. 13.3.3

13.4 Aggregation

GTU : Dec.-11, May-12, Winter-12, Marks 3

- Aggregation is a type of association.
- It is used to represent **whole-part** relationship.
- It normally posses **has-a** relationship.
- It is denoted as follows -

- **For example** - DVD player and Car are the two classes that can be associated by aggregate relationship. Note that DVD player can exists without car and car can be without a DVD player. We can read this relationship as "DVD player is a part of Car"

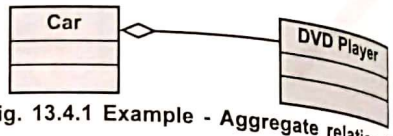


Fig. 13.4.1 Example - Aggregate relationship

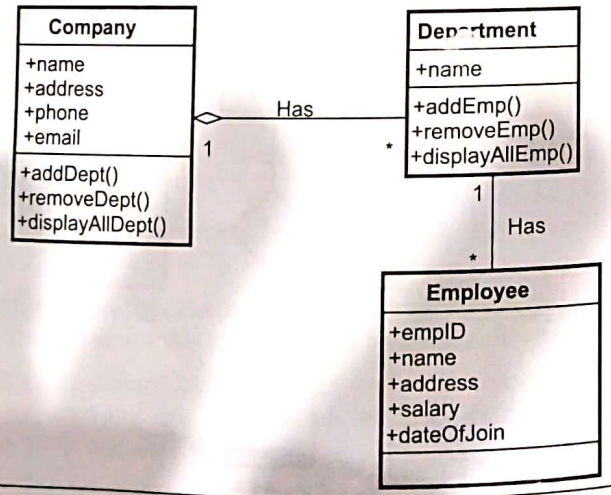
- **Transitivity** - It is an important property of aggregation. It means of class X is a part of class Y and if class Y is a part of class Z then class X becomes the part of class Z.
- **Antisymmetric** - According to this property of aggregation if class X is a part of class Y then class Y is not a part of class X.

13.4.1 Difference between Aggregation and Association

The aggregation is a special kind of association.

In aggregation relationship the two objects share a **whole-part** relationship. But if two objects are independent but still there are related to each other then the association relationship is indicated.

The use of aggregation instead of association often comes by judgement of designer.



13.4.2 Difference between Aggregation and Composition

Aggregations and Composition is two forms of **whole-part** relationship. The composition is restricted aggregation. That is, when one object contains another object and if contained object can not exist without the existence of the container object, then it is called composition. It is denoted as follows -

For example - Steering and Car are the two classes that are associated by the composite relationship. This is restricted relationship because a car can not exist without the steering.

Fig. 13.4.2 Example - Composite relationship

We can read the relationship as "Steering is a part of Car". The arrowhead is pointing towards the **whole** class.

Difference between aggregation and composition

Both the composition and aggregation represent the whole part relationship but the composition is more restricted than the aggregation. The composed object can not exist without the other object. Hence it is a strong relationship.

This restriction is not there in aggregation. The existence of contained object is entirely optional in case of aggregation.

For example - Books Library contains books and students. The student and Library share the aggregate relationship but the Book and Library share the composition relationship. As without books the library is not possible.

13.4.3 Propagation of Operations

Propagation of of operations is a mechanism which allows to move particular operation through the network of objects. For example - Following Fig. 13.4.3 shows that

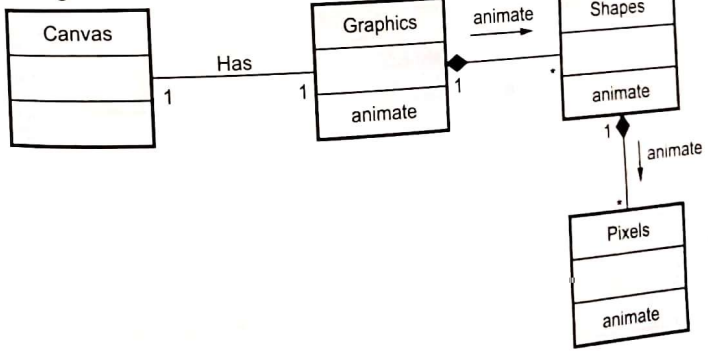


Fig. 13.4.3 Propagation

purposes. The abstract polymorphic function is incomplete and is overridden by the child class functions.

13.6 Multiple Inheritance

Definition : The multiple inheritance is a kind of inheritance in which the derived classes are derived from more than one super classes. The class with more than one superclass is called **join class**.

This kind of implementation may lead to some conflicts in the child classes due to multiple paths from superclass. But such ambiguities can be eliminated during the implementation.

Example

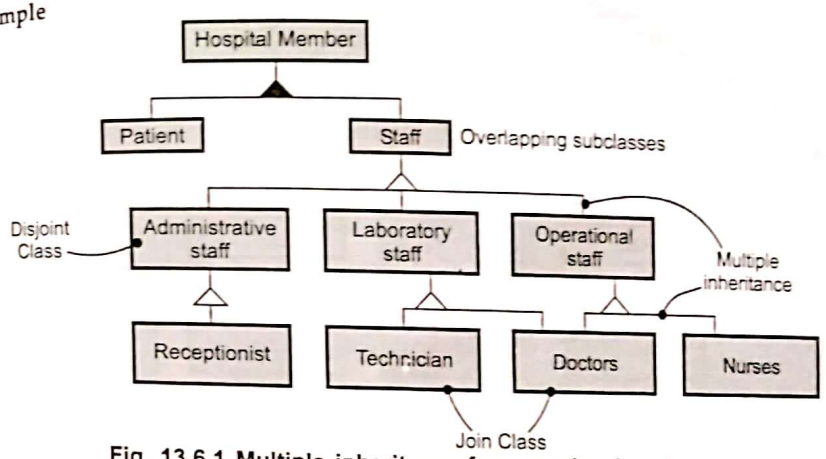


Fig. 13.6.1 Multiple inheritance from overlapping classes

If some parent has no more than one child in multiple inheritance then that class is disjoint class.

The hollow triangle indicates the disjoint subclasses whereas the solid triangle indicates the overlapping subclasses.

The Fig. 13.6.1 shows disjoint class.

Advantages :

1 The single classes can perform **multiple functionality**.
2 The **reusability** of the code gets increased.

Disadvantages :

1. The conceptual and implementation model becomes **complex**.
2. In multiple inheritance - There is a **mixing of rules and multiple paths** can be followed for reaching to derived classes, conflicts might arise in the implementations.

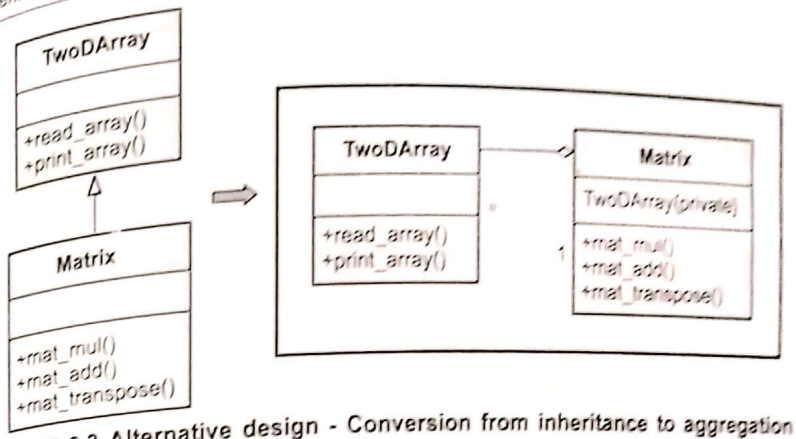


Fig. 13.6.3 Alternative design - Conversion from inheritance to aggregation

4. **Nested generalization :** In this technique, one generalization is factored then second, then third and so on. Thus all possible combinations of generalizations are factored. This preserves the inheritance but it duplicates the declarations and the code. The principles of object oriented techniques are also violated by this approach.

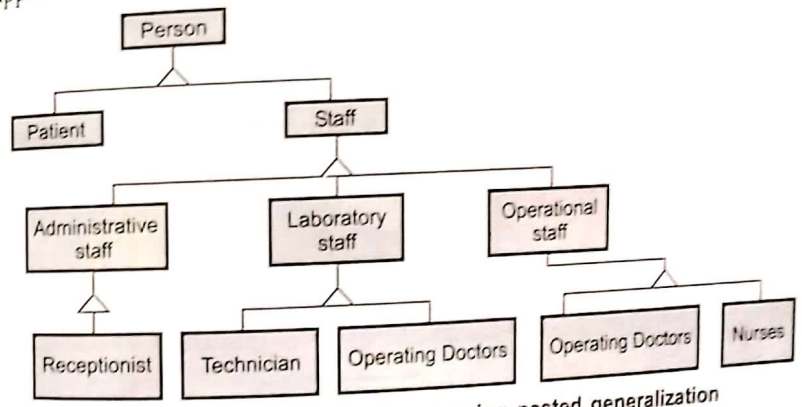


Fig. 13.6.4 Multiple inheritance using nested generalization

Issues in Workaround

Following are some issues while selecting the workaround -

1. If a subclass have several superclasses and all the superclasses are equally important then **make use of delegation** and **bring the symmetry** in the model.
2. If **one superclass** is more **important** than rest of the superclasses then implement **single inheritance** and **apply delegation.**
3. **Avoid nested generalization** if **large amount** of code get duplicated.
4. If there are **less** number of **combinations** then consider **nested generalization.**

5. In the nested generalization, **factor most important criterion** first and then next most important and so on.
6. If one superclass is having more features than other superclasses then **preserve the inheritance** through the path.
7. Maintain **strict identity** of the model.

Review Questions

1. *Discuss in brief multiple inheritance.*
2. *What do you understand by multiple inheritance ? Describe with suitable example*
3. *What is multiple inheritance and what are its associated problems ? How does the concept of inheritance of specifications help to overcome these problems ? Explain.*

13.7 Metadata

GTU : May-12, Mark 1

Metadata is a data that describes other data. The class definition is basically a metadata. In UML many times it is preferred to model the metadata as a separate class.

For example - Consider a banking system, in which the records for the customer and their accounts is maintained. A separate description class AccountDescription is maintained which describes the account number.

Many times classes have their own classes. These classes are called as metaclasses.

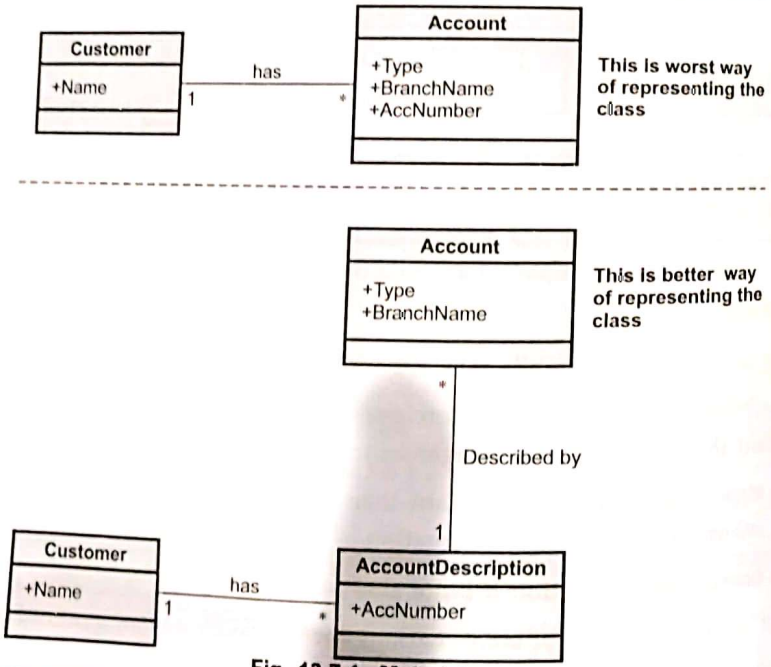


Fig. 13.7.1 Metadata

Review Question

1. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Derived Data.*

GTU : May-12, Mark 1

13.8 Constraints

GTU : May-12, Mark 1

Constraints are commonly used in class diagram. In UML constraint is a condition or restriction on UML element. These elements can be objects, classes, attributes, links, associations and generalization sets. It must evaluate to Boolean value - true or false. Constraint must be satisfied by a correct design of the system. OCL i.e. Object Constraint Language is a constraint language predefined in UML. However, constraint can be specified by some other languages such as Java, Machine readable languages and natural languages.

13.8.1 Constraints on Object

Following is a simple example that shows how the constraint can be applied on the object. Normally, the constraint can be given in the form of expression and is specified within the curly brackets.

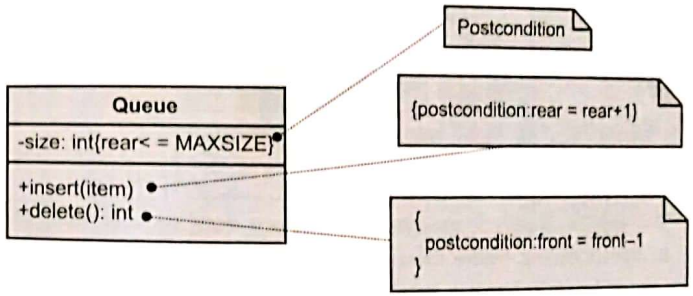
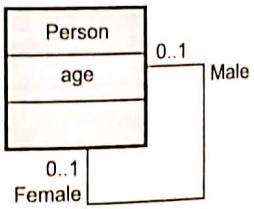


Fig. 13.8.1 Three ways to use constraints



13.8.2 Constraints on Generalization Sets

There are four types constraints on the Generalization sets. These are,

1. Complete

The complete specifies that all the children in the generalization are specified completely there is no specification possible.

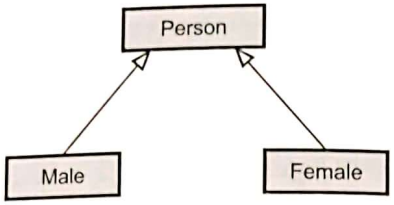


Fig. 13.8.2 Complete generalization

2. Incomplete

In the incomplete generalization not all the children in the generalization are specified completely.

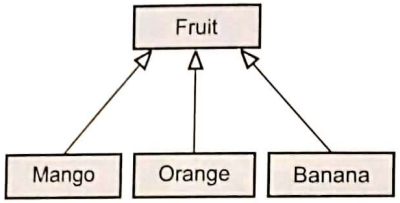


Fig. 13.8.3 Incomplete generalization

3. Disjoint

It specifies that objects of parent may not have more than one of the children as type. For example - In the following figure the class **AdministrativeStaff** has only one child.

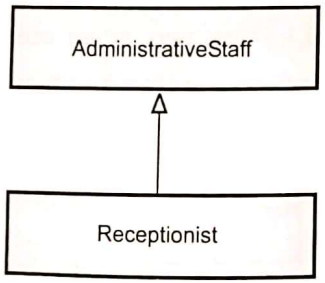


Fig. 13.8.4 Disjoint constraint

4. Overlapping

It specifies that objects of parent may have more than one of the children as type. For example - Here train can be passenger type vehicle or goodscarrier.

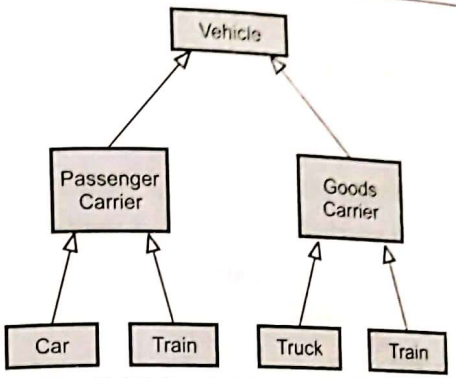


Fig. 13.8.5 Overlapping generalization

13.8.3 Constraints on Links

The constraints on the links can be allowing the multiplicity. The multiplicity denotes how many objects are related to the given object. Multiplicity for an attribute specifies how many values are possible for that attribute.

By applying constraints on link we can make the objects to appear in some specific order. The constraint {order} can be applied for this link. Following figure represents how constraint can be applied on the links -

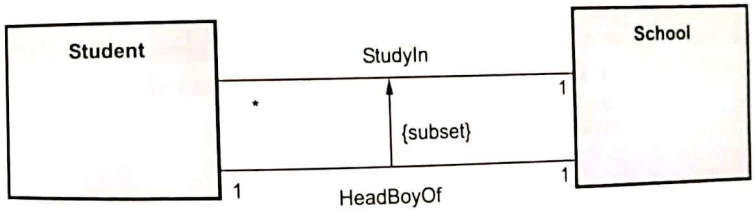


Fig. 13.8.6 Subset constraints on links

13.8.4 Use of Constraints

In UML we can specify the constraints in two ways - first by writing the constraints in curly braces and second by specifying the constraints in a dog-eared comment box. Refer Fig. 13.8.1 and Fig. 13.8.6.

Review Question

1. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Constraint.* **GTU : May-12, Mark 1**

13.9 Derived Data

Derived data is a function which is obtained using one or more functions or data elements.

The derived data is redundant because it is obtained from other element.

In class diagram, data attributes, associations and classes can be derived.

The notation for derived data is use of slash "/" in front of the element name.

Following figure represents the derived attribute. In the banking system, the account balance can be derived using the withdrawal.

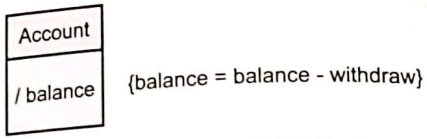


Fig. 13.9.1 Derived Attribute

Following is an example in which derived association and derived class represented for the baking system.

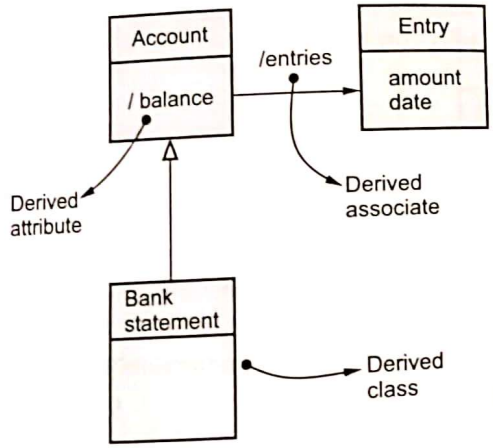


Fig. 13.9.2 Derived class and association

GTU : May-12, Mark 1

13.10 Packages

In UML, package is a general purpose mechanism for modeling the elements into groups. Using packages the elements can be organized properly. The package can be represented as shown in the Fig. 13.10.1.

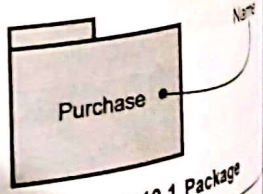


Fig. 13.10.1 Package

Review Question

1. *Define the purpose of the following term with suitable example and UML notation with respect to class model - Package.* GTU : May-12, Mark 1

13.11 University Questions with Answers

(Regulation 2008)

Dec. 2011

Q.1 *Explain the following : this, super, final* **[Refer section 13.4]** [3]

May 2012

Q.2 *Define the purpose of the following term with suitable example and UML notation with respect to class model - Aggregation.* **[Refer section 13.4]** [1]

Q.3 *Define the purpose of the following term with suitable example and UML notation with respect to class model - Derived Data.* **[Refer section 13.7]** [1]

Q.4 *Define the purpose of the following term with suitable example and UML notation with respect to class model - Constraint.* **[Refer section 13.8]** [1]

Q.5 *Define the purpose of the following term with suitable example and UML notation with respect to class model - Package.* **[Refer section 13.10]** [1]

Winter 2012

Q.6 *Explain aggregation. Is it same as association ?* **[Refer section 13.4]** [3]

❑❑❑

14 State Modeling

08

Syllabus

Events, States, Transition and conditions, State diagram, State diagram behavior

Contents

14.1 Introduction

14.2 Events

14.3 States ..... May-12 ..... Marks 2

14.4 Transition and Conditions

14.5 State Diagram ..... Winter-12, Summer-13 ..... Marks 7

14.6 State Diagram Behavior ..... May-12, Winter-12, Summer-13 ..... Marks 7

14.7 Nested States

14.8 University Questions with Answers

14.1 Introduction

State model describes the sequence of operations that occur in response to external stimuli. The state model is represented by using the **state diagram**. The state diagram is a graphical representation in which the events and states are represented. Events represent external stimuli and the state represents values of objects.

14.2 Events

- An **Event** is a specification of noteworthy occurrence of something that has location in time and space. For state model, the event is an occurrence of stimulus because of which the state **transitions** take place. For example - user switch off the alarm. The timer counts down.
- The events are often denoted by the **verbs in past tense**.
- One event may follow another event, or events might occur independently.
- The two events that are independent of each other are called **concurrent events**.
- If the time difference between occurrence of two events is very large then the events are called concurrent events. If the event occurs at distant places then also they are called as concurrent events. The ordering between concurrent events need not be specified.
- There are various types of events, the most commonly used events are signal events, change events and time event.

14.2.1 Signal Event

- A message is a named object that can be passed asynchronously by one object to another. The **signal** is a type of the message or the classifier for the message.
- Signals are similar to the classes.
  - The signals have instances just like the classes.
  - The signals allow to use the generalization relationship so that the hierarchy of events can be modeled. Because there might be some general events and some might be the specific events. For example: **BeepSound** signal can be **BootingUpBeep** or **DiskFailureBeep**
  - The signals might have the attributes and operations associated with them.
- There are various ways by which the signal can be sent -
  - In state machine, the signal can be sent by **action of transition.**
  - For the two roles of interaction, the signal can be modeled as a message.
  - By executing some method, the signal can be sent. Thus the signal specifies the behavior of the element.

- In UML the signal can be represented as stereotyped class or a dependency relationship. When the signal is modeled as class, it has the stereotype <<signal>> and then the signal is modeled by executing some message it is modeled using the stereotype **<<signal>>**
- **Example :**

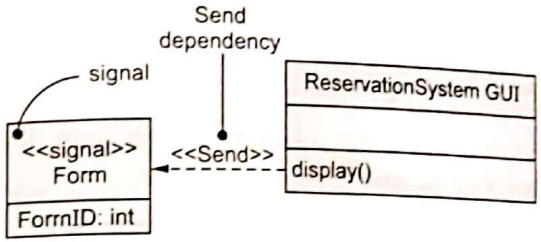


Fig. 14.2.1 Signal

As the Fig. 14.2.1 shows that the signal can be modeled as stereotyped classes or we can use a dependency relation to show a signal. (In Fig. 14.2.1 it is shown using the stereotype **send**)

14.2.2 Change Event

- The **change event** represents a change in state or satisfaction of some condition. The keyword **when** followed by some Boolean expression is used for denoting the change from false to true.
- For example - when (temp<10 °C) , when(power > maximum limit)

14.2.3 Time Event

- The time event represents the passing of time. The keyword **after** followed by some expression is used for denoting the time event. This expression represents the period of time. For example : After 5 seconds
- The occurrence of time event can be at absolute time. This can be represented using the keyword **at** . For example **at(1September 2012, 6:00 UT)** specifies that on 1st September 2012 at universal time 6 O'clock.

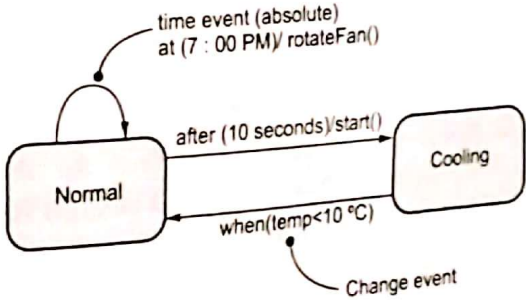


Fig. 14.2.2 Time and change event

- Following Fig 14.2.2 represents the time and change event in the state model -

GTU : May-12, Marks 2

14.3 States

- **State** is a condition or situation during the life of an object during which it satisfies some condition or performs some activity, or waits for some event.
- Object remains in particular state for finite amount of time. For example, Microwave might be in various states such as **Start** (Initial State when switched on), Activating(waiting for some buttons to get pressed), Activa (working/progressing) and Stopped.
- The state can be represented in UML by rounded box containing some name. The name is optional but it represents the name of the state. It is written in boldface at center of the box and the first letter is capitalized.

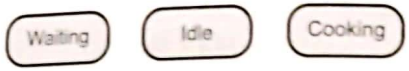


Fig. 14.3.1

- If the attributes of the state do not affect the behavior of the object then we do not represent them in the state.
- **Difference between event and state**

Events represent particular moment of time whereas the events represent interval of time. A state corresponds to the interval between two events received by the object. Both events and state depend upon the level of abstraction.

Review Question

1. *Differentiate state and event.* GTU : May-12, Marks 2

14.4 Transition and Conditions

- Transition is a relationship between two states. It indicates that the object in the first state will perform some action and enter in some another state when specific condition gets satisfied or some event occurs. When the state gets changed then it is said that the transition is fired.
- Before the transition to get fired the object is in source state and after the transition gets fired the object is in target state.
- For example : The **Microwave** changes its state from **Start** to **Activating** when the **OneMinute** button gets pressed (for cooking the food). Here pressing the button is an event that causes transition from one state to another.

- There are five parts of transitions -

1. **Source state :** This is the starting state for the transition and is affected by the transition.
2. **Event trigger :** It is recognized by the object in the source state. It causes the transition to get fired.
3. **Guard condition :** It is a Boolean expression. It is evaluated when the transition is triggered. When the expression is evaluated to true then the transition is fired, and then it is evaluated to false the transition is not fired.
4. **Effect :** An executable behavior such as action that acts on the object.
5. **Target state :** The active state after completion of transition.

- **Example :**

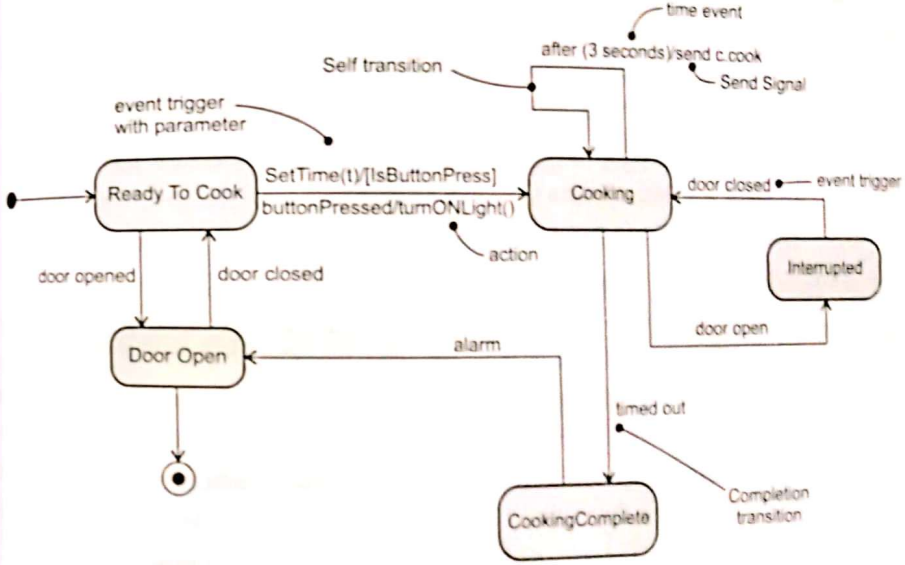


Fig. 14.4.1 Transitions

- **Event trigger :**
  - Event is occurrence of stimulus that can trigger a state transition.
  - Event can be calls, signals, passing of time or a change in state.
  - An event in the form of a signal or a call may have parameters that are available when the transition occurs. The guard conditions and actions are also available for the event triggers with parameters.
  - The completion transition can also be represented with no event trigger.

- **Guard condition :**
  - It is a Boolean expression which is placed inside the square bracket.
  - It is placed after trigger event.
  - A guard condition is evaluated only after the trigger event for its transition occurs.
  - Multiple transitions with same trigger event and from same source can be possible.
  - At the time when the event occurs, the guard condition is evaluated just once for each transition and if the transition is retriggered then the guard condition is evaluated once again.
  - The condition of the object can also be specified along with the guard condition.
- **Effect :**
  - Effect is the behavior which is executed when the transition fires.
  - Various examples of effects are - operation calls, sending and destructing another object, inline computations, sending signal to an object and so on.
  - When the sending of signal is shown in the state machine then the signal name is prefixed by the keyword **<<send>>**
  - When the state machine is not executing any effect of previous transitions then only the transitions occur.

14.5 State Diagram

GTU : Winter-12, Summer-13, Marks 7

- The state diagram is a **graph** that represents the **flow of control** from state to state.
- In this graph the nodes represent the **states** and the connecting edges represent the **transitions** between the states.
- The sequence of states is due to the events that occur in the system.
- The name of the states must be unique. The state should lie within the scope of the state diagram.
- All objects of a class diagram execute the state diagram for that class. That means the states of various objects can be represented using the state diagram.
- The **implementation of state diagram** means converting the semantic(meaning) of the states and transitions into the programming code.

- **State model** consists of multiple state diagrams probably representing behavior of one class for each state diagram. The diagrams in the state model must match at their interfaces.

Sample state diagram

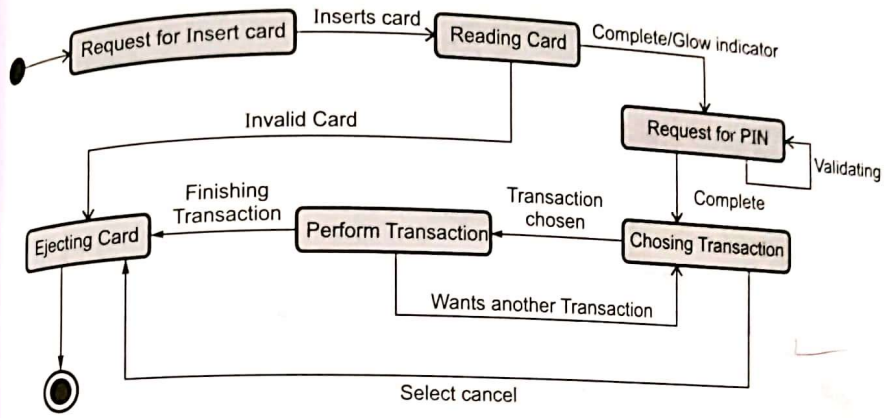


Fig. 14.5.1 State diagram for ATM transaction

In this state diagram, the state sequence caused when the customer interacts with the ATM machine for performing transactions is represented.

At the start the ATM machine is in idle state and when the customer interacts with the ATM machine then it requests for insertion of the ATM card. When the customer inserts the card, it is validated. Then the PIN is requested by the machine. The PIN entered by the customer is then validated. Only the valid PIN and Card holder is allowed to perform further transaction (such as withdrawal/ deposition of money or even checking the account balance). On completion of the transaction, the ATM card will be ejected by the machine.

Note that this state diagram does not expose all the states or the values of the object. For instance - if the PIN is invalid then ejecting the card- this sequence is not shown in the diagram, instead, the sequence that is followed on the valid PIN is represented.

14.5.1 One Shot State Diagram

The state diagram that has finite number of states representing one-shot life cycle or representing continuous loop is called One shot state diagram.

The one shot state diagram represents the initial and final states with finite number of states. For example following Fig. 14.5.2 represents the one shot state diagram.

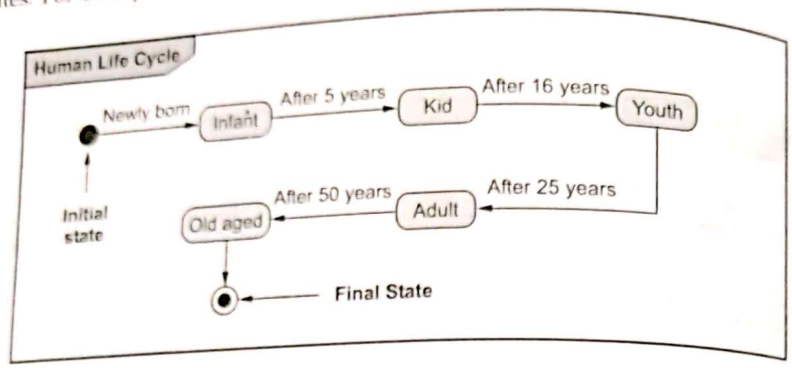


Fig. 14.5.2 One shot sequence diagram

The alternative notations for the **initial** and **final states** are

Initial State	○
Final State	⊗

Example 14.5.1 *What does one shot diagram represent ? Show one shot diagram for chess game.* **GTU : Winter-12, Summer-13, Marks 7**

Solution : One shot diagram : Refer section 14.5.1.

Following Fig. 14.5.3 represents the one shot diagram for chess game -

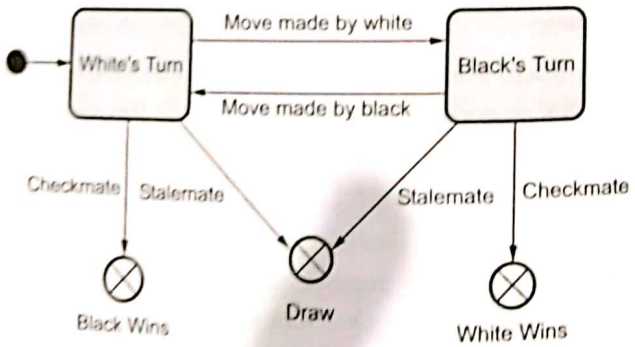


Fig. 14.5.3

14.5.2 Summary of Basic Notations

Following are some basic notations that are commonly used in the state diagram.

State : State is a condition of the object at particular moment of time. This time is particularly between the events. For example -

1. Waiting for user to enter PIN 2. Authenticating user 3. Dispensing cash.

The state is represented by the rounded box. There are special notations used for representing the initial and final states. (Refer section 14.5.1)

Transition : The relationship between two states is indicated using transition. When some event occurs then the object moves or transits from one state to another. This is represented using transition. The transition is represented by an edge with arrow head. The arrow head points to the target state.

Event : An event is significance occurrence of something. For example - Customer inserts ATM card.

Following figure represents these concepts.

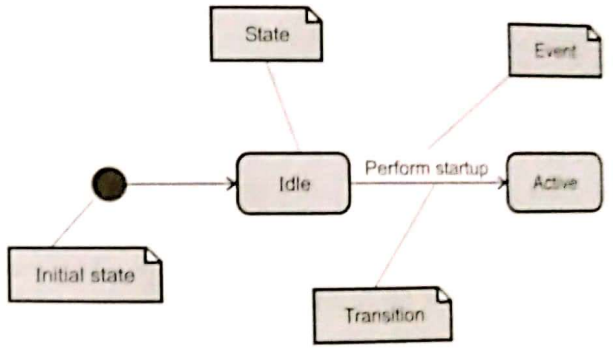


Fig. 14.5.4 State events and transitions in state diagram

State diagram : This is basically a graph that contains the states as nodes and edges as transitions. It is enclosed in a rectangular frame with the diagram name in a small pentagonal tag in upper left corner.

Guard condition :

- It is a Boolean expression which is placed inside the square bracket.
- It is placed after the trigger event.
- The guard condition is evaluated only after the event is triggered and represented along its transitions.

Effects : It is an executable behavior such as **action** in response to event. It typically acts on the object. It is represented along the transition and listed after "/"

The one shot state diagram represents the initial and final states with finite number of states. For example following Fig. 14.5.2 represents the one shot state diagram.

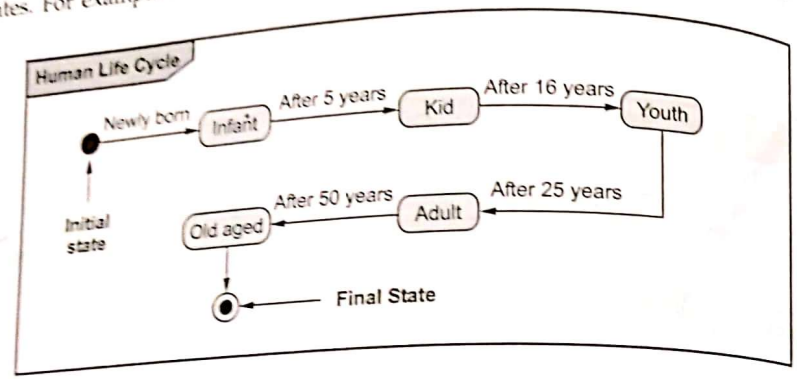


Fig. 14.5.2 One shot sequence diagram

The alternative notations for the **initial** and **final states** are

Initial State	○
Final State	⊗

Example 14.5.1 *What does one shot diagram represent ? Show one shot diagram for chess game.* **GTU : Winter-12, Summer-13, Marks 7**

Solution : One shot diagram : Refer section 14.5.1.

Following Fig. 14.5.3 represents the one shot diagram for chess game -

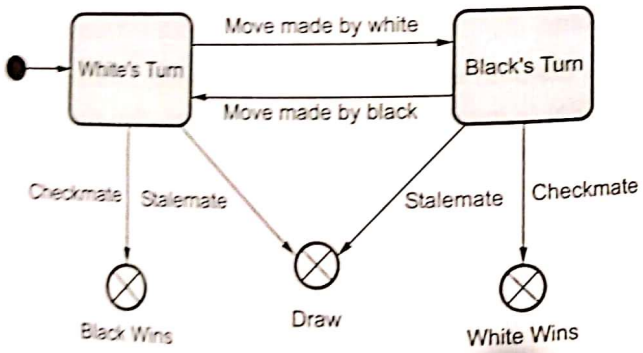


Fig. 14.5.3

14.5.2 Summary of Basic Notations

Following are some basic notations that are commonly used in the state diagram.

State : State is a condition of the object at particular moment of time. This time is particularly between the events. For example -

1. Waiting for user to enter PIN 2. Authenticating user 3. Dispensing cash.

The state is represented by the rounded box. There are special notations used for representing the initial and final states. (Refer section 14.5.1)

Transition : The relationship between two states is indicated using transition. When some event occurs then the object moves or transits from one state to another. This is represented using transition. The transition is represented by an edge with arrow head. The arrow head points to the target state.

Event : An event is significance occurrence of something. For example - Customer inserts ATM card.

Following figure represents these concepts.

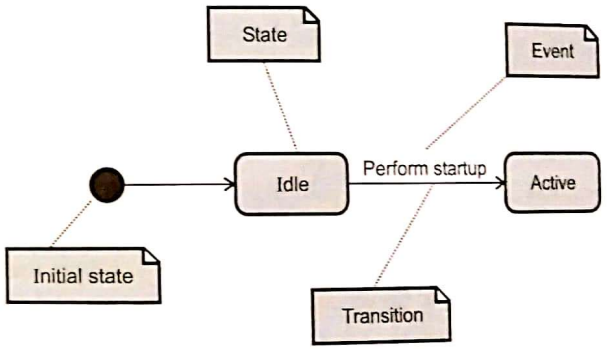


Fig. 14.5.4 State events and transitions in state diagram

State diagram : This is basically a graph that contains the states as nodes and edges as transitions. It is enclosed in a rectangular frame with the diagram name in a small pentagonal tag in upper left corner.

Guard condition :

- It is a Boolean expression which is placed inside the square bracket.
- It is placed after the trigger event.
- The guard condition is evaluated only after the event is triggered and represented along its transitions.

Effects : It is an executable behavior such as **action** in response to event. It typically acts on the object. It is represented along the transition and listed after "/".

For example

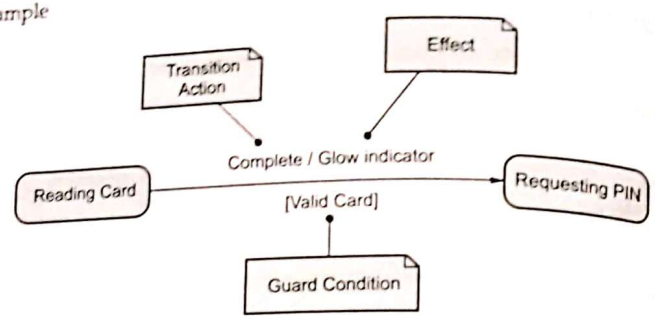


Fig. 14.5.5 Guard condition and effects in state diagram

Review Question

1. *Define : a) State b) Guard condition.* **GTU : Winter-12, Marks 4**

14.6 State Diagram Behavior

GTU : May-12, Winter-12, Summer-13, Marks 7

The state diagram will make use of some more notations to represent how object goes through different transitions. Let us see these effects one by one -

14.6.1 Activity Effects

- It is an executable behavior such as **action** in response to event. An **activity** is an actual behavior that may occur due to any number of effects.
- Activity can be performed at certain times such as - i) along the transition ii) on entry and exit of the states or iii) on occurrence of some event within the state.
- The activities are also used to represent the internal control operations. For example 'generate beep sound' when temperature exceeds 100° C.
- The activity can be represented by "/" and the activity name following the event due to which the activity occurs.
- Using the keyword **do** the ongoing activity can be represented.
- Following is a simple state diagram which represents the use of activity within it.

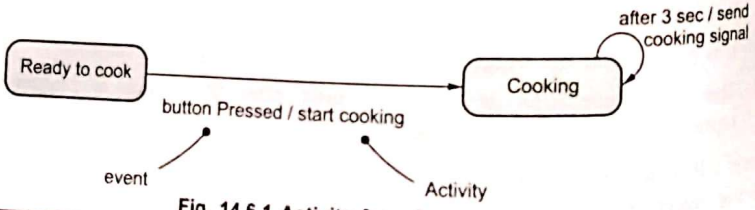


Fig. 14.6.1 Activity for microwave oven

14.6.2 Do-Activities

- When object is in some state then that object may perform some work. This ongoing activity is modeled by **do-activities**.
- The object continues with its activities until it is interrupted or the activity is performed completely.
- The notation **"do/"** is used to represent the do-activities.
- The do activity occurs within in the state and can not be attached to the transition.
- For example - In the Washing machine, When the washing activity is on going then the counter is getting decremented. This can be represented using the keyword **do**.

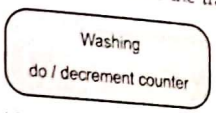


Fig. 14.6.2 Do-activity for washing machine

14.6.3 Entry and Exit Activities

- There are some activities that can be performed on entry of particular state. These activities are called **entry activities.** Similarly, while exiting the state some clean up actions need to be performed. These activities are called **exit activities.**

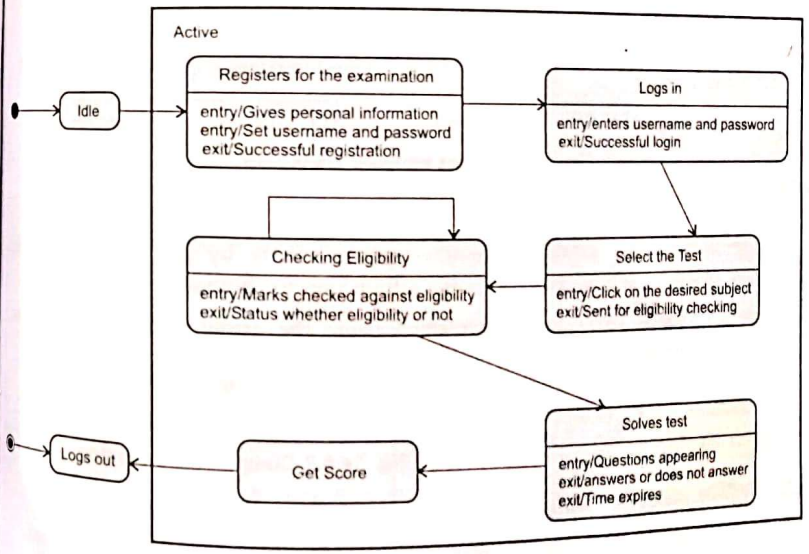


Fig. 14.6.3 Representing entry-exit activities in state diagram

- While specifying these activities the keyword **entry** and **exit** must be prefixed by that action.
- The exit activities are less common that entry activities but those are used by some events.
- Following Fig. 14.6.3 represents the **entry and exit** activities (See Fig. 14.6.3 on previous page)
- If a state has multiple activities then they will occur in following order -
  - Activities on incoming transitions.
  - Entry activities
  - Do activities
  - Exit activities
  - Activities on outgoing transitions
- The events might interrupt the ongoing activities. Even if the do activity is interrupted the exit activity will executed.
- **Difference between Events within a state and self transition :** The events within the state cause an activity to be performed. On the other hand the self transition causes entry and exit activities to be executed. For example -

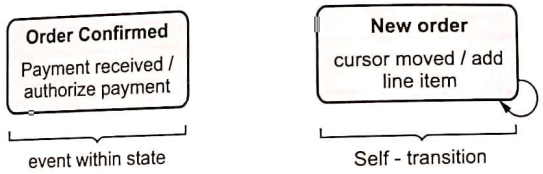


Fig. 14.6.4 Event and self transition

14.6.4 Completion Transition

- State diagrams are created to complete some activity by performing sequential tasks. When an activity gets completed a transition to another state occurs.
- The completion transition is represented using the arrow without event name. Such unlabeled transitions are called the completion transitions. For example - Refer Fig. 14.6.5.

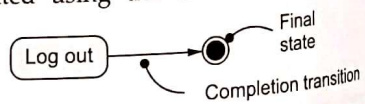


Fig. 14.6.5 Completion transition

- Once the completion transition occurs no completion event occurs.
- Normally guard conditions are not represented along the completion transitions

14.6.5 Sending Signals

- Due to occurrence of some event the signals can be sent from one object to another.
- The objects can interact with each other by sending the signals.
- For example - the **cooking** state a signal **timeout** to **Cookingcomplete** state.
- When a states accept several events from more than one object then it might affect the final states. This is called the **race condition**. The race condition is undesired in the system design.

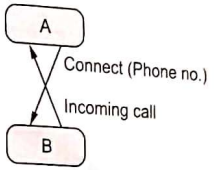


Fig. 14.6.6 Race condition

14.6.6 Examples

Example 14.6.1 *State diagram for stock maintenance system.*

Solution :

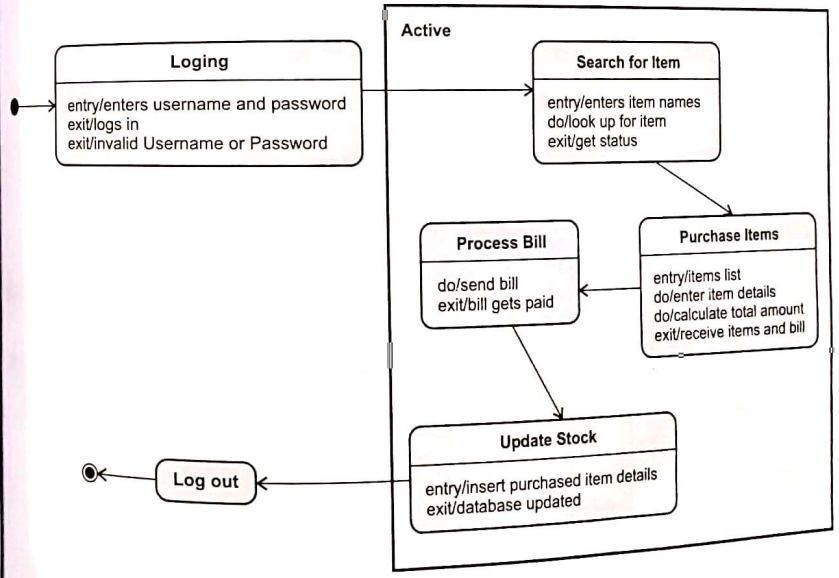


Fig. 14.6.7

Example 14.6.2 *State diagram for online reservation system.*

Solution :

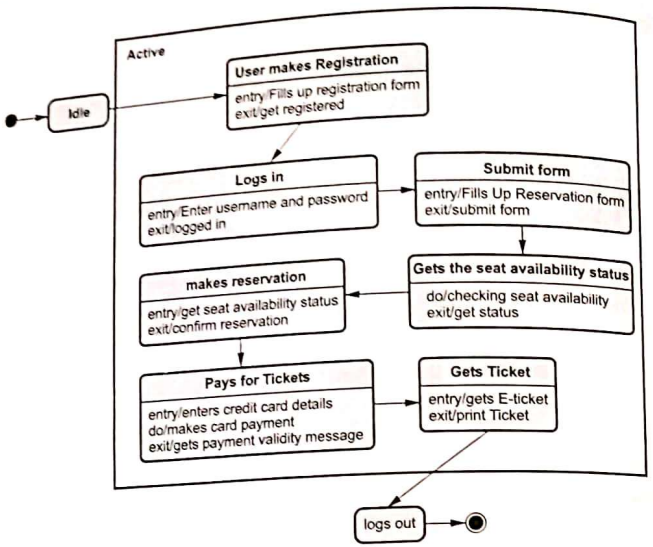


Fig. 14.6.8

Example 14.6.3 *State diagram for elevetor.*

Solution :

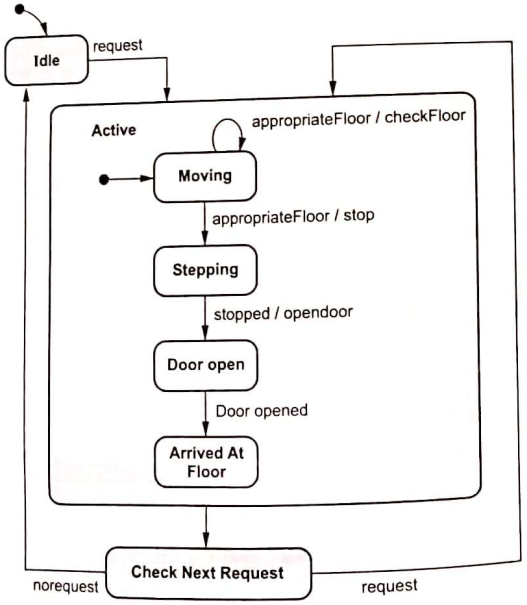


Fig. 14.6.9 Modeling lifetime of object

Example 14.6.4 *State diagram for home security system.*

Solution :

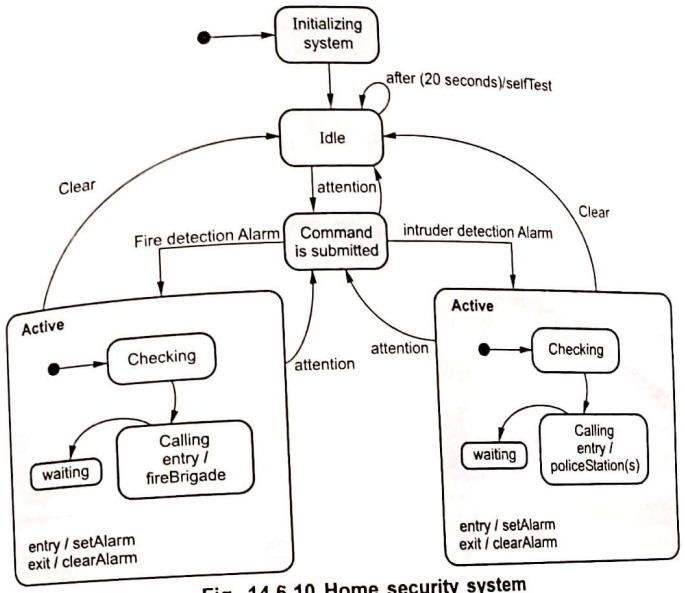


Fig. 14.6.10 Home security system

Example 14.6.5 *State diagram for microwave oven.*

Solution :

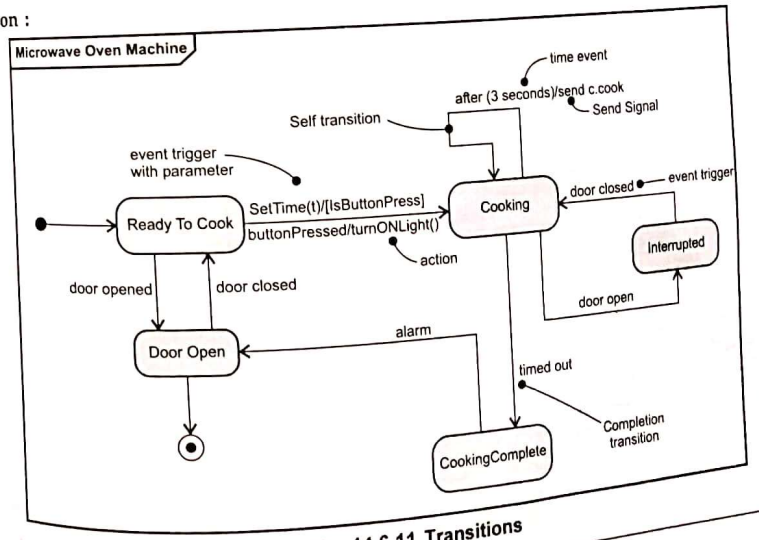


Fig. 14.6.11 Transitions

...ution : The state model for answering machine is as given below.

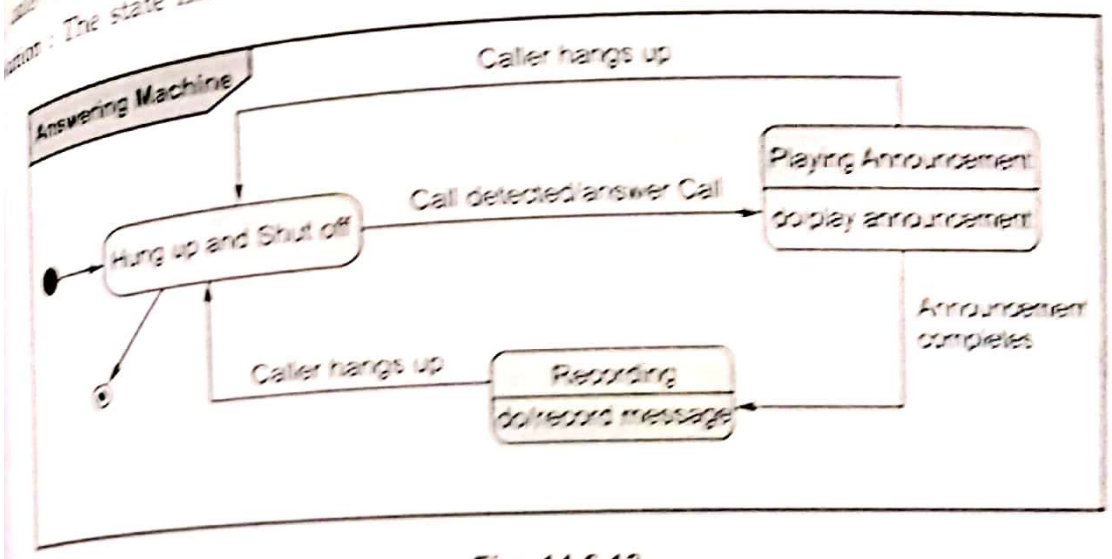


Fig. 14.8.13

Example 14.8.8 *Draw state diagram for the control of a telephone answering machine. The machine detects an incoming call on the first ring and answers the call with a prerecorded announcement when the announcement is complete, the machine records the caller's message. When the caller hangs up, the machine hangs up and shuts off. What changes must be made if machine answers after five rings ?* **GTU : Summer-13, Marks 7**

Solution : **State diagram for telephone answering machine :** Refer example 14.8.7.

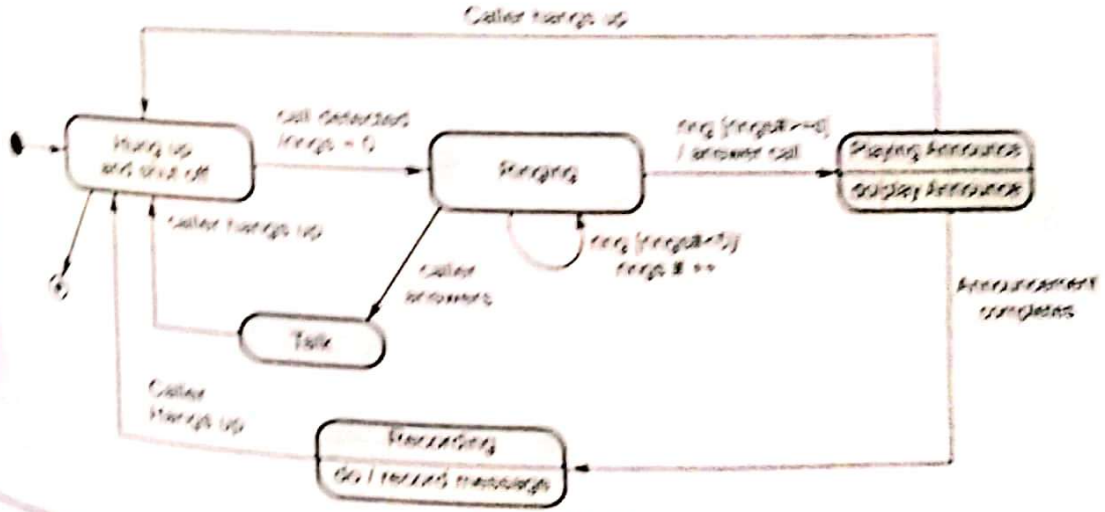


Fig. 14.8.14

Changes that need to be made if machine answers after five rings are represented by state diagram - (Refer Fig. 14.6.14 on previous page)

Example 14.6.9 *In a personal computer, a disk controller is typically used to transfer a stream of bytes from a floppy disk drive to a memory buffer with the help of a host such as the Central Processing Unit (CPU) or a Direct Memory Access (DMA) controller. Draw state diagram for the control of the data transfer. Add the following to the diagram : reset indicate data not available, indicate data available, data read by host, new data ready, indicate data lost.* **GTU : Winter-12, Marks 7**

Solution :

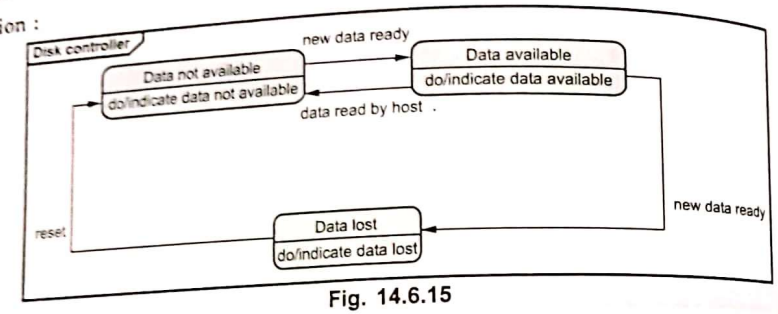


Fig. 14.6.15

14.7 Nested States

The nested states are the states that occur within the one super state. For example in the following state machine diagram, the **Active** state is a super state in which the sub states such as **Display, Reading Card Validating PIN** and so on are present. When the Perform shutdown event occurs then the system is restored to the **Idle** state from the **Active state.**

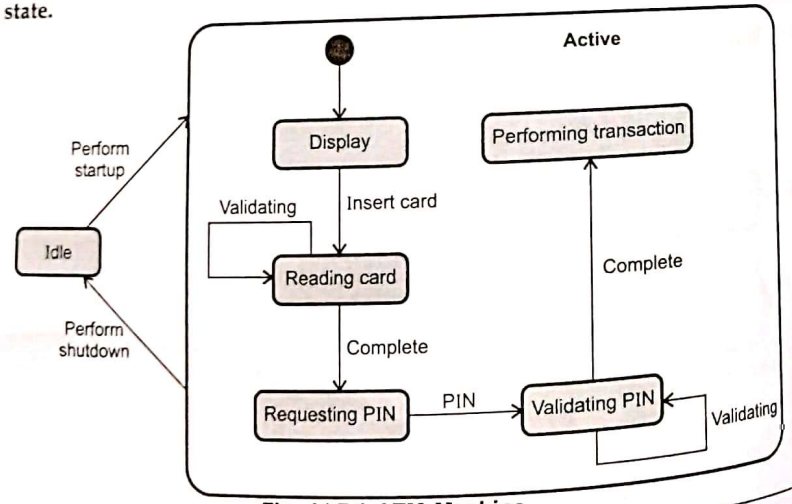


Fig. 14.7.1 ATM Machine

14.8 University Questions with Answers

(Regulation 2008)

May 2012

Q.1 *List different types of events. Identify states and events for Photocopier(Zerox) machine from the description given below and draw the state diagram for the same. Initially the machine is off. When the operator switches on the machine, it warms up during which it performs some internal tests. Once the tests are over, the machine is ready for making copies. When operator loads a page to be photocopied and press 'start' button, the machine starts making copies according to number of copies selected. While machine is making copies, machine may go out of paper. Once the operator loads sufficient pages , it can start making copies again. During the photocopy process, if paper jam occurs in the machine, operator may need to clean the path by removing the jammed paper to make the machine ready.* **[Refer example 14.6.6]** [7]

Q.2 *Differentiate state and event.* **[Refer section 14.3]** [2]

Winter 2012

Q.3 *Define : a) State b) Guard condition.* **[Refer section 14.5]** [4]

Q.4 *What does one shot diagram represent ? Show one shot diagram for chess game.* [7]
[Refer example 14.5.1]

Q.5 *Draw state diagram for the control of a telephone answering machine. The machine detects an incoming call on the first ring and answers the call with a prerecorded announcement. When the announcement is complete, the machine records the caller's message. When the caller hands up, the machine hangs up and shuts off. Place the following in the diagram : call detected, answer call, play announcement, record message, caller hangs up, announcement complete.* [7]
[Refer example 14.6.7]

Q.6 *In a personal computer, a disk controller is typically used to transfer a stream of bytes from a floppy disk drive to a memory buffer with the help of a host such as the Central Processing Unit (CPU) or a Direct Memory Access (DMA) controller. Draw state diagram for the control of the data transfer. Add the following to the diagram : reset indicate data not available, indicate data available, data read by host, new data ready, indicate data lost.* **[Refer example 14.6.9]** [7]

Summer 2013

Q.7 *What does one shot diagram represent ? Show one shot diagram for chess game.* **[Refer example 14.5.1]** [7]

Q.8 *Draw state diagram for the control of a telephone answering machine. The machine detects an incoming call on the first ring and answers the call with a prerecorded announcement when the announcement is complete, the machine records the caller's message. When the caller hangs up, the machine hangs up and shuts off. What changes need to be made if machine answers after five rings ?* **[Refer example 14.6.8]** [7]

❑❑❑

15 Interaction Modeling

10

Syllabus

Use case models, Sequence models, Activity models.

Contents

15.1 *Introduction*

15.2 *Use Case Model* ..... **Winter-12, Summer-13** .... Marks 7

15.3 *Sequence Model* ..... **Summer-13** ..... Marks 7

15.4 *Activity Model* ..... **Winter-12, Summer-13** .... Marks 7

15.5 *Use Case Relationships* ..... **May-12, Winter-12** ..... Marks 7

15.6 *Procedural Sequence Models* ..... **May-12** ..... Marks 7

15.7 *Special Constructs for Activity Models* ..... **May-12** ..... Marks 7

15.8 *University Questions with Answers*

15.1 Introduction

- There are three models that are used in UML modeling - Class model, state model and interaction models. The class model describes the objects in the systems and their relationship.
- The state model represents the life cycle of the objects and interaction model represents how the objects interact with each other.
- In **interaction modeling**, the **interaction among the objects** is represented and the useful results that are produced in this interaction is highlighted.
- The **state model** and **interaction model** describe the **behavior** of the system. These two models are complementary to each other by viewing the behavior of the system in different perspective.
- There are different levels of abstractions used in interaction modeling. Different types of modeling is used at these levels. It includes **use case modeling, sequence diagrams and activity diagrams.**
- Each **use case** represents the functionalities of the system. Using use cases the informal requirements of the system can also be captured.
- The **sequence model** represents how the messages are exchanged among the objects over time. These messages represent the signals and procedure calls. The sequence of operations is represented using this kind of modeling.
- The **activity diagram** represents the flow of control among the communicating objects. It shows the data flow as well as the control flow.

15.2 Use Case Model

GTU : Winter-12, Summer-13, Marks 3

Use case model focus on the behavior of the system. It exposes the functionalities that can be provided by the system to its user.

15.2.1 Actor

- **Actor** represents the role when users interact with the use cases.
- It is direct external user of the system. It is object or set of objects that communicates directly with the system but it is not the part of the system.
- Each object defines the manner in which the behavior of the system is expected. For example the actor **Student** will enroll for the course in a **course registration system.**
- An object can bound to multiple actors to represent the various behavioral aspects of the system. For example the object **Book** can be bound to two actors - **Librarian** as well as **Student.**

- The actor can be a human, device or other system. For example - in a Company Information System, an Employee is an actor.
- An actor has **well defined purpose**. That means the actor helps to represent particular behavior of the system. For example the **Passenger** will interact with the **Reservation System** in order to make the reservations or to cancel the reservation.
- Actors will also help to represent the identity of different objects and the scope of these objects. The actors can be directly or indirectly connected to the system.

15.2.2 Use Cases

- **Use case** specifies the particular **functionality** of the system that can be obtained by interacting with the actors. For example - Withdrawal of money from ATM system, Issueing book to the student, reserving a seat, purchasing items and so on.
- Each use case includes one or more actors or the system itself.
- Use case includes a sequence of messages among the system and the actors. For example - For Course Reservation System the use case for Reservation of Course will include following sequence of messages.
  - Student selects the option "Reservation".
  - Student scrolls through the available courses and then selects the desired course.
  - Student Fills up the Form.
  - Student Submits the Form.
- In use cases, there can be fixed sequence of messages or the messages might have variations. For example depending upon the availability of course the student either might select or does not select the course.
- In use cases the error conditions can also be represented.
- Use case represents the behavior of the system using Main Flow, variations in normal flow, exceptions and error conditions and so on.
- The use cases can be written using following template -

Use Case template	
Use Case	Write the name of the Scenario.
Actor	Specify the role of the entity who interacts with the system.
Summary	Specify the purpose or goal of the system.
Preconditions	This is the condition which has to be satisfied before the use case starts.
Description	Sequence of steps that describe the main scenario of the use case.

- A scenario is written [illegible]

Student logs in.
System validates the student.
System displays the Home page for Course Reservation System.
Student scrolls through the available course.
Student selects the desired available course.
The system displays the form to be filled up.
The student fills up the form and submits it.
The system validates the eligibility of form.
System displays the form acceptance message.
The student receives the eligibility message and proceeds for payment.
Student selects the mode of payment.
Student fills up the required payment details.
Student confirms the reservation.
System validates the payment.
System generates the receipt.
Student selects the print receipt option.
Student gets the printout of the receipt.
Student logs out.

Fig. 15.3.1 Scenario for online course reservation system

- Normally the interaction between the objects is written at high level. At later stage of development the text message can be written in more detail.
- In the scenario, the messages that are passed between the interacting objects are written. Each message transmit the information from one object to another.
- While writing the scenario - first identify the objects that interact with each other. Then determine the sender and receiver of each message. And finally decide the sequence of messages. Finally the internal activities can be added due to which the scenarios can be transformed into the implementation code.

15.3.2 Sequence Diagram

- The scenarios are simple to write but they does not clearly specify the sender and receiver of each message.

- A separate sequence diagram can be prepared to represent the exceptional conditions of the use cases.
- The sequence diagram is normally created to cover the basic behavior of the system.

15.3.3 Guidelines for Sequence Models

Sequence models are used to represent the informal themes of use cases. These can be modeled to add up the details of the system.

Normally there are two kinds of sequence models - **scenarios** and **sequence diagram**. The scenarios are the textual collection of events and sequence diagram represents the sequence of messages graphically. Following is a guideline used while modeling the sequence models -

1. **Write one scenario for each one use case :** The scenario contains the logical sequence of operations performed by the system and actors. The simple mainline actions can be enlisted in the scenarios. If there are some alternate mainline actions then those can be written as separate scenarios.
2. **Abstract the scenarios into the sequence diagram :** Using the actions enlisted in the scenarios the sequence diagram can be created. In the sequence diagram the contribution of each actor can be clearly shown.
3. **Complex interactions into smaller one :** If there are large number of interactions in the system, then those can be separated in logically and can be modeled as a separate sequence diagrams.
4. **Create sequence diagram for error condition :** The system response to error conditions can be separately shown by creating a sequence diagram. For example -

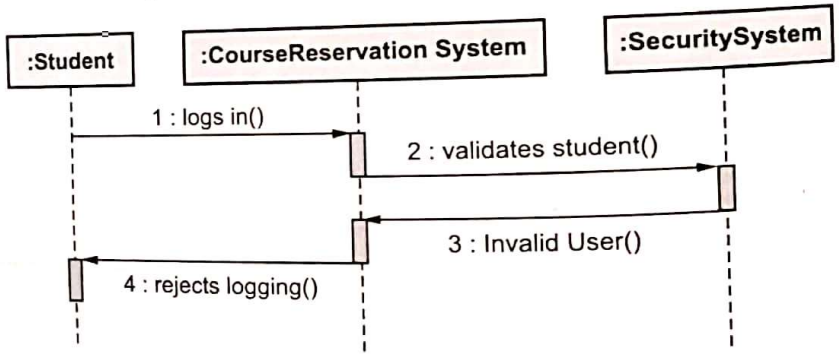
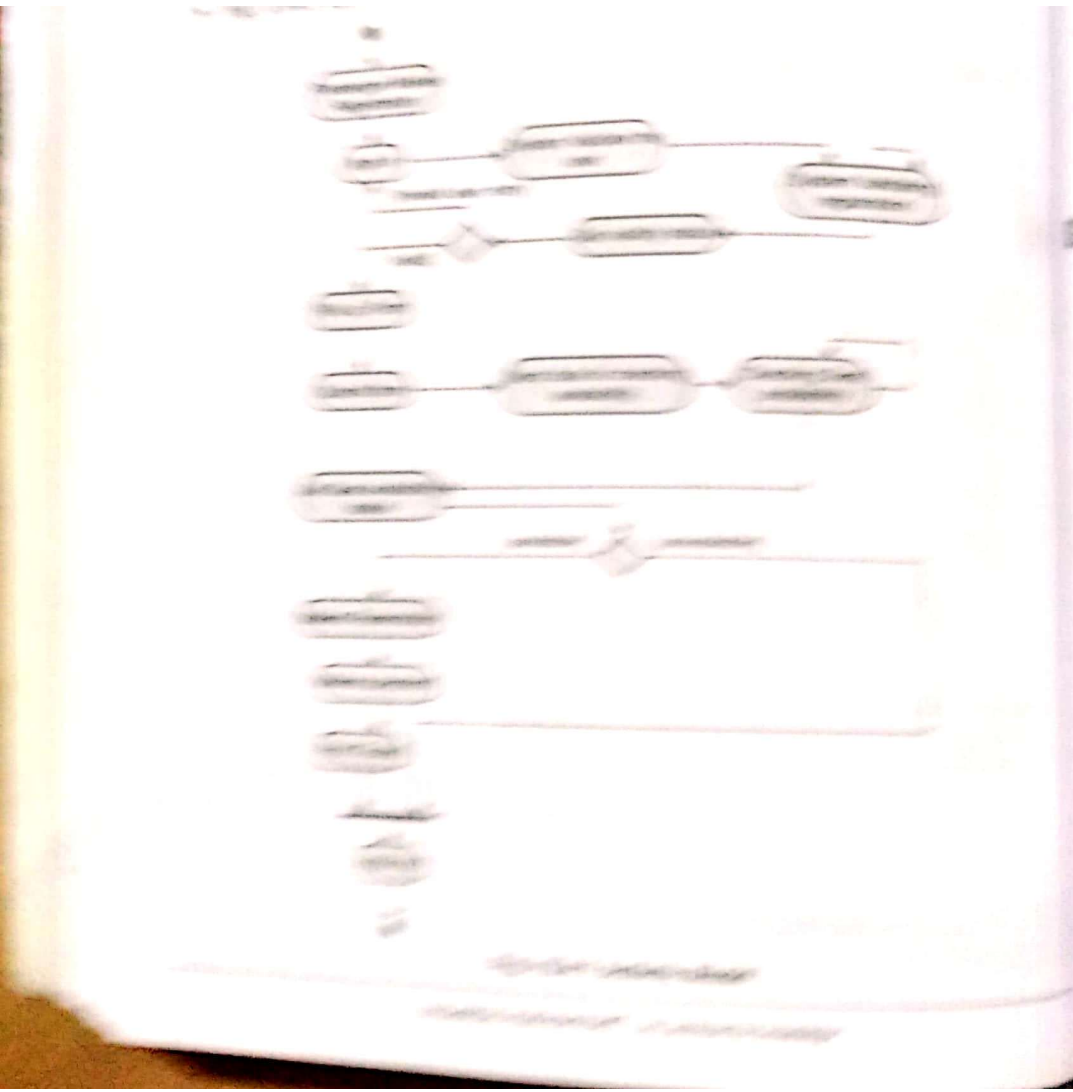


Fig. 15.3.3 Sequence diagram to model the invalid login



15.4 Activity Model

GTU : Winter-12, Summer-13, Marks 7

- An activity diagram is just similar to the flow chart in which flow of control from activity to activity is shown. Using algorithms or workflow the activity diagram can be drawn.
- In activity diagram the focus is on operations instead of objects.
- Activity diagrams are drawn during the early stages of designing algorithms or workflow.
- Fig. 15.4.1 represents the activity diagram for **Online reservation system**.

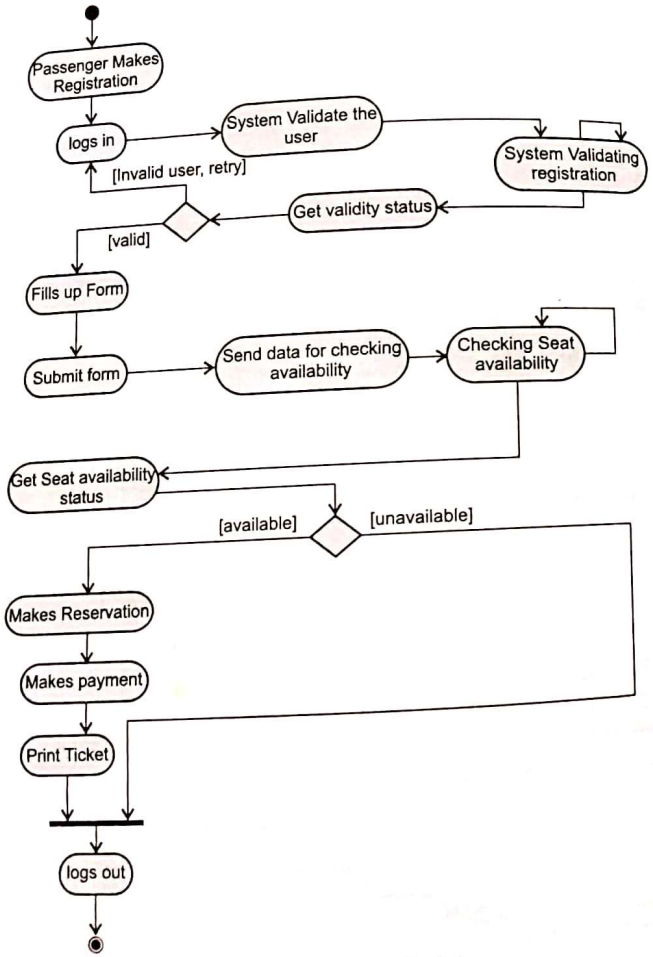


Fig. 15.4.1 Activity Model

15.4.1 Activities

- The activities are nothing but the operations. These operations can be identified from the state diagram.
- The activity diagram shows the complex processes that occur in specific sequence. The constraints imposed on these processes are also illustrated in activity diagram.
- Normally the activities complete their execution and terminate by themselves. But there are some activities that continue their execution until some external event interrupt them.
- When one activity finishes another activity starts. The two activities are connected by the **transitions**. These transitions are unlabelled.
- One complex activity can be decomposed into finer activities. For example the **Makes Reservation** this activity can be composed into smaller activities such as **Get Reservation Status, Scrolls through Status** and **Click Confirm button**.

15.4.2 Branches

- In activity diagram the branching is used to specify the alternate path based on some boolean expression.
- Each arrow of the alternative path is labeled by some condition. This condition is specified within the square bracket. For example In Fig. 15.4.2 the conditions **Not OK** and **OK** are specified within the square brackets.

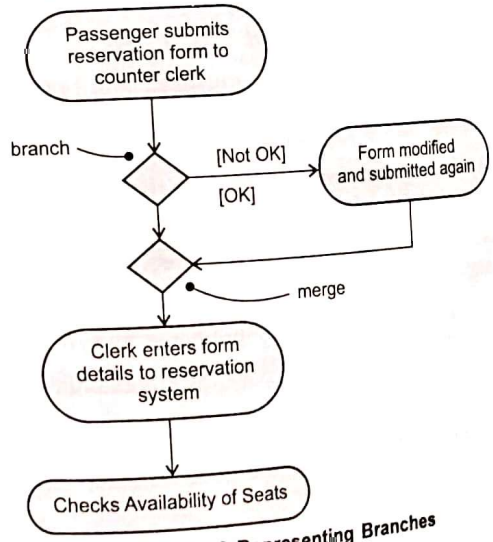


Fig. 15.4.2 Representing Branches

- Graphically branch is specified as **diamond**.
- The branch generally have one incoming and two or more outgoing flows and on each outgoing path the conditions are specified.
- When two paths merge together, they merge in **diamond**.

15.4.3 Initiation and Termination

Initial State : In activity diagram, the initial state is represented by a solid circle. From this circle the transition to the first activity starts. Thus the initial state represents that all the activities start their execution from this point.

Termination : In activity diagram, the termination is represented by **bull's eye** i.e. a solid circle surrounded by a hollow circle. This state represents the termination of all the activities at this point. Fig. 15.4.3 represents these states.

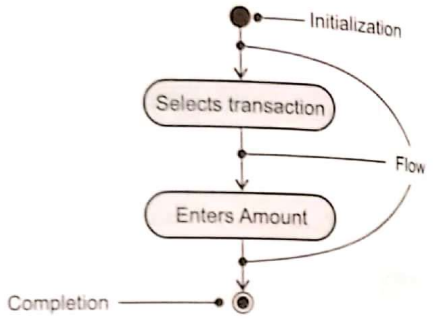


Fig. 15.4.3 Initiation and Termination

15.4.4 Concurrent Activities

- There are two types of activities - sequential activities and concurrent activities.
- The activities that can be performed one after the other are called sequential activities.
- The computer system can perform more than one activities at a time. The activities that can be performed at the same time are called concurrent activities.
- The concurrent activities can split in several activities or can join and merge in some action.
- In activity diagram the concurrent activities are represented with the help of **forking and joining**.
- The fork and join in activity diagram is represented by **synchronization bars**.
- The synchronization bars are represented by thick horizontal or vertical bars.
- The **fork** represents the separation of two control flows whereas **join** represents joining of two control flows.

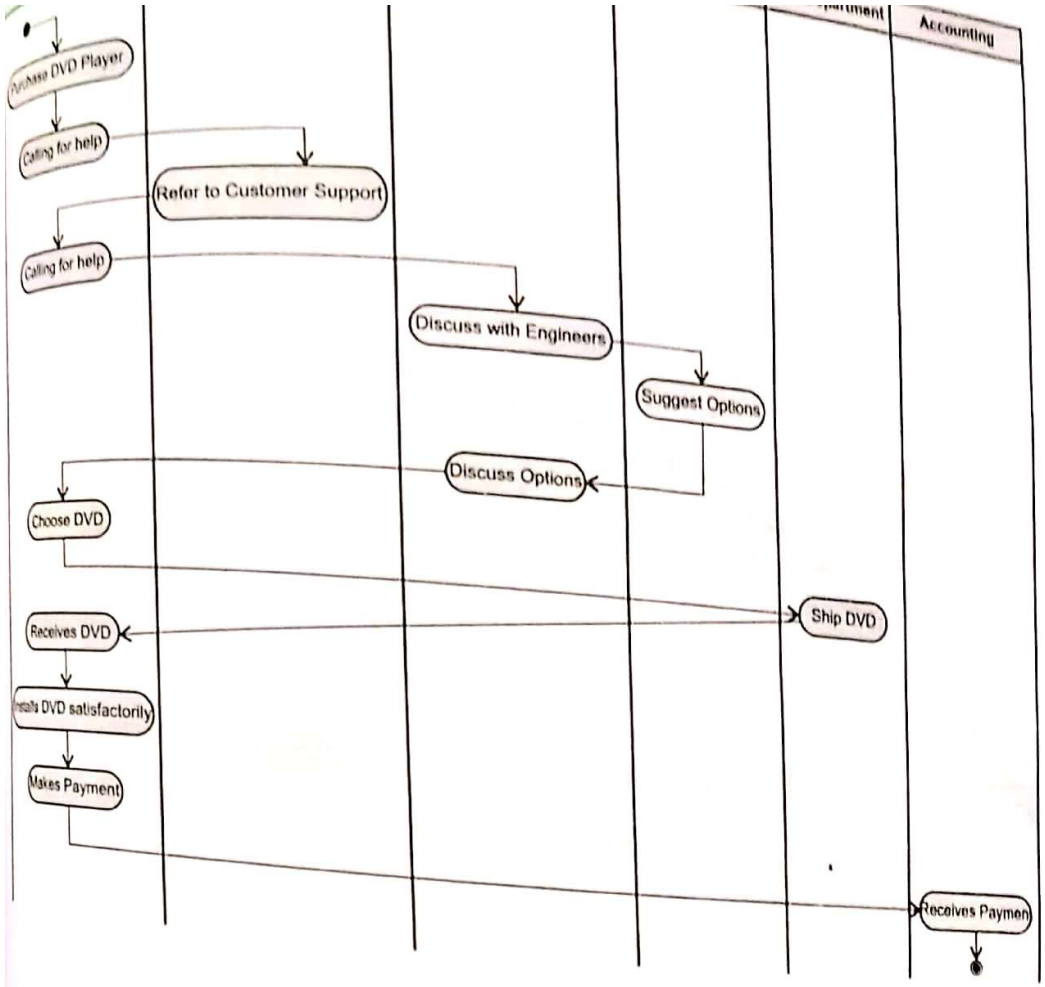
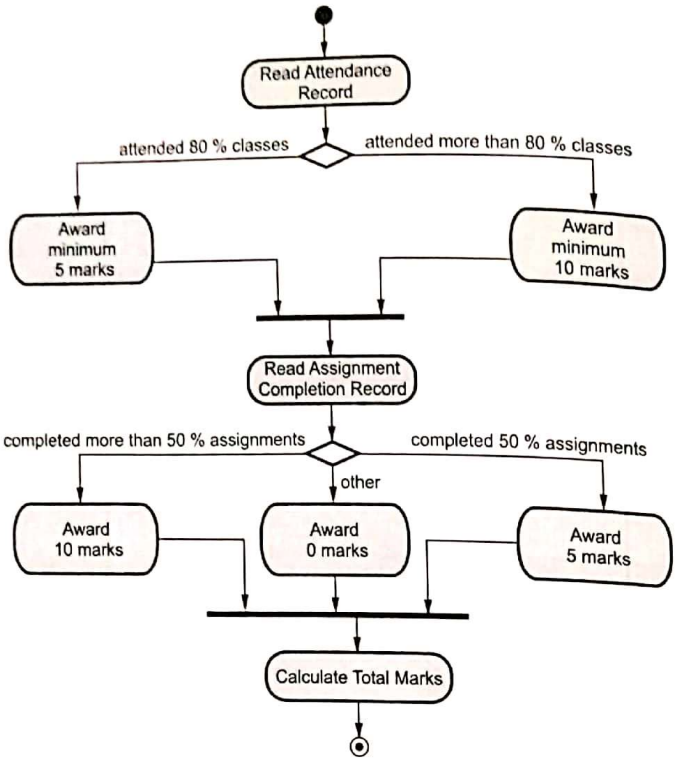


Fig. 15.4.5 Swimlane diagram

Example 15.4.2 *Prepare an activity diagram for awarding marks to regular students. If the student has attended 80 % classes, he is awarded minimum 5 marks. If the student has attended more than 80 % classes, he is awarded minimum 10 marks. The students who have completed assignments are given 10 marks. Those who have completed 50 % are given 5 marks and rests are given 0 marks.* **GTU : Summer-13, Marks 7**

Solution :



15.5 Use Case Relationships

GTU : May-12, Winter-12, Marks 7

As we know, the use cases are designed in order to expose the functionalities of the system. The complex use cases can be built using various relationships such as include, extend and generalization.

15.5.1 Include Relationships

- The include relationship is used to show that the base case is obtained by the behavior of other use cases.
- Using the stereotype **<<include>>** the include relationship can be represented.
- When there are multiple steps to carry out a single task then that task is divided into the sub-functions and these sub-functions can be denoted by the use cases. It is denoted as -

<<include>>

Example -

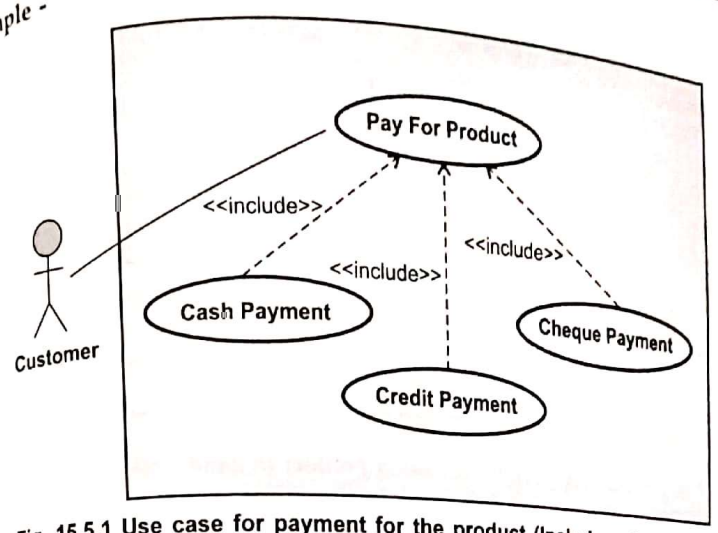


Fig. 15.5.1 Use case for payment for the product (Include- relationship)

- While designing the use case diagrams, using the **include** relationship, much fine details of the behavior must not be drawn.
- The factoring of use cases into more detailed use cases must be done to some appropriate level. That means each individual sub-use case must represent at least one functionality.

15.5.2 Extend Relationship

- The extend relationship is used when an extended use case is connected to the **base use case.**
- The extend relationship adds incremental behavior to use case. It represents frequent situations in which some initial conditions occurs and when new features are added in the module.
- The extension relationship is often a fragment that means it cannot occur alone; rather it will occur with the base use case.
- It can be denoted as -

<<extend>>

- The base use case can specify an insert location within the base class. These extension are called extension points. In most of the cases, an extend behavior occurs only when condition is true. Following Fig. 15.5.2 represents the extend relationship -

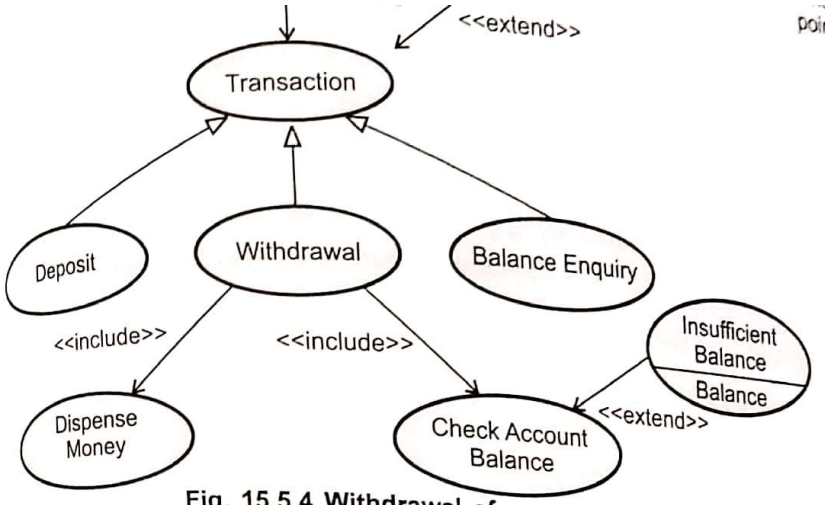


Fig. 15.5.4 Withdrawal of money from ATM

15.5.5 Guideline for Use Case Relationship

Following are some guidelines that can be used while designing the use case relationships -

- **Generalization Relationship :** If the use case has several variations then make the most general use case as parent or base use case and the specialized use cases as the child use cases. Do not use the generalization relationship to share the behavior of particular use case. For this purpose the include relationship can be used.
- **Include Relationship :** In order to fragment the particular behavior make use of include relationship. Secondly, the name of the use case must be meaningful to the users. For example - **Validate process** can be the base use case which can be further elaborated using the include relationship.
- **Extend Relationship :** For representing the optional features of the modeling the extended use cases are used. The extended use case can also be used when the system might be deployed in different configurations.
- **Include Relationship Vs. Extend Relationship :** The include and extend relationships are used to represent the division of single functionality into smaller functionalities. But the **include relationship** is used to represent what are the necessary functionalities that need to include to accomplish the main functionality. On the other hand the **extend relationship** is used to represent the optional behavior of the main functionality.

15.5.6 Examples

Example 15.5.2 *Describe use case "Validate User" in modeling an ATM system.*

Solution : There are two actors for the Bank ATM system. The **Customer** and the Bank. There are two types of customers - current account holder and savings account holder.

Following are the steps by which the customer interacts with the ATM system -

1. Customer inserts the card.
2. The ATM system validates the card.
3. If there is valid card entered by the customer then the ATM system requests the customer for entering PIN, The customer enters the PIN.
4. The PIN entered by the customer is validated.
5. If the valid PIN is entered then only the customer is prompted for performing the transactions.
6. Customer can select the desired transaction such as Withdrawal of money, Deposition of fund or simply enquires for the available balance amount.

The use case for Validate User is as given below -

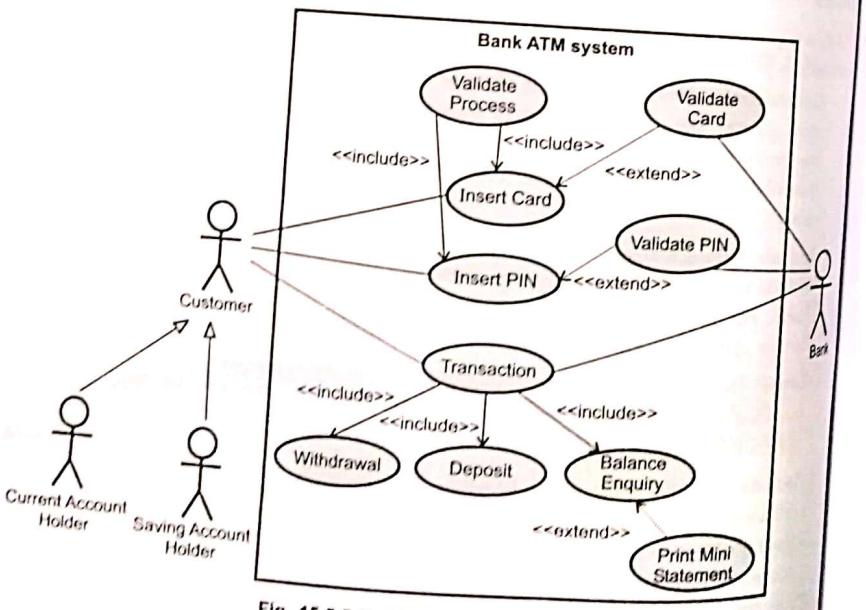


Fig. 15.5.5 Validate User use case

Use case Name : Validate Process

This use case is for validating the user interacting with the system. It consists of two use cases -

1. Insert Card
2. Enter PIN

For the card insertion the extended use case will be activated. By which the card is validated.

For the Enter PIN use case, the extended use case Validate PIN will be activated.

These validations can be performed by the Bank or by the authentication system which is associated with the bank.

Example 15.5.3 *Consider a software system like 'examination system'. Assume that there are use cases defined like 'view marks', 'input exam id','view practical marks', optionally send/forward my mark sheet as an Email'. Show how use case relationships like includes, generalizations and extends can be used to appropriately model above use cases and their relationships in context of use case diagram.* **GTU : Dec.-10, Marks 7**

Solution :

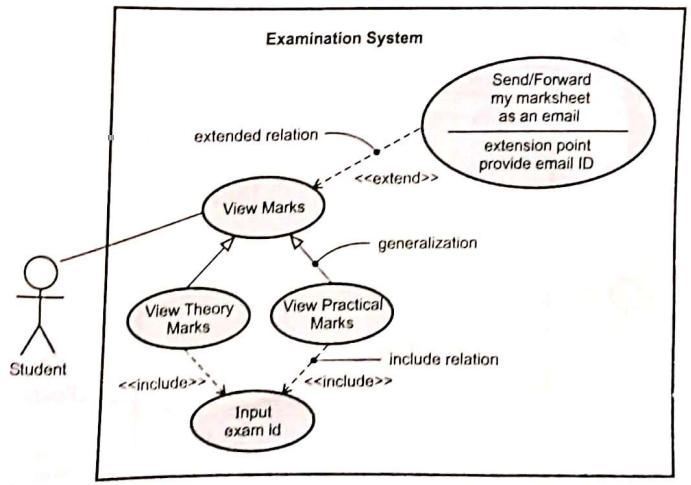


Fig. 15.5.6 Use case diagram for examination system

Example 15.5.4 *Give use case diagram for hospital management system.*

GTU : May-14, Marks 7

Solution :

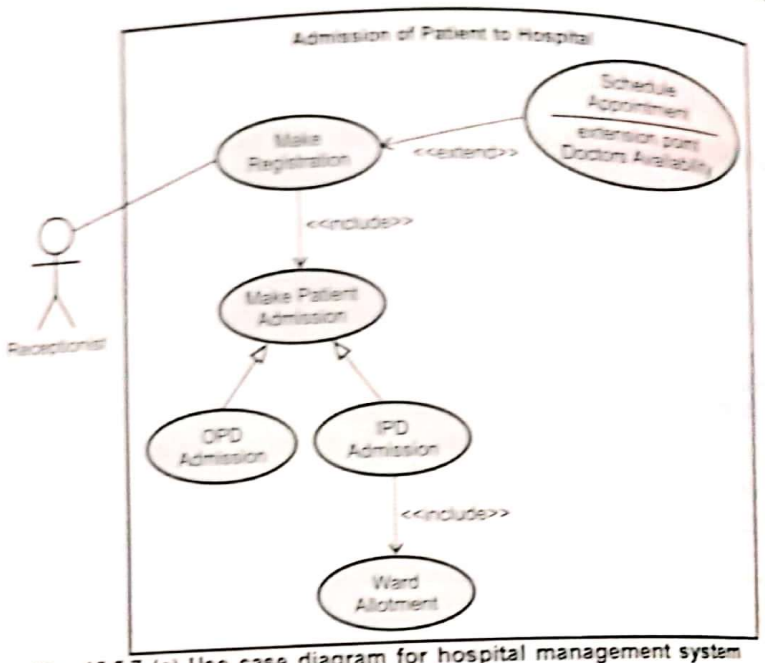


Fig. 15.5.7 (a) Use case diagram for hospital management system

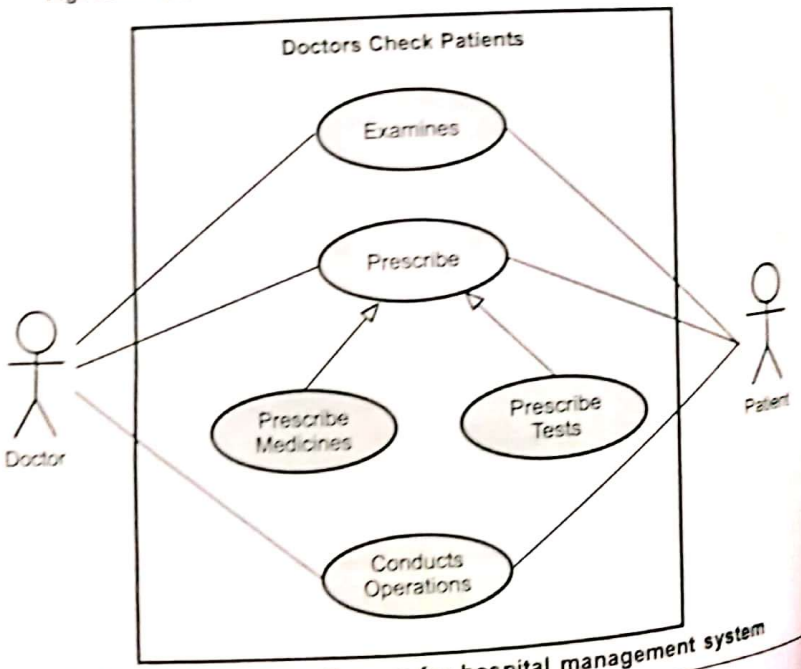


Fig. 15.5.7 (b) Use case diagram for hospital management system

In the credit card validation system, there are various actors that surrounds the system. The Customer can be of Individual customer and Corporate Customer. There two more customers that are included in this transactions and those are Retail Shopkeeper and Card authorization system. Various functionalities involved are making card transaction, processing of bill, reconciling of transactions and customer account management.

Example 15.5.6 *Consider an automated soda machine that gives cool drinks. Draw use case model of the soda machine.*

Solution :

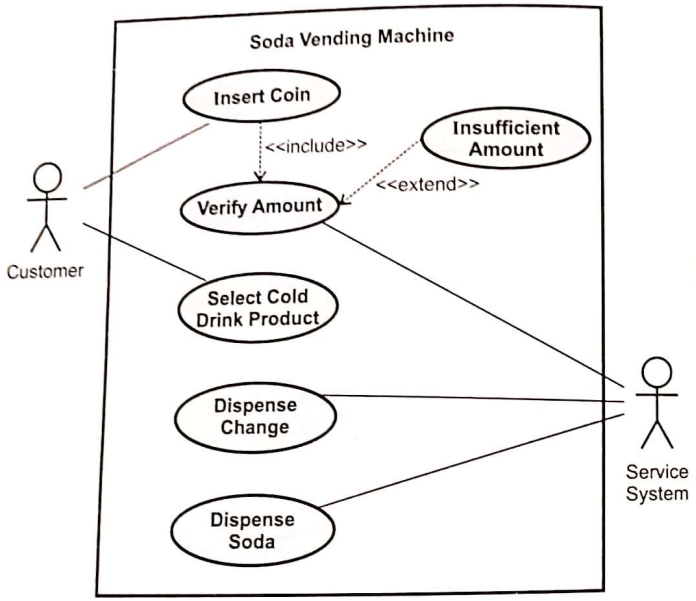


Fig. 15.5.9 Use case model for automated soda machine

Example 15.5.7 *Draw use case diagram to model the behavior of a cellular phone. Explain briefly.* **GTU : Dec.-11, Marks 7**

Solution :

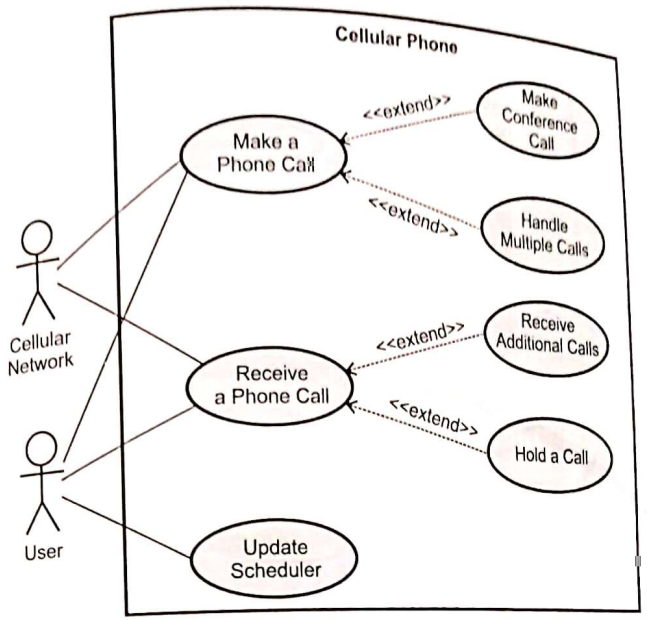


Fig. 15.5.10 Use case model for cellular phone

In above use case diagram, there are two important actors - user who operates the cell phone and the Cellular network. The three important use cases are -

Use case Name : Make a phone call

In this use case the activity of user makes a call is represented. While making a call user might make a conference call for some group communication.

Use case Name : Receive a phone call

The receiving call can be attended in normal scenario. But user can receive some additional calls or he/she can hold some receiving call. These two activities are represented using the extended relationship.

Use case Name : Update scheduler

The user can update his/her schedule using calendar.

Example 15.5.8 *What is the importance of use case diagram ? Explain the relationships between use cases with suitable example and proper UML notations. Draw use case diagram for 'online railway ticket reservation system'.* **GTU : May-12, Marks 7**

Solution : Importance of Use case diagram - Refer section 15.2.3.

Relationships in use cases : Refer section 15.5.

Use case diagram for online railway ticket reservation system

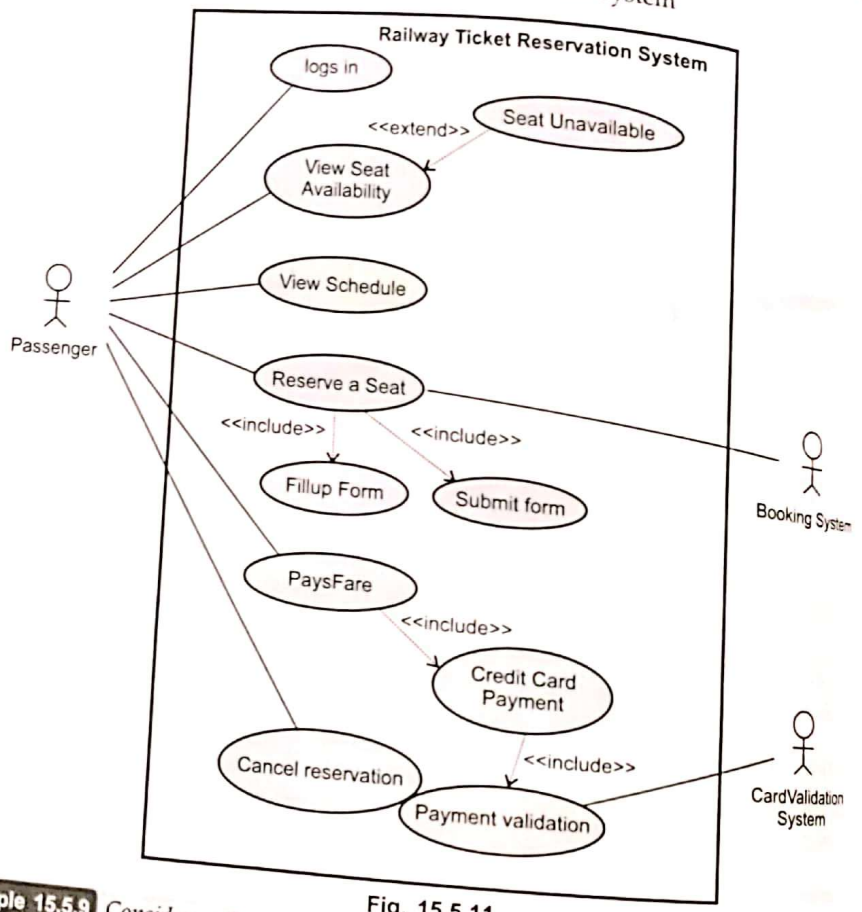


Fig. 15.5.11

Example 15.5.9 *Consider software that manages electronic music files. Prepare a use case diagram and include appropriate relationships for use cases.*

a. Play a song b. Play a library c. Randomized order d. Delete a song e. Destroy a song f. Add a song.

GTU : Winter-12, Marks 7

Solution : We will first elaborate each use case. Then the use case diagram can be prepared by adding the appropriate relationship.

a. Play a song : Add the desired song to the end of the play list.

b. Play a library : Add songs to the library for playing it.

c. Randomized order : Randomly reorder the songs in the play list.

15.6 Procedural Sequence Models

GTU : May-12, Marks 7

- The sequence diagram contains a collection of objects that are interacting with each other by exchanging the messages.
- The objects remain in active state even after sending the messages. These objects can receive the messages from other objects.
- In UML the **procedure calls** can be represented using the sequence models.

15.6.1 Sequence Diagrams with Passive Objects

- In sequence diagram all the objects are not active for all the time. When an object receives the message for invoking its procedure then that object becomes active and when the operation gets completed and control returns back to the caller then the object becomes passive.
- In UML, the period of time for an object's execution is shown as thin rectangle. This is called **focus of control** or **activation**.
- The entire period during which the object exists is called the **lifeline**.
- Following is a simple sequence diagram for online course registration system -

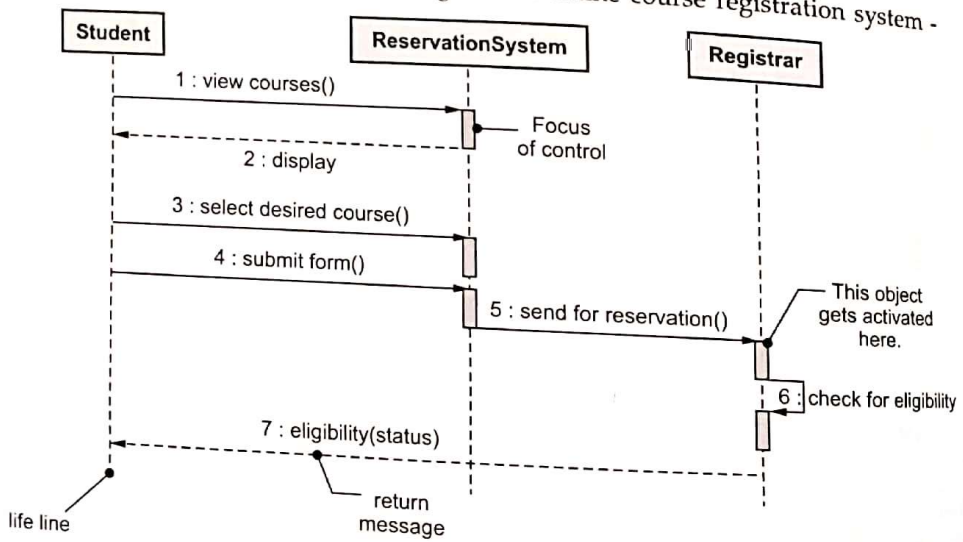


Fig. 15.6.1 Sequence diagram for online course reservation system

15.6.2 Sequence Diagram with Transient Objects

- A sequence diagram emphasizes on the time ordering of messages.
- The objects or the roles that take part in the interaction are placed at the top of the sequence diagram that are placed horizontally.

- The object that initiates the interaction is placed at the left and increasingly the subordinate objects or the roles are placed to the right.
- The messages that are passed among the objects are placed along the vertical axis, in order of increasing time from top to bottom.
- There is a lifeline in the sequence diagram. The lifeline is a vertical dashed line that represents the existence of the object over a period of time.
- Objects exists in the sequence diagram for the complete period of their interaction. Hence they are placed at the top of the diagram with a line that runs vertically from top to bottom.
- Using **create** message the objects are created and using **destroy** message the objects are destroyed.
- If the interaction represents the history of specific object then the objects are underlined. But usually we do not underline the objects.
- A tall thin rectangle represents the focus of control in the interaction diagram. It shows the period of time during which the object is performing its interaction. The top of the rectangle is aligned with the start of the action and the bottom of the rectangle is aligned with the end of the action.

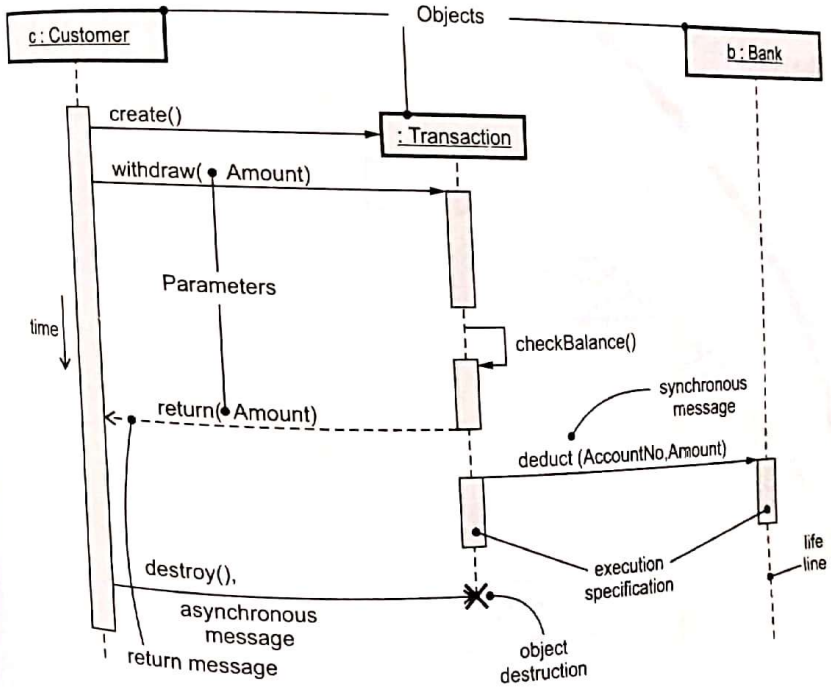


Fig. 15.6.2 Sequence diagram

Chapter Outline

Differentiate active, passive and transient object in sequence diagram. Draw sequence diagram for process interaction and case of ATM based banking system.

15.7 Special Constructs for Activity Models

For designing the large and complex systems the additional features of activity diagram are shown. In this section we will discuss some additional features of activity models.

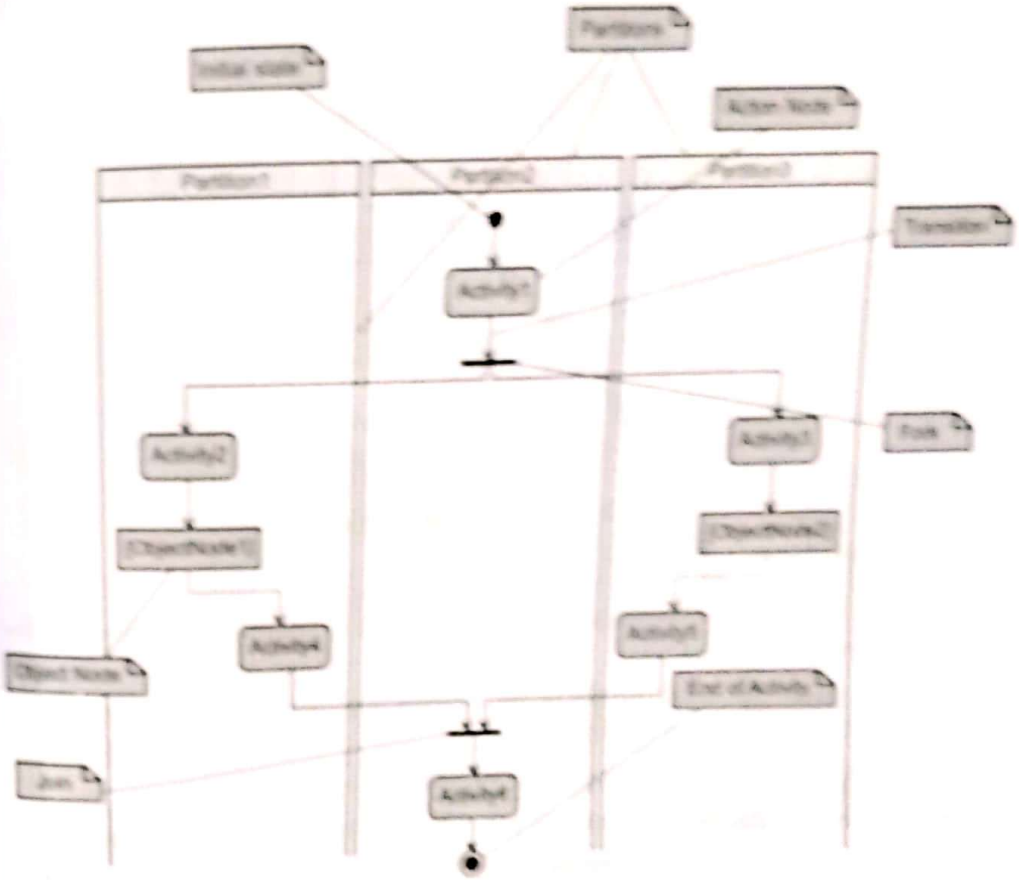


Fig. 15.7.1 Activity diagram notations

15.7.1 Sending and Receiving Signals

In activity diagram, the sending signal activity can be represented as convex pentagon. The receiving signal is represent by concave pentagon.

In the activity diagram, when previous activity gets completed then only the sending signal can be shown and when preceding activity completes the receipt construct waits until the signal is received. Only after receiving the signal the next activity starts.

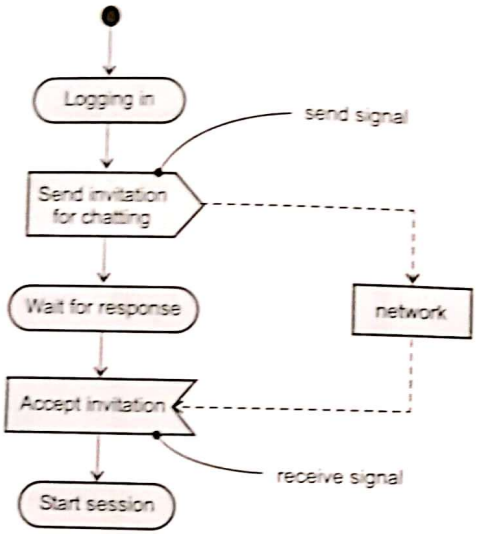


Fig. 15.7.2 Activity diagram with signals

15.7.2 Swimlanes

- Many times the activity diagram needs to partition in groups to represent the flow of business processes.
- Each group represents the particular business flow belonging to some category. Such a group is called **swimlane.**
- Graphically the swimlane is denoted by a vertical solid line.
- A swimlane specifies certain set of activities. Each swimlane has a unique name.
- Every activity belongs to only one swimlane but there can be transitions across the swimlanes.
- Following is an swimlane diagram for processing the **online order** in which three swimlanes are shown.

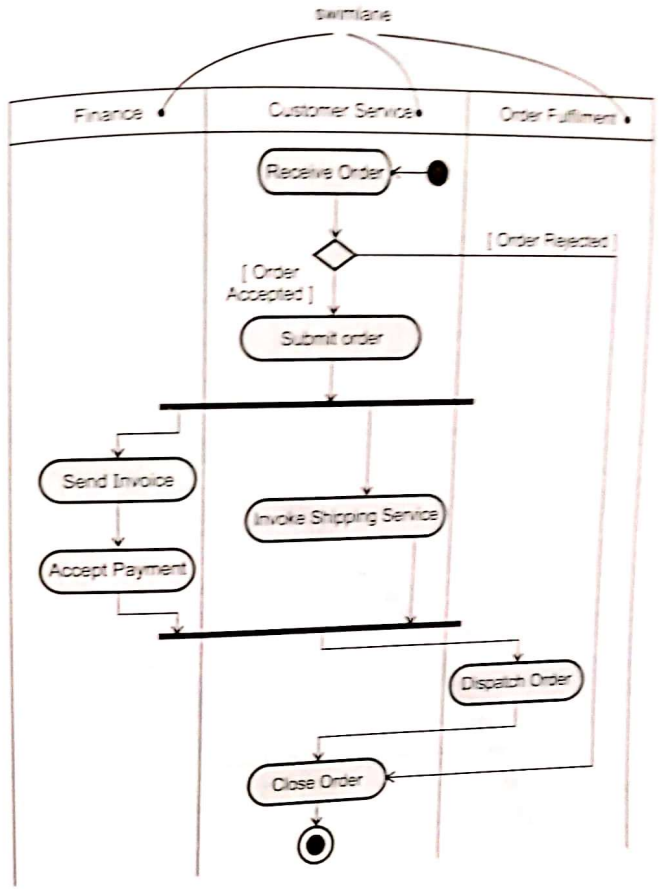


Fig. 15.7.3 Swimlane for order processing system

15.7.3 Object Flows

The objects may be involved in the activity diagram. These objects are associated with the control flow of the activity diagram. This is called **object flow.** The state of the object is specified for each of these objects.

Following is a simple activity diagram in which the object flow is represented for the objects of the classes **Order** and **Invoice**

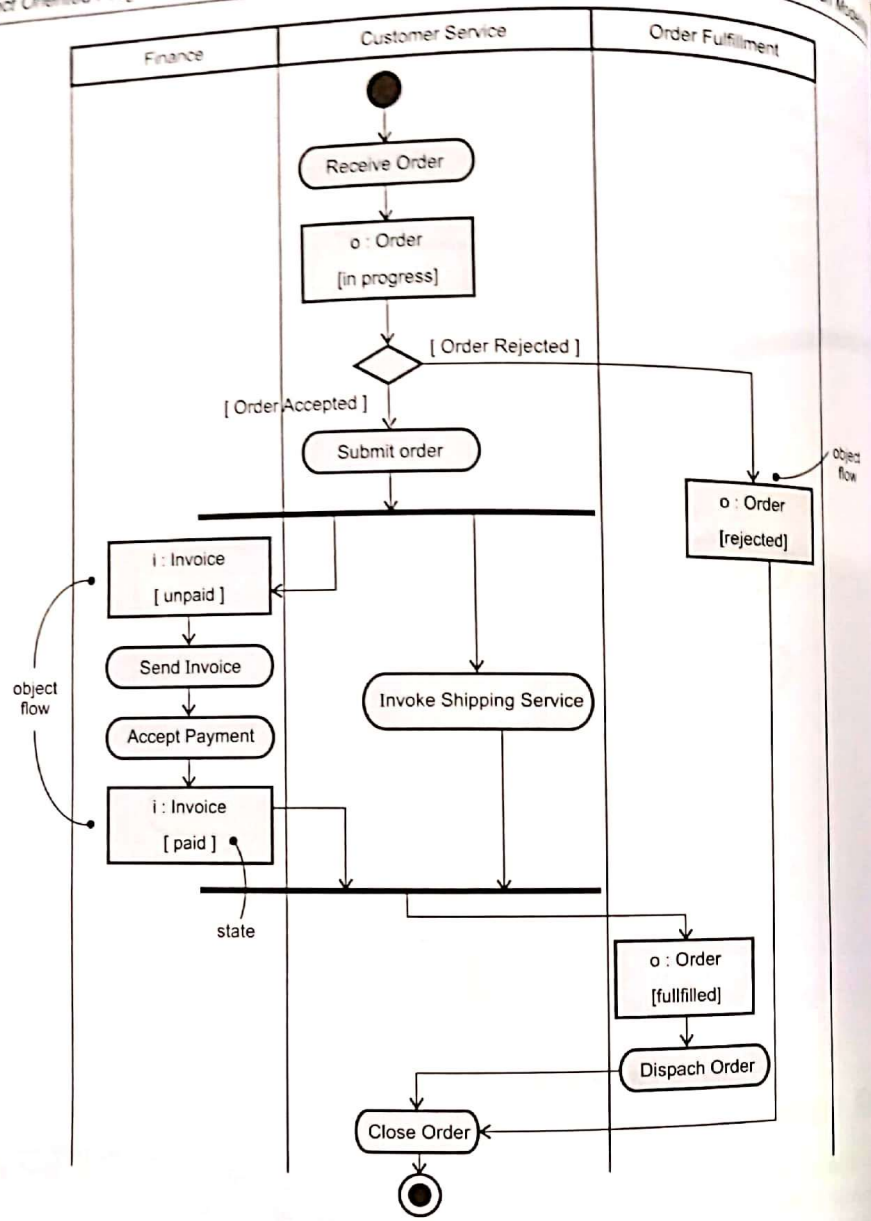


Fig. 15.7.4 Object flow

15.7.4 Synchronization Bars

The synchronization bars are the vertical or horizontal bars in the activity diagr... The fork and join is represented in activity diagram by synchronization bar. R... Fig. 15.4.4 from section 15.4.4.

15.7.5 Examples

Example 15.7.1 *Give activity diagram for hospital management system.*

Solution :

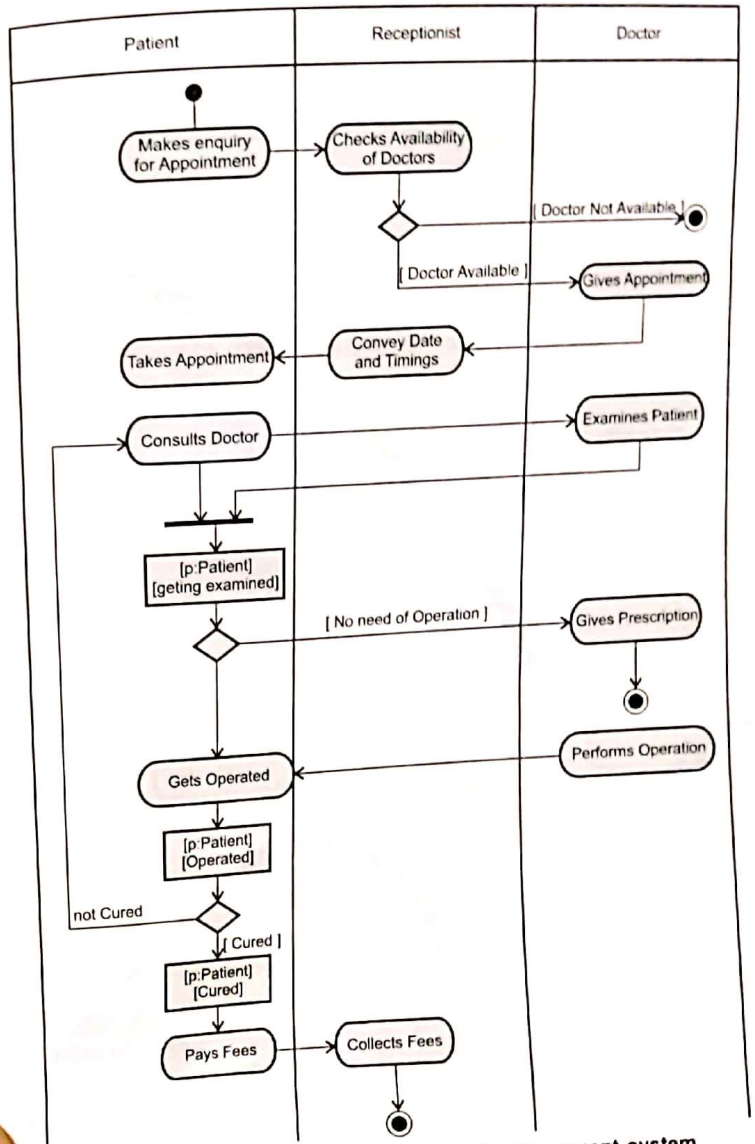


Fig. 15.7.5 Activity diagram for hospital management system

Example 15.7.2 *Draw activity diagram for withdrawal of money from ATM.*

GTU : Dec-10, Marks 7

Solution :

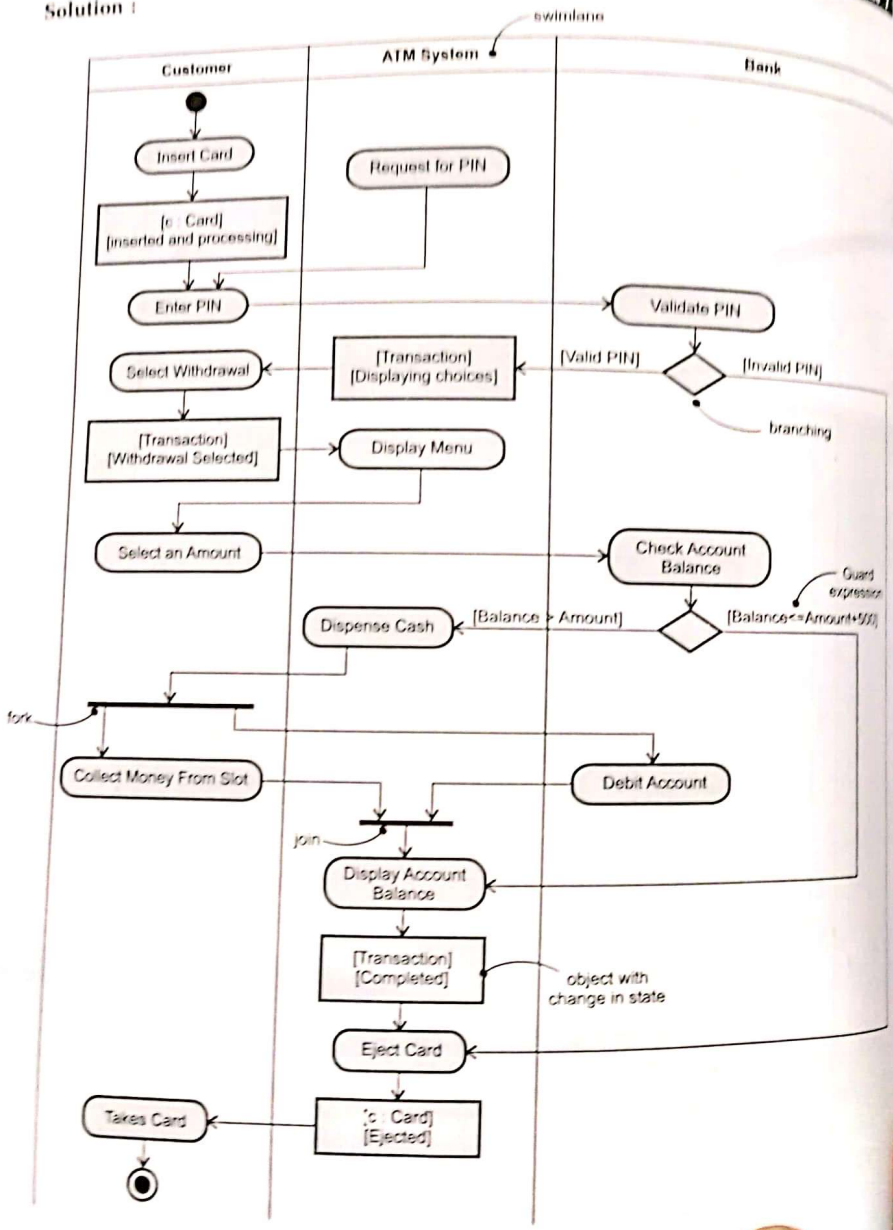


Fig. 15.7.6 Modeling workflow

The activity diagram given in Fig. 15.7.6, is for handling the ATM System. The customer **withdraws money** from ATM machine. This workflow is illustrated in this figure. There are two objects **Card** and **Transaction** that are shown explicitly which take part in the workflow. Note that the state of these objects gets changed due to the actions that are carried out during the processing. This workflow involves branching, fork and join. It is little bit complex workflow.

Review Question

1. *Explain the purpose of activity diagram? In which situation activity diagram is not necessary ? Explain the use of following concepts for activity diagram : synchronization bar, swimlane and sending receiving signals.*

GTU : May-12, Marks 7

15.8 University Questions with Answers

(Regulation 2008)

May 2012

Q.1 *Differentiate active, passive and transient object in sequence diagram. Draw sequence diagram for 'process transaction' use case of ATM based banking system.* **[Refer section 15.6]** [7]

Q.2 *Explain the purpose of activity diagram ? In which situation activity diagram is not necessary ? Explain the use of following concepts for activity diagram : synchronization bar, swimlane and sending-receiving signals.* **[Refer section 15.7]** [7]

Q.3 *What is the importance of use case diagram ? Explain the relationships between use cases with suitable example and proper UML notations. Draw use case diagram for 'online railway ticket reservation system'* **[Refer section 15.5]** [7]

Winter 2012

Q.4 *Write use case summaries for a vending machine.* **[Refer section 15.2]** [3]

Q.5 *A customer decides to upgrade her PC and purchase a DVD player. She begins by calling the sales department of the PC vendor and they tell her to talk to customer support. She then calls customer support and they put her on hold while talking to engineering. Finally the customer support tells the customer about several supported DVD options. The customer chooses a DVD and it is shipped by the mail department. The customer receives the DVD, installs it satisfactorily and then mails her payment to accounting.* **[Refer section 15.4]**

Construct an activity diagram for this process. Use swimlanes to show the various interactions. **[Refer section 15.4]** [7]

Q.6 *Consider software that manages electronic music files. Prepare a use case diagram and include appropriate relationships for use cases.*
a. Play a song b. Play a library c. Randomized order d. Delete a song e. Destroy a song f. Add a song. **[Refer section 15.5]** [7]

Summer 2013

Q.7 *Prepare an activity diagram for awarding marks to regular students. If the student has attended 80 % classes, he is awarded minimum 5 marks. If the student has attended more than 80 % classes, he is awarded minimum 10 marks. The students who have completed assignments are given 10 marks. Those who have completed 50 % are given 5 marks and rests are given 0 marks.* **[Refer section 15.4]** [7]

Q.8 *Prepare a use case description for issue a book from the library.* **[Refer section 15.2]** [7]

Q.9 *Prepare sequence diagram for booking a train ticket online. Also prepare sequence diagram for booking a train ticket online that fails.* **[Refer section 15.3]** [7]

❑❑

Object Oriented Programming using Java - Laboratory

Experiment 1	Write a program to convert rupees to dollar. 60 rupees =1 dollar	L - 3
Experiment 2	Write a program to calculate percentage marks of the student if marks of 6 subjects are given	L - 4
Experiment 3	Write a program to enter numbers and perform mathematical operations on them.	L - 4
Experiment 4	Write a program to find length of string and print second half of the string	L - 6
Experiment 5	Write a program to accept a line and check how many consonants and vowels are there in a line	L - 7
Experiment 6	Write a program to count number of words that start with capital letters	L - 8
Experiment 7	Write a program to find that given number or string is palindrome or not	L - 9
Experiment 8	Create a class which asks the user to enter sentence, and it should display count of each vowel type in the sentence. The program should continue till user enters a word "quit". Display the total count of each vowel for all sentences	L - 10
Experiment 9	Write an interactive program to print a string entered in a pyramid form. For instance, the string "stream" has to be displayed as follows	L - 12
Experiment 10	Write an interactive program to print a diamond shape. For example if user enters the number 3, the diamond will be printed	L - 13
Experiment 11	Create a class called Student. Write a student manager Program to manipulate the Student information from files by using FileInputStream and FileOutputStream	L - 14
Experiment 12	Refine a class called Student. Write a student manager Program to manipulate the Student information from files by using BufferedReader and BufferedWriter	L - 17
Experiment 13	Refine a class called Student. Write a student manager Program to manipulate the Student information from files by using DataInputStream and DataOutputStream	L - 19
Experiment 14	Prepare a class diagram for given group of classes using multiplicity, generalization, association concepts. And add at least 5-7 attributes and 3-5 operations for particular class page, shape, point, Line, Arc, Ellipse,Rectangle, Circle	L - 20
Experiment 15	Prepare a class diagram for given group of classes using multiplicity, generalization, association concepts. And add at least 5-7 attributes and 3-5 operations for particular class City, Airport, Airline, Pilot, Flight, Plane, Seat, Manager	L - 20

Experiment 1 *Write a program to convert rupees to dollar. 60 rupees =1 dollar*

Solution :

```
import java.util.Scanner;
class RupeesToDollar
{
  public static void main(String[] args)
  {
    Scanner in=new Scanner(System.in);
    System.out.println("Enter rupees");
    double rupees=in.nextDouble();//reading double value from console
    if (rupees>1)
    {
              double dollar = rupees/60.0; //conversion to dollar
            System.out.println("The " + rupees + " Rs. = " + dollar + " dollars");
    }
    else
              System.out.println("Try again");
  }
}
```

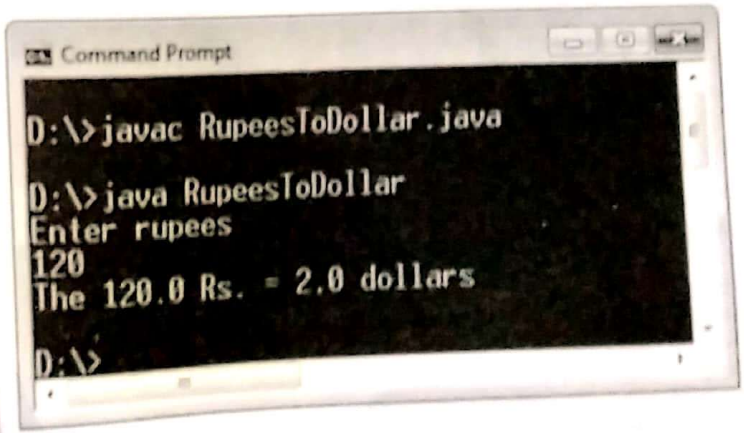


Fig. 1

SHORT QUESTIONS AND ANSWERS

Chapter 1 : Basics of Java

Q.1 List out any two features of Java.

Ans. : 1. Java can be compiled and interpreted.

2. Java is a platform independent and portable programming language

3. Java is an object oriented programming language.

Q.2 Why Java is called object oriented programming language ?

Ans. : In Java everything is an object. The code and data lies in class in Java. Hence it s called object oriented programming language.

Q.3 Why Java is called dynamic and extensible language ?

Ans. : This language is capable of dynamically linking new class libraries, methods and bjects. Java also supports the functions written in C and C++. These functions are alled native methods. Hence Java is called dynamic and extensible language.

Q.4 State two differences between C++ and Java.

Ans. :

No.	C++	Java
1.	The C++ is a language that needs to be compiled.	The Java is a language that gets interpreted and compiled.
2.	C++ is platform dependent.	Java is a platform independent.
3.	C++ code cannot be embedded in any scripting language.	Java code can be embedded within a scripting language by means of Applet programming.

5 What is byte code ?

s. : Byte code is an intermediate form of Java programs. Byte code consists of an timized set of instructions that are not specific to the processor.We get bytecode after mpiling the source java program using the compiler javac.

What is JVM ?

s. : Java Virtual Machine is a set of software and program components which ecutes the byte code.

What is JDK ?

s. : Java Development Kit is nothing but a collection of tools that are used for velopment and execution of Java programs.

Q.8 What is the use of appletviewer ?

Ans. : Appletviewer is used for executing Java applets.

Q.9 What is javac ?

Ans. : The Java compiler which translates the source code to the bytecode form and stores it in a separate class file.

Q.10 What is Java Token ?

Ans. : The smallest individual and logical unit of Java statements are called tokens. Five types of tokens are called - 1. Reserved keyword 2. Identifier 3. Literals 4. Operators 5. Separators

Q.11 What are literals ?

Ans. : Literals are the kind of variables that store the sequence of characters for representing the constant values. Five types of literals are - Integer literal, floating point literal, Boolean literal, Character literal, string literal.

Q.12 What is byte data type ?

Ans. : This is in fact smallest integer type of data type. Its width is of 8-bits with the range – 128 to 127.

The variable can be declared as byte type as

```
byte i,j;
```

Q.13 What is dynamic initialization in Java ?

Ans. : In Java we can declare a variable at any place before it is used. This facility is called dynamic initialization.

Q.14 What is precedence relation ?

Ans. : Precedence relation specifies which operation must be done first during the expression evaluation.

Q.15 What is the difference between while and do...while statement ?

Ans. : The difference between while and do...while is that do.. while statement must be executed at least once. That means the statement inside the do... while body gets executed first and then while condition is checked for next execution of the statement.

Q.16 Write a Java program to display the even numbers between 1 to 10 ?

Ans. :

```
System.out.println(" "+i);
}
```

Chapter 2 : Arrays and String

Q.1 Define the term array.

Ans. : Array is a collection of similar type of elements.

Q.2 Explain how to declare an array ?

Ans. : Syntax of declaring an array is

```
Data_type array_name[]
nt a[];
```

Then the memory for the array is created using the keyword **new**.For example

```
=new int[10];
```

Q.3 What is ragged array ?

Ans. : Ragged array is more than one dimensional array in which each dimension has different size. For example {1,2,3,4},{5,6},{7,8,9}} is ragged two dimensional array.

Q.4 How would you define string in Java ?

Ans. : In java **String** is used to define the string object. For example

```
String s="Hello, How are you?"
```

Q.5 Specify at least two methods of initialization of string.

Ans. :

Method 1:

```
char str[]={'P','R','O','G','R','A','M'};
String s=new String(str);
```

Method 2:

```
String s="PROGRAM";
```

Q.6 Which method is used to extract the character from the string object ? Give example.

Ans. : The method charAt(index) is used to extract the character from the String object. For example

```
String s="Java";
char ch;
ch=s.charAt(0);// character J will be retrieved.
```

Q.7 What is String Buffer class ?

Ans. : The String Buffer class is used declare string. Using StringBuffer class we can create a buffer of characters.

Q.8 State the difference between String and String Buffer class.

Ans. : String buffer class is similar to String class. But using StringBuffer class we can insert some components to existing string or modify the existing string but in case of String class once the string is defined then it remains fixed.

Q.9 What is the difference between length and capacity function of String Buffer class ?

Ans. : The length function returns the number of characters in the string and the capacity function returns the number of characters in the string +16 additional characters.

Q.10 Which method is used to get the specific character of a string of a string buffer class ?

Ans. : The charAt(int index) function returns the specific character specified by the index. For example

```
StringBuffer str=new StringBuffer("friend");
str.charAt(3); //This will return the character e
```

Q.11 What is command line argument ?

Ans. : The command line argument is an argument that can be provided from command line to program. For example

```
C:\>java test 10 20
```

Here 10 and 20 are the command line arguments.

These command line arguments can be passed to the Java program in args of main function

```
public static void main(String args[])
```

Q.12 What is wrapper classes ?

Ans. : Wrapper classes are those classes that allow primitive data types to be accessed as objects. For example

Primitive Data Type	Wrapper Class
Boolean	java.lang.Boolean
Byte	java.lang.Byte

Q.13 What is the use of toString() method in Java ?

Ans. : The toString() method is used to convert the numerical value to string. For example

String str=Integer.toString(int_value); //The value of variable int_value can be converted to string

Q.14 Specify any two characteristics of wrapper classes.

Ans. :

1. The wrapper classes do not contain the constructors
2. The methods of the wrapper classes are static.
3. After assigning the values to wrapper class we can not change them.

Q.15 Write declaration to convert the value "programmer" in the string variable 'convert' to "programming".

Ans. :

```
public class StringBuffDemo
{
 public static void main(String args[])
 {
  StringBuffer str1=new StringBuffer("Programmer");
  System.out.print("\nThe word is: "+str1);
  str1.delete(8,10);
  str1.append("ing");
  System.out.print("\nThe word is changed to: "+str1);
 }
}
```

Output

```
The word is: Programmer
The word is changed to: Programming
```

Q.16 Enlist various methods used by wrapper class.

Ans. : Various methods used by wrapper class are - floatValue(), longValue(), doubleValue(), toString()

Q.17 Write a program to convert the strings "10" and "20" to corresponding values and perform their addition.

Ans. :

```
class Conversion
{
    public static void main(String[] args)
    {
        String str1="10";
        String str2="20";
        int result;
        result=Integer.parseInt(str1)+Integer.parseInt(str2);
        System.out.println("The addition = "+result);
    }
}
```

Output

```
The addition = 30
```

Chapter 3 : Classes, Objects and Methods

Q.1 Define the term class in Java.

Ans. : The class in Java is a collection of data and functions that manipulate the data. The data components of class are called data fields while the function components are called member functions or methods.

Q.2 Define the term object in Java.

Ans. : An object is an instance of a class. The objects represent real world entity. The objects provide the practical basis for using the class.

Q.3 Why are classes important in OO technology ?

Ans. : The reasons are as follows -

1. Classes bind together the relative data and methods in a cohesive unit. Due to this arrangement, only certain methods are allowed to access the corresponding data.
2. If any modification is needed, then the class can be viewed as one module and the changes made in one class does not spoil rest of the code.
3. Finding error from such a source code becomes simple.

Q.4 What is the difference between object and class ?

Ans. : Following are some differences between the class and the object -

What is the difference between structure and class ?

Ans. :

No.	Structure	Class
1	By default the members of structure are public.	By default the members of class are private.
2	The structure can not be inherited.	The class can be inherited.
3	The structures do not require constructors.	The classes require constructors for initializing the objects.
4	A structure contains only data members.	A class contains the data as well as the function members.

Q.6 Define encapsulation.

Ans. : Encapsulation means binding of data and method together in a single entity called class.

Q.7 How to define a class ? Give an example of class in Java.

Ans. :

```
class Customer
{
  int ID;
  String Name;    //Data Field
  Customer()  //Constructor
  {  }
  double withdraw_money() //method
   { ...}
}
```

Q.8 How would you declare an object of type animal named lion that takes a weight of 500 and length of 45 as parameters ?

Ans. :

```
public class Animal
{
 int weight,length;
 Animal(int wt,int len)
 {
     weight=wt;
     length=len;
 }
 void Show()
 {
     System.out.println("\n The weight of animal is: "+weight);
     System.out.println("\n The length of animal is: "+length);
 }
```

```
class AnimalMain
{
    public static void main(String args[])
    {
        Animal Lion=new Animal(500,45);
        Lion.Show();
    }
}
```

Output

```
The weight of animal is: 500
The length of animal is: 45
```

Q.9 What is meant by private access specifier ?

Ans. : The private access specifier allows classes, methods and data fields accessible only from within the own class. The functions outside the class cannot access the data members or the member functions.

Q.10 Which operator is used to create an object ?

Ans. : The new operator is used to create an object. For example -

Q.11 What is object reference ?

Ans. : Object reference is an information on how to find particular object. The object is a chunk of memory and the object reference is a way to get to that chunk of memory.

```
String str1=new String("Hello");// str1 is an object reference variable
```

Q.12 What is the purpose of constructor in Java ?

Ans. : The constructor is a specialized method used for initializing the object.

Q.13 Specify any two important characteristics of constructor.

Ans. : 1. The name of the constructor and the class name is the same.

2. The constructor is invoked when the object associated with the corresponding class is created.

3. The constructor should not have any return type.

Q.14 Why there is no destructor in Java ?

Ans. : The garbage collection facility in Java is available which is responsible for freeing up the memory allocated for the objects. Thus garbage collection takes care of automatic memory management. Hence there is no destructor in Java

Q.15 Specify one differentiating point between method and constructor.

Ans. : The name of the constructor and the belonging class name must be the same. The method can have any valid alphanumeric name.

Q.16 What is constructor overloading ?

Ans. : Multiple constructors with different signatures is called constructor overloading

Q.17 What is method overloading ?

Ans. : Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameters.

Q.18 What is recursion ?

Ans. : Recursion is a programming technique in which the method calls itself repeatedly. The most common example is computing factorial using recursion.

Q.19 What do you understand by the term recursive method ?

Ans. : When the parameter passed to a particular method allows to call itself, then that method is called recursive method.

Q.20 What is the difference between recursion and iteration ?

Ans. : Iteration is a process of executing certain set of instructions repeatedly without calling the self- function. Recursion is a process of executing certain set of instructions by calling the self-function.

Q.21 How to create an object using new operator ?

Ans. :

Syntax:

```
className objectName=new className()
```

Example

```
Test obj=new Test();
```

Q.22 Explain the meaning of the statement -"The object has physical reality".

Ans. : The statement means object occupies space in memory.

Q.23 What is the purpose of this keyword ?

Ans. : When a calling object wants to refer its own values then **this** reference is used. The keyword **this** is used for making the **this** reference. Using **this** reference we can refer to class's hidden data fields. For example

```
this.a=a;
```

Q.24 In Java _____ is always static method

Ans. :

```
main()
```

Q.25 What is the difference between static and non static variables ?

Ans. : A static variable is shared among all instances of class, whereas a non static variable (also called as instance variable) is specific to a single instance of that class.

Q.26 What is the purpose of finalization ?

Ans. : Finalization is the facility provided by the Java for the classes for cleaning up the native resources before the objects are garbage collected. The garbage collector is

unable to control the cleaning up of native resources which are used earlier. Then the responsibility of cleaning up those native allocations falls on the object's finalization code.

Thus the purpose of finalization is to clean up the native resources used earlier. The finalize() method must be run before invoking the garbage collector.

Q.27 What are access modifiers ?

Ans. : Access modifiers control access to data fields, methods and classes. There are three access modifiers used in Java - 1. Public 2.Private and 3. Protected.

Q.28 What is nested class ?

Ans. : When one class is defined inside another class then it is called nested classes. The scope of inner class is bounded by the scope of the outer class.

Q.29 How to define inner class ? Give its syntax.

Ans. :

```
Access_Modifier class outerClass
{
    //code
    Access_Modifier class InnerClass
    {
        //code
    }
}
```

Q.30 Enlist various types of inner classes

Ans. : 1. Static Member classes 2. Member Inner Classes

3. Local Inner Classes 4. Anonymous Inner Classes

Q.31 Mention any two constraints on abstract class

Ans. : 1. Abstract class can not be instantiated.

2. Abstract classes always act as a parent class.

3. The abstract keyword is used to define the abstract class.

Chapter 4 : Inheritance and Interfaces

Q.1 What is inheritance ?

Ans. : In Java, inheritance means derived class borrows some properties of base class. At the same time the derived class may have some additional properties.

Q.2 Explain the terms superclass and subclass.

Ans. : The class which is inherited is called superclass and the class that does the inheriting is called subclass. Thus subclass is a specialized version of superclass which uses some properties of superclass along with its own properties.

Q.4 Which keyword is used to invoke the constructor of superclass ?

Ans. : The keyword super() is used to invoke the constructor of superclass.

Q.5 Enlist various types of inheritance.

Ans. : Various types of inheritance are

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hybrid inheritance

Q.6 How to prevent inheritance ?

Ans. : If we declare particular class as final then no class can be derived from it. For example

```
final class Test
{
  ...
  ...
  ...
}
class Test1 extends Test
{
  ...
  ...
  ...
}
```

The above code will produce an error"cannot inherit from final Test".

Thus the use of keyword final for the class prevents the inheritance.

Q.6 In Java what is the use of interface ?

Ans. : In java, the interface is used to specify the behaviour of a group of classes. Using interfaces the concept of multiple inheritance can be achieved.

Q.7 What is method overriding ?

Ans. : Method overriding is a mechanism in which a subclass inherits the methods of superclass and sometimes the subclass modifies the implementation of a method defined in superclass.

Q.8 What is constructor chaining ?

Ans. : A constructor invokes its superclass's constructor explicitly and if such explicit call to superclass's constructor is not given then compiler makes the call using super() as a first statement in constructor. Normally a superclass's constructor is called before the subclass's constructor. This is called constructor chaining.

Q.9 Enlist the places where the final keyword can be used.

Ans. : The final keyword can be used at three places : 1. For declaring variables 2. For declaring the methods 3. For declaring the class

Q.10 What is the difference between class and interface ?

Ans. :

- The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.
- The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.

Q.11 What is the use of instanceof operator ?

Ans. : The instanceof operator is used to test if an object is instance of some class.

Q.12 What is polymorphism ?

Ans. : Polymorphism is a mechanism which allows to have many forms of the method having the same name.

Q.13 Distinguish between static and dynamic binding.

Ans. : Static binding is a type of binding in which the function call is resolved at compile time.

The dynamic binding is a type of binding in which the function call is resolved at run time.

The static binding is called the early binding and the dynamic binding is called late binding.

Q.14 What is the purpose of 'final' keyword ?

Ans. : The keyword 'final' is used to avoid further modification of a variable, method or a class. For instance if a variable is declared as final then it can not be modified further, similarly, is a method is declared as final then the method overriding is avoided.

Q.15 What is meant by dynamic method dispatch ?

Ans. : The dynamic method dispatch is run time polymorphism in which a call to overridden function is resolved at runtime instead of at compile time. Hence is the name.

Q.16 What is the use of equals method of object class ?

Ans. : The equals method of object class is for comparing values specified by two objects.

Q.17 What is the use of System.out.println() statement ?

Ans. : The System.out.println is used to display some message on console.

...at is package ?

...kage defines the name ...e is created and its name is

What is the use of import

The import statement is us... ...in the package.

What is the purpose of ja...

The java.net package is ...king applications which ...

What is the purpose of

...The java.util package i... ...om number generation, e...

What is static import ...

...: Static import is a feat... ...class name. For exampl... ...port static java.lang.Math...

What is the advantag...

...: The advantage of us... ...creases as it avoids writin...

Enlist four Java AP...

...: 1. java.util 2. java...

What is an excep...

...: Exception is a me... ...ccur in the program. Fo...

ArithmeticException : T... ...ivide by zero.

IOException : This excep...

Q.2 What will happen...

Ans. : An uncaught exc... As a result eventually th...

Q.1 What is package ?

Ans. : Package defines the name space in which the classes are stored. In Java a directory is created and its name is package name.

Q.2 What is the use of import statement in Java ?

Ans. : The import statement is used to import the package. Java makes use of the class present in the package.

Q.3 What is the purpose of java.net package ?

Ans. : The java.net package is for providing the useful classes and interfaces for the networking applications which are used in sockets and URL.

Q.4 What is the purpose of java.util package ?

Ans. : The java.util package is used for using some commonly used utilities such as random number generation, event model, date and time facilities and so on.

Q.5 What is static import ?

Ans. : Static import is a feature in which the import statement must be written along with class name. For example -

```
import static java.lang.Math.sqrt;
```

Q.6 What is the advantage of static import ?

Ans. : The advantage of using static import statement is that the readability of the code increases as it avoids writing of belonging class name each time.

Q.7 Enlist four Java API packages that are commonly used.

Ans. : 1. java.util 2. java.net 3. java.lang 4. java.awt

Chapter 6 : Exception Handling

Q.1 What is an exception ? Give example.

Ans. : Exception is a mechanism which is used for handling unusual situation that may occur in the program. For example -

ArithmeticException : This exception is used to handle arithmetic exceptions such as divide by zero.

IOException : This exception occurs when an illegal input/output operation is performed.

Q.2 What will happen if an exception is not caught ?

Ans. : An uncaught exception results in invoking of the uncaughtException() method. As a result eventually the program will terminate in which it is thrown.

Q.3 What is the benefit of exception handling ?

Ans. : When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.

Q.4 What is compile time and run time error ?

Ans. : The errors that are detected by the Java compiler during the compile time are called compiler time errors. The runtime errors are basically the logically errors that get caused due to wrong logic.

Q.5 What is the use of try, catch keywords ?

Ans. : try - A block of source code that is to be monitored for the exception.

catch - The catch block handles the specific type of exception along with the try block.

Q.6 What is the difference between throw and throws ?

Ans. :
- The throw keyword is used to explicitly throw an exception.
- The throws keyword is used to declare an exception.

Q.7 What is ArrayIndexOutOfBoundsException ?

Ans. : When we use an array index which is beyond the range of index then ArrayIndexOutOfBoundsException occurs.

Q.8 What is the need of multiple catch ?

Ans. :
- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.

Q.9 What is finally ?

Ans. : The finally specifies the code that must be executed even though exception may or may not occur.

Q.10 What is IOException ?

Ans. : When an illegal input/output operation is performed then IOExcpetion is used.

Q.11 Explain the checked and unchecked exceptions.

Ans. :
- **Checked Exception :** These types of exceptions need to be handled explicitly by the code itself either by using the try-catch block or by using throws.
- **UnChecked Exception :** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions.

How to throw the cu

Following is a syntax for th
throw our own exception
syntax for throwing out own
throw new Throwable's s
the Throwable's subclass is

What is throwable class

: The Throwable class is a

What is the difference

Error
Errors can not be handled.
Program crashes or stops wo error occurs.

Cha

Q.1 What is thread ?

Ans. : Thread is a tiny light-weight process.

Q.2 Write the differen

Ans. : Thread is a lighty

Q.3 Enlist various lif

Ans. : Various states of
1. New or create
4. Blocked

Q.4 What are the t

Ans. : Thread can be

Q.5 What is the u

Ans. : The run() met

Q.6 least

...n custom exception ?

...ream for throwing the custom exception

...exceptions using the keyword throw.

...ing out own exception is -

...Throwable's subclass

...subclass is actually a subclass derived from the Exception class.

...rowable class ?

...class is a superclass of all errors and exceptions in Java.

...difference between Error and Exception ?

Error	Exception
...e handled.	Exceptions can be handled using exception handler.
...m or stops working when an	Program reports user friendly message about the abnormal situation and exits gracefully.

Chapter 7 : Multithreaded Programming

...s thread ?

...is a tiny program running continuously. It is sometimes called as ...process.

...the difference between Thread and Process.

...d is a lightweight process while process is a heavy weight process.

...various life cycle states of a thread.

...his states of the thread are

...w or create 2. Runnable 3. Waiting

...Locked 5. Terminated

...re the two ways of implementing the thread ?

...ad can be implemented using

...e the use of run() method in thread programming ?

...run() method is used to implement thread in

w to throw the custom exception ?

lowing is a syntax for throwing the custom exception

row our own exceptions using the keyword throw.

x for throwing out own exception is -

row new Throwable's subclass

Throwable's subclass is actually a subclass derived from the Exception class.

What is throwable class ?

The Throwable class is a superclass of all errors and exceptions in Java.

What is the difference between Error and Exception ?

Error	Exception
rs can not be handled.	Exceptions can be handled using exception handler.
gram crashes or stops working when an or occurs.	Program reports user friendly message about the abnormal situation and exits gracefully.

Chapter 7 : Multithreaded Programming

What is thread ?

s. : Thread is a tiny program running continuously. It is sometimes called as ht-weight process.

Write the difference between Thread and Process.

ns. : Thread is a lightweight process while process is a heavy weight process.

Enlist various life cycle states of a thread.

ns. : Various states of the thread are

1. New or create 2. Runnable 3. Waiting
4. Blocked 5. Terminated

Q.4 **What are the two ways of implementing the thread ?**

Ans. : Thread can be implemented using 1. Thread class 2. Runnable interface

Q.5 **What is the use of run() method in thread programming ?**

Ans. : The run() method is used to implement thread's behavior.

Q.6 **Give at least one reason- why runnable interface is preferable than the thread class.**

Ans. : Following are the two reasons

1. If a class extends a thread class then it can not extends any other class which may be required to extend.
2. If a class thread is extended then all its functionalities get inherited. This is an expensive operation.

Q.7 What is thread priority ?

Ans. : Thread priority is a simple integer value that can be assigned to the particular thread. The Java thread scheduler selects the threads using their priorities.

Q.8 Enlist the two Java functionalities that are associated with the thread priority.

Ans. : **1. setPriority :** This function is used to set the priority to each thread.

2. getPriority : This function is used to get the priority for the thread.

Q.9 What is the meaning of the term preemption ?

Ans. : The preemption is a situation in which when the currently executed thread is suspended temporarily by the highest priority thread.

Q.10 What is daemon thread ?

Ans. : Daemon thread is a low priority thread which runs in background. For example the thread doing the garbage collection operation for the Java runtime system is a daemon thread.

Q.11 What is ThreadGroup ?

Ans. : A ThreadGroup is a collection of threads that can be managed together.

Q.12 What is thread synchronization ?

Ans. : When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization

Q.13 Explain the term - monitor used in thread synchronization.

Ans. : The thread synchronization is based on the concept called monitor. Monitor is used as mutually exclusive lock or mutex. When thread owns this monitor at a time then the other thread can not access the resources.Other thread have to be there in waiting state.

Q.14 What are wait and sleep methods in Java ?

Ans. : The wait and sleep methods are used to make the thread to wait for some predefined time. The wait is called on object while sleep is called on thread.

Q.15 What is the use of notify() method ?

Ans. : If particular thread is in the sleep mode then that thread can be resumed us
the notify call.

What is the use of notifyall() method ?

: This method resumes all the threads that are in suspended state.

What is deadlock ?

: Deadlock is a situation in which when two or more threads are waiting for each er to release the objects they are holding.

Chapter 8 : Input Output Programming

What is stream ?

ns. : Stream is basically a channel on which the data flow from sender to receiver.

What is input stream and output stream ?

ns. : An input object that reads the stream of data from a file is called input stream nd the output object that writes the stream of data to a file is called output stream.

What is byte stream ? Enlist its super classes.

ns. : The byte stream is used for inputting or outputting the bytes. The **InputStream** nd **OutputStream** are the superclass of byte stream.

What is character stream ? Enlist its super classes.

ns. : The character stream is used for inputting or outputting the characters. The Reader and **Writer** are the superclass of character stream.

Q.5 What is the use of file ?

Ans. : Files are useful for storing the persistent and shared information.

Q.6 What is absolute file name ?

Ans. : The filename that can be specified with the complete path name and drive letter is called absolute file name.

Q.7 What is relative file name ?

Ans. : The file name which is specified as a path relative to the current directory is called relative file name. For example new File("myprogram.html");

Q.8 What is file constructor ?

Ans. : The file constructor is used in java while handling Java I/O. For example

```
File f1=new File("input.dat");
```

Q.9 What is the use of seek method ?

Ans. : The seek() allows you to specify the index from where the read and write operations will start. The syntax is

```
void seek(long position);
```

Q.10 Mention the syntax of file object.

Ans. : File(Sting Dir_path)

File(String Dir_path,String String File_name)

File(File Dir_Obj,String File_name)

Q.11 Write a Java code to check if the command line argument is file or not.

Ans. :

```
class Test
{
    public static void main(String [] args)
    {
        File obj=new File(args[0]);
        If(obj.isFile())
        {
            System.out.println("This is a file name"+obj.getPath());
        }
    }
}
```

Q.12 Enlist the two common constructors of FileInputStream.

Ans. : The common constructors of FileInputStream are

FileInputStream(String filename);

FileInputStream(File fileobject);

Q.13 What is the use of Input Stream Reader and Output Stream Writer ?

Ans. : The InputStreamReader converts byte into character and OutputStreamWriter converts the characters written into byte.

Q.14 What is the use of FileReader and FileWriter ?

Ans. : Normally for reading the simple text files the FileReader is used. FileWriter class is used for writing the contents to the text file.

Q.15 How will you read a single character on the console ?

Ans. :

```
import java.io.*;
class ReadChar
{
 public static void main(String args[]) throws IOException
 {
  //declaring obj for read() method
  BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
  char ch;
  ch=(char)obj.read();//reading single character
  System.out.println(ch);//outputting it
 }
}
```

Chapter 9 : Collection Classes

What is collection ?

: Collection is a group of objects which are designed to perform certain task. tasks are associated with alteration of data structures.

Enlist various collection interfaces.

: Various collection interfaces are - List, Map, Set and Queue

Enlist the two commonly used exceptions by the collection interface.

: 1. UnsupportedOperationException 2. ClassCastException

Enlist any five collection classes.

: 1. AbstractCollection 2. AbstractList 3. HashSet 4. Vector 5. PriorityQueue

What is AbstractList collection class ?

: The AbstractList class implements List interface. Using this class it is easier to ate a List based on random access data structure.

What is ArrayList ?

: The implements the List interface. It is used to implement the dynamic array.

What is dynamic array ?

: The dynamic array means the size of the array can be created as per the quirements. The ArrayList is used for creating dynamic array.

Which java package need to be imported for using the collection classes ?

: java.util package is required for using the collection classes in Java

Enlist various methods used by LinkedList.

: Various methods used by LinkedList are - add(), remove(), addFirst(), ddLast(),removeFirst(),removeLast().

What is enumeration ?

: Enumeration generates series of elements one at a time. This interface is used to ccess the elements defined in some collection like **Vector** or **ArrayList**.

Name the any two methods used by Enumeration.

: The enumeration makes use of two methods -

1. **nextElement() :** It returns the next object in the list.
2. **hasMoreElements() :** If the next element is present in the collection then it returns true otherwise returns false.

Q.12 What is the difference between Vector and ArrayList ?

Ans. : Vector and ArrayList both implement the dynamic array. But the difference between the two is that -

1. All the methods in vector are synchronized whereas the methods in the ArrayList are not synchronized.
2. Vector doubles the size of its array when its size is increased. The ArrayList on the other hand increases by half of its size when the size is increased.
3. The ArrayList is faster than Vector.

Q.13 What is the use of Java util package ?

Ans. : In Java, the UTIL package defines the number of useful classes.The UTIL package provides the facility for Collection framework, event model, date and time facilities and many other utility classes.

Q.14 Enlist the classes used in Java Util package.

Ans. : Various classes supported by Java Util Package are - Collection, Comparator, Map, Set, List, AbstractList, Date, Dictionary.

Chapter 10 : Networking with Java.net

Q.1 What is the purpose of Java.net package ?

Ans. : The java.net package is for providing the useful classes and interfaces for networking applications which are used in sockets and URL.

Q.2 What is ContentHandler ?

Ans. : This class is a superclass of all the classes that read the data from a class URLConnection. It also builds the appropriate local object based on MIME types.

Q.3 What is DatagramSocket ?

Ans. : This class represents the Datagram Socket (UDP socket).

Q.4 What is ServerSocket ?

Ans. : This class is used for implementing server-side sockets this class is useful.

Q.5 What is InetAddress Class ?

Ans. : The InetAddress class from **java.net** package represents the IP addresses. It works with either host name or numerical IP address of corresponding host.

Q.6 Enlist the three important method used by InetAddress Class

Ans. : The three important methods from InetAddress class are

1. getLocalHost 2.getByName 3. getAllByName

Write a Java code to obtain the IP Address of the local host machine

```
import java.net.*;
class InetAdd1

  public static void main(String args[]) throws UnknownHostException
  {
      InetAddress local_add=InetAddress.getLocalHost();
      System.out.println("Local Host is: "+local_add);
  }
}
```

Q.8 Write a Java code to obtain the host name of the local host machine when IP addess is given

Ans. :

```
import java.net.*;
class InetAdd4
{
  public static void main(String args[]) throws UnknownHostException
  {
      InetAddress addr=InetAddress.getByName("192.168.0.166");
      System.out.println(addr.getHostName());
  }
}
```

Q.9 Enlist the methods that can be used to identify different types of IP addresses.

Ans. : Various methods are -

1. **isLoopbackAddress()**
2. **isAnyLocalAddress()**
3. **isSiteLocalAddress()**
4. **isMulticastAddress()**

Q.10 What are Inet4Address and Inet6Address classes ?

Ans. : The Inet4Address and Inet6Address classes are inheriting the basic **InetAddress** class. These classes are for supporting IPv4 and IPv6 respectively.

Q.11 Enlist some important methods of InetAddressClass.

Ans. :

Q.13 What is socket programming ?

Ans. : The programming in which the client server communication is demonstrated is called socket programming.

Q.14 What is reserved port ? Give example.

Ans. : There are some ports reserved for some specific services. These ports are called reserved ports. For example - port number 21 is for FTP service, port 80 is for HTTP protocol.

Q.15 What is client and server ?

Ans. : Client is a device which wants to get service from particular server. While server is a device which has resources and from which the services can be obtained. For example : There are various types of servers such as web server which is for storing the web pages, there are print servers for managing the printer services or there are database servers which store the databases.

Q.16 What is DatagramSocket Class ?

Ans. : A DatagramSocket class acts as a sending or receiving point for a packet delivery service.

Q.17 What is DatagramPacket Class ?

Ans. : DatagramPacket class is used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet.

Q.18 Enlist two methods used in DatagramPacket.

Ans. :

int getLength()	Returns the length of the data to be sent or the length of the data received.
int getPort()	Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received.

Q.19 What is bind method ?

Ans: The bind method is used for binding the DatagramSocket to a specific address and port.

Q.20 What is the purpose of send and receive methods of DatagramSocket class?

Ans: The Send method sends a datagram packet from the socket and the receive method receives the datagram packet from the socket.

Chapter 11 : Introduction to OOMD Techniques

What is OOMD ?

: The OOMD i.e. Object Oriented Modeling and Design is a software engineering ach which models the system as interacting objects. Each object represents a n entity which plays a vital role in building of that system.

What is object orientation ?

: Object orientation means organization of software as a collection of discrete cts. Each object consists of data structure and behavior which are closely coupled.

Enlist basic characteristics required by OO approach.

s. : Various basic characteristics required by OO approach are -

1. Identity 2. Classification 3. Inheritance 4. Polymorphism

What does the identity characteristic of OO approach specify ?

s. : The identity means data is arranged in distinguishable entities called objects.

ch object is treated as an inherent entity that means - each object is distinct even if eir attribute values are same.

Mention two reasons- Why models are created ?

ns. : Following are reasons for model creation - 1. Communication with Customer - Testing of physical entity 3. Complexity reduction 4. Visualization of the system.

What is abstraction ?

Ans. : Abstraction means examining the selective area of the problem. The goal of abstraction is to separate out all the important aspects of the specific area of the problem and ignore all unimportant aspects of that area.

Q.7 Enlist the three viewpoints used to model the system.

Ans. : The three viewpoints used to model the system are

1. Class Model 2. State Model 3. Interaction Model

Q.8 What does the class model represent ?

Ans. : The class model represents the static, structural, data aspects of the system.

Q.9 What does the state model represent ?

Ans. : The state model represents the temporal, behavioral, control aspect of the system.

Q.10 What does the interaction model represent ?

Ans. : The interaction model represents collaboration of objects and interaction aspect of the system.

Q.11 Enlist various diagrams that can be created during interaction modeling.

Ans. : Various diagrams that are created during the interaction modeling are - sequence diagram, use case diagram and activity diagram.

Q.12 Why do we need object oriented systems development ?

Ans. : The object oriented systems development is needed to model the requirements of the system.

Q.13 What is the need for modeling ?

Ans. : The modeling is required for specifying, visualizing, constructing and documenting the artifacts of software systems.

Chpater 12 : Class Modeling

Q.1 What is object ?

Ans. : The object is an instance of a class.

Q.2 What is class ?

Ans. : Class is a group of objects having same attributes and operations, relationships and semantics.

For example - Student, Employee, Company, Course all are classes. Each Student has rollnumber, name and address.

Q.3 Give the hint to identify the classes from the given problem statement.

Ans. : The classes appear as common nouns or noun phrases.

Q.4 Give the hint to identify the attributes of a class.

Ans. : Attribute is a logical data value of an object. Generally the entities that store some data value becomes the attributes of a class.

Q.5 Name the two diagrams that are normally drawn to design the class Model.

Ans. : The class diagram and object diagrams are drawn to design the class model.

Q.6 What is class diagram ?

Ans. : Class diagram is a graphical representation used for modelling classes and their relationships. It describes all possible objects belonging to the classes.

Class diagram is used for abstract modelling and for implementing actual program.

Q.7 What is object diagram ?

Ans. : The object diagram represents the objects and their relationships with each other.

Using the one class diagram various object diagrams can be created.

...What is link and associa...

...Link is a connection betw... with another.

...sociation is a group of lin...

...What is multiplicity ?

...The multiplicity represen... multiplicity can be writte... ...ated with other object.

...Give various notations u...

Notations
1
0..1
*
0..*
1..*
2..10

...11 What is the difference b...

...: The multiplicity is the ...hereas the cardinality is the c...

...12 What is ordering in asso...

...: Ordering is a type of as... ...ust appear in some specific o... ...d.

...13 What are Bags and Sequ...

...: Bag is a collection of obj... ...ord bag appears at some end ...equence is a collection of obje... ...equence.

...14 What is association class...

...: When two classes are re... ...ssociation itself can have attr... ...epresented by a class. Such a cl...

What is link and association ?

: Link is a connection between the objects. It represents a simple association of one ct with another.

ssociation is a group of links that have common structure and common semantics.

What is multiplicity ?

: The multiplicity represents "how many" objects are connected.

multiplicity can be written as expression. It describes "one" or "many" objects ciated with other object.

Give various notations used for multiplicity.

:

Notations	Description
1	Only one instance
0..1	Zero or one instance
*	Many instances
0..*	Zero or many instances
1..*	One or many instances
2..10	You can specify the integer range

What is the difference between multiplicity and cardinality ?

s. : The multiplicity is the constraint on the collection of the associated objects hereas the cardinality is the count of the objects that are in collection.

What is ordering in association ?

s. : Ordering is a type of association is used to denote the set of objects at one end ust appear in some specific order. The {ordered} keyword is used at the association d.

What are Bags and Sequences ?

ns. : Bag is a collection of objects at other end that are duplicated. That means if the ord bag appears at some end then the object at that end can appear more than once.

equence is a collection of objects at other end that are duplicated or appear in some equence.

What is association class ?

Ans. : When two classes are related with each other by an association link, then the ssociation itself can have attributes and operations. Hence this association can be epresented by a class. Such a class is called association class.

Q.15 What is qualified association ?

Ans. : The qualified association has qualifier which is used to select particular object from the set of objects. Refer Fig. 12.3.7.

Q.16 When do we use association ?

Ans. : The association relationship is used in the system design when we want that the relation between two object has to remembered for some time.

Q.17 Why should we avoid using too many associations in the class model ?

Ans. : If too many associations are shown in the class model then the model becomes complex.

Q.18 List out the relationships used in class diagram.

Ans. : Following are the relationships that can be used in the class diagram -

1. Association
2. Aggregation
3. Composition
4. Generalization

Q.19 What is generalization relationship ?

Ans. : Generalization is the relationship between a superclass and one or more subclasses.

Chapter 13 : Advanced Class Modeling

Q.1 What is scope ? Enlist types of scope used in the class.

Ans. : Scope represents whether particular feature is available for class or object. There can be static scope and dynamic scope.

Q.2 What is visibility ?

Ans. : The visibility specifies the how the attributes and operations are visible in the system. There are four levels of visibility -public, protected, private and package.

Q.3 What are the properties that are associated with the association end ?

Ans. : Various properties of association end are -

1. Association End Name 2. Multiplicity 3. Ordering 4. Bags and Sequences

Q.4 What is n-ary association ?

Ans. : The n-ary association is the association among three or more classes. Normally having n-ary association in the class diagram is not preferred and therefore it is a practice to split n-ary association into binary association.

select particular object

hen we want that the

class model ?

the model becomes

diagram -

and one or more

ss or object. There

are visible in the
d package.

n end ?

nd Sequences

lasses. Normally
therefore it is a

Q.5 Define aggregation and Composition.

Ans. : Aggregation : Aggregation is used to represent the whole-part relationship. It normally posses the Has-a relationship. For example - DVD player and Car are the two classes that can be associated by the aggregate relationship.

Composition : Composition is a special kind of aggregation which represents the whole-part relationship. To be more specific, a restricted aggregation is called composition. For example - Steering and Car are the two classes those can be associated by composite relationship.

Q.6 Differentiate aggregation with composition.

Ans. : Both the composition and aggregation represent the whole part relationship but the composition is more restricted than the aggregation. The composed object can not exist without the other object. Hence it is a strong relationship.

This restriction is not there in aggregation. The existence of contained object is entirely optional in case of aggregation.

For example - Books Library contains books and students. The student and Library share the aggregate relationship but the Book and Library share the composition relationship. As without books the library is not possible.

Q.7 What is abstract class ?

Ans. : The abstract classes are those classes that do not have the direct instances. The names of abstract classes are denoted in italics.

Q.8 What is multiple inheritance ?

Ans. : The multiple inheritance is a kind of inheritance in which the derived classes are derived from more than one super classes. The class with more than one superclass is called join class. Refer Fig. 13.6.1.

Q.9 What is delegation ?

Ans. : Delegation an implementation mechanism in which object forwards one operation to another object for execution. The advantage of delegation is that only meaningful operations are delegated and there is no danger of inheriting meaningless operations.

Q.10 What is metadata ?

Ans. : Metadata is a data that describes other data. The class definition is basically a metadata. For example - Consider a banking system, in which the records for the customer and their accounts is maintained. A separate description class AccountDescription is maintained which describes the account number. This class is called metadata.

Q.11 What is constraints ?

Ans. : Constraints are commonly used in class diagram. In UML constraint is a condition or restriction on UML element. These elements can be objects, classes, attributes, links, associations and generalization sets.

Q.12 How are the constraints represented ?

Ans. : Constraint is a condition (a Boolean expression) which must evaluate to a Boolean value - true or false. . A constraint is shown as a text string in curly braces.

Q.13 What is overlapping constraint ?

Ans. : The overlapping constraint specifies that objects of parent may have more than one of the children as type. For example - Here train can be passenger type vehicle or goodscarrier.

Q.14 What is derived Data ?

Ans. : Derived data is a function which is obtained using one or more functions or data elements.

The derived data is redundant because it is obtained from other element.

In class diagram, data attributes, associations and classes can be derived.

Q.15 What is package ?

Ans. : package is a general purpose mechanism for modeling the elements into groups. Using packages the elements can be organized properly.

Chapter 14 : State Modeling

Q.1 Give the use of state diagram.

Ans. : UML state machine diagram represents the events and states of object and the behavior of object in reaction to an event.The state chart diagram shows the life cycle of the object.

Q.2 Give the meaning of event ?

Ans. : Event : An event is significance occurrence of something.

For example : Customer inserts ATM card.

Q.3 What is state and transition ?

Ans. : States : A state is a condition of object at a particular moment of time. This time is mostly between the events.

For example : 1. Waiting for user to enter PIN 2. Authenticating the User

Transitions : The relationship between the two states is indicated using the transition. When some event occurs the object moves or transits from one state to another. This is represented by the transition.

onstraint is a condition
asses, attributes, links,

evaluate to a Boolean
ly braces.

may have more than
enger type vehicle or

or more functions or

ement.
rived.

lements into groups.

es of object and the
shows the life cycle

t of time. This time

e User
using the transition.
to another. This is

example : When the system operator "performs startup"(an event occurs) the sition from Idle state to Active state occurs.

What is state independent object ?

: If the object responds to the same way to an event then that object is considered state-independent with respect to that event. For example - If an object receives the ssage "Customer finished the transaction" or "Customer pressed the Cancel button" corresponding method will be Ejecting Card.

What is state dependent object ?

: If objects respond differently to the events depending upon their states or mode n such objects are considered as state-dependent objects.

example - For an "insert Card" event the state will be "Reading Card" whereas ter PIN" event the state will be "Validating PIN"

What is the difference between event and state ?

: Event represents particular moment of time whereas the events represent interval ime.

What is Guard Condition ?

: It is a Boolean expression and is evaluated when the transition is triggered.

What is target state ?

: The active state after completion of transition is called target state,

What is one shot state diagram ?

: The state diagram that has finite number of states representing one-shot life cycle epresenting continuous loop then it is called one shot state diagram.

What is activity in state diagram ? How is this represented in state diagram ?

: Activity is an actual behavior that may occur due to any number of effects. It can represented by "/" and the activity name following the event due to which the ivity occurs. Using the keyword do the ongoing activity can be represented.

What is do-activities ?

: When object is in some state then that object may perform some work. This going activity is -modeled by do-activities.The notation do/ is used to represent the activities.

What are entry activities ?

: There are some activities that can be performed on entry of particular state. ese activities are called entry activities.

While exiting the state some cleanup actions are need to be performed. These activities called exit activities.

Q.13 If a state has multiple activities then in which order they must be executed

Ans. : The execution order will be -

1. Activities on incoming transitions
2. Entry activities
3. Do activities
4. Exit Activities.

Q.14 What is the difference between events within a state and self-transition ?

Ans. : The event within a state cause an activity to be performed. On the other hand self-transition causes entry and exit activities to be executed.

Q.15 What is nested state ?

Ans. : The nested states are the states that occur within the one super state.

Chapter 15 : Interaction Modeling

Q.1 What do you understand by the term use cases ?

Ans. : Use cases are the text documents and the use case modeling is an act of writin the text and not the diagrams.

Q.2 What do you mean by actors ?

Ans. : Actors : The actors are the entities that interact with the system or produc within the context of behavior of the system. Actor can be person, computer system o organization.

Q.3 List the relationships used in use cases.

Ans. : Following are the relations that are used in use cases -

1. Include 2. Extend 3. Generalization

Q.4 List out the steps for finding use cases.

Ans. : Following are the steps used for finding the use cases -

1. Identify the actor. And for each actor, find the tasks or functions that actor can perform or a system wants that an actor should perform it.
2. Name the use cases.
3. Describe use cases by using the terminologies with which user of the system is familiar. The description but be less/no ambiguous.

Q.5 What is secondary or supporting actor ?

Ans. : The supporting actors are used in conjunction with the primary actors in order to support for the main services. The secondary actors support the system in such a way

med. These activities are

must be executed ?

self-transition ?

On the other hand the

uper state.

g is an act of writing

system or product
computer system or

functions that actor
n it.

user of the system

in order to
way

that the primary actors can perform their task. For example payment validation system s a supporting actor for the on-line purchase system.

Q.6 What is scenario ?

Ans. : Scenario is a specific sequence of actions or interactions between actor and the system. It is also called as use case instance. The scenarios describe the text stories. For example : scenario of issuing of a book from library, withdrawal of money from ATM.

Q.7 Explain the include relationship.

Ans. : This is the most commonly used relationship which denotes that a given use case may include another.

Q.8 Explain the extend relationship.

Ans. : The extend relationship is used when an extended use case is connected to the base use case. While using the extend use the extension points can be explicitly shown in the base use cases.

Q.9 What is extension point ?

Ans. : The extension points are used when we use the extend relationship. The extension points are the labels in the base use case which are referred by the extending use case. These are triggered by some conditions.

Q.10 Explain the generalization relationship.

Ans. : Generalization is used when the we find two or more use cases having common structure, behavior and purpose. The test word used for generalization relationship is "kind of".

Q.11 What is sequence diagram ?

Ans. : The sequence diagram is a graphical representation that shows the sender and receiver objects along with sequence of messages.

Q.12 What is activity diagram?

Ans. : An activity diagram is just similar to the flow chart in which flow of control from control from one activity to another activity is shown.

Q.13 What is the meaning of activities ?

Ans. : The activities are nothing but the operations. These operations can be identified from the state diagram.

Q.14 What is bull's eye in activity diagram ?

Ans. : In activity diagram the termination is represented by ... rcle surrounded by a hollow circle.

Q.15 What is the way by which the concurrent activities are represented ?

Ans. : The concurrent activities are represented by forking and joining in activity diagram

Q.16 What is fork and join in activity diagram ?

Ans. : In activity diagram, the fork represents the separation of two control flows whereas join represents joining of two control flows.

Q.17 What is synchronization bar ?

Ans. : In activity diagram, the synchronization bar are represented by thick horizontal line. The synchronization bar is used to represent the fork and join in activity diagram.

Q.18 How to represent the flow of execution of the process in activity diagram ?

Ans. : In activity diagram, the activity token can be placed on the activity node to represent that certain activity is executing. And when the activity completers the token can be removed and placed on outgoing array.

Q.19 Enlist two guideline principles for modeling activity diagram.

Ans. : 1. Carefully handle branches and conditions

2. Carefully handle concurrent activities.

Q.20 What is swimlane ?

Ans. : Many times the activity diagram needs to partition in groups to represent the flow of business processes. Each group represents the particular business flow belonging to some category such group is called swimlane.

Q.21 What is 'object flow' in activity model ?

Ans. : The objects that are associated with the control flow of the activity diagram is called object flow.

□□□