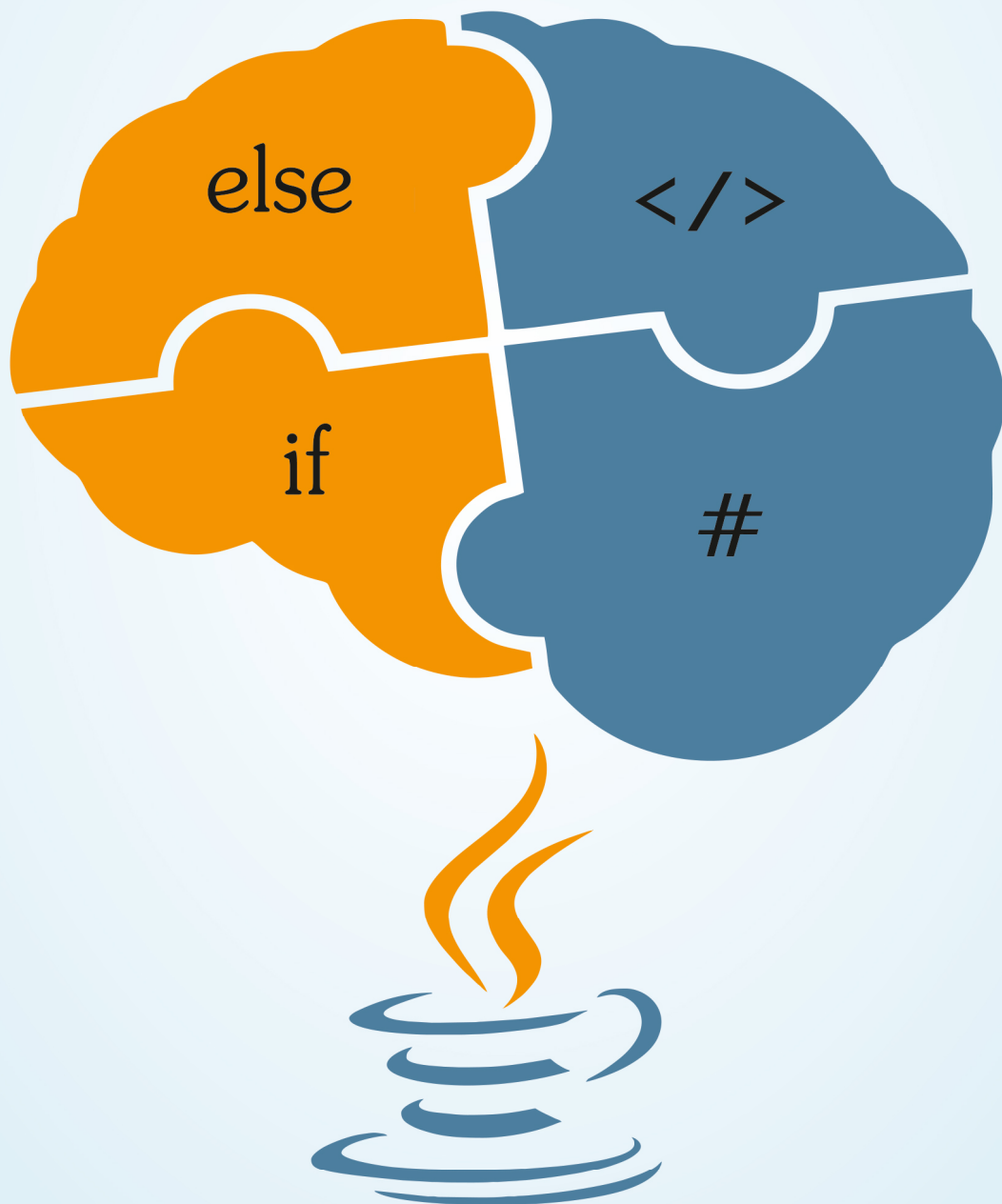


PROGRAMMING

in

JAVA



Dr. V. B. Narasimha, Dr. Yogesh Kumar Sharma
Dr. S. Nagaprasad, Dr. Manju khari


Programming
in
JAVA



Published by: **Book Bazooka Publication**

Website: BookBazooka.com

Email Address: info@bookbazooka.com

 BookBazookaOfficial

 BookBazooka

 BookBazooka

 BookBazooka

 BookBazooka

 BookBazooka

ISBN: 978-93-86895-76-9

Price: 450.00/- inr

Authors: Dr. V. B. Narsimha, Dr. Yogesh Kumar Sharma, Dr. S. Nagaprasad & Dr. Manju Khari

© All Rights including Copyrights reserved with the Authors.

Publication Year: - 2019

All rights reserved. No part of this book may be reproduced or utilised in any form or by any electronic, mechanical or means, now known or hereafter invented, including photocopying and recording, or in any information storage and retrieval system without permission in writing from the publisher or author.

About Authors



Dr. V. B. Narsimha

Dr. V. B. Narsimha working as Assistant Professor in the Department of Computer Science and Engineering at University College of Engineering, Osmania University, Hyderabad Telangana. He completed his MCA from University College of Engineering, Osmania University, Hyderabad, He got his M.Tech (Computer Science and Engineering) degree from IETE, Hyderabad. His Ph.D completed from Osmania University. His research areas are Data Mining, Networking, Image Processing, Machine Learning, Software Engineering etc. In his research life he published around 40 international journal papers in his research area. In his research career he presented around 25 National and International conference papers, for his research interest he around attended 20 National and International workshops. He worked as a Lecturer in Department of Computer Science, Osmania University, Post Graduate College, Gadwal and Nalgonda from 01.08.1994 to 22.08.2000 and 01.08.2002 to 14.03.2004 respectively. Presently under his guidance 15 scholars are working in various areas at faculty of Informatics, Department of Computer Science and Engineering, Osmania University, Hyderabad and one scholar awarded Ph.D in the year 2019. Guiding 2 PhD scholars in Department of Computer Science and Engineering, JNTU, Hyderabad. Guiding 2 PhD scholars in Department of Computer Science and Engineering, K L University, Vijayawada, Andhra Pradesh. Guided more than 150 students in their academic projects in Computer Science and Engineering, University College of Engineering, Osmania University. Course Writer for PGRRCDE, Osmania University and Institute of Public Enterprises. Worked as System Analyst in Software Services and Resources Inc. Atlanta, Georgia (USA) from 28.08.2000 to 18.04.2002. He worked as a principal Osmania University P.G.College, Gadwal in the year 1997 to 2000.



Dr. Yogesh Kumar Sharma

Dr. Yogesh Kumar Sharma, Presently working as a Associate Professor (HOD / Research Coordinator) Department of Computer Science Engineering and IT at "Shri Jagdish prasad Jhabarmal Tibrewala University", Chudela, Jhunjhunu (Rajasthan). He completed his B.Sc with Computer Application in the year of 2002 awarded from S.M.L. (P.G.) College, Jhunjhunu, University of Rajasthan, M.C.A from Modi Institute of Management and Technology, Kota, University of Kota in the year 2005, Ph.D. in Faculty of Computer Science, from "Shri Jagdish prasad Jhabarmal Tibrewala University", Jhunjhunu (Rajasthan) in the year 2014. His research areas Data Communication & Networking, Operating System, Computer Organization and Architecture, Data Mining, Image processing, Cloud Computing, Software Engineering etc. In his research life he Published 75 Papers in National and International journals, 25 National and International conferences, 03 workshops. Under his guidance 05 research scholars awarded Ph.D. Presently 08 research scholars working under his guidance. He invited as a Guest Lecturer for Students in M.Sc. I.T. (Master Program in Information Technology) on the behalf of National University of Science and Technology, Muscat, Oman, Nov. 2019. He is Paper Setter, Answer Sheet Evaluator (Copy Checker) and Practical Examiner in University of Rajasthan, Jaipur, Pandit Deendayal Upadhyay University (Shekhawati University), Sikar, Maharaja Ganga Singh University, Bikaner, University of Kota, Kota, Board of Secondary Education Rajasthan, Ajmer. Ph.D. He is Evaluator and Ph.D. Final Viva-Voce Examiner in OPJS University, NIMS University, Mewar University. Published 2 patents on Title: Parallel Processing System to Reduce Complexity in Data-Mining of Industrial & Social Big-Data. And Title: Computer Implemented Method for Detecting Downlink Control Channel in Long Term Evolution Wireless Communication. He has a member of IAENG, IACSIT, CSTA and UACEE.



Dr. S. Nagaprasad

Dr. S. Nagaprasad working as a Faculty in Computer science and Applications, Dept. of Computer Science and Applications, Tara Government College, Sangareddy, Telangana state. He completed his B.C.A. in the year of 1998-2001 awarded from Osmania university, M.Sc (I.T.) in the year of 2001-2003 awarded from Sikkim Manipal University, Sikkim, his research work completed from Ph.D in Computer Science and Engineering, from Acharya Nagarjuna University, Sep-2015. His research areas data mining, networking, image processing, machine learning etc. In his research life he present completed 30 international journals in his research area, 15 National and International conferences, for his research interest he around attended 10 workshops. He worked as faculty in Computer Science at S.K.N.R. Government Arts and Science College, Jagtial Telangana state for 10 years, and he worked as faculty in Computer Science at S.R.R. Government Arts and Science College, Karimnagar Telangana state for 05 years. Presently he is Guiding 5 Ph.D. scholars in Computer Science and Engineering “Shri. Jagdishprasad Jhabarmal Tibrewala University”, Jhunjhunu (Rajasthan).



Dr. Manju Khari

Dr. Manju Khari an Assistant Professor in Ambedkar Institute of Advanced Communication Technology and Research, Under Govt. Of NCT Delhi affiliated with Guru Gobind Singh Indraprastha University, Delhi, India. She is also the Professor-In-charge of the IT Services of the Institute and has experience of more than twelve years in Network Planning & Management. She holds a Ph.D. in Computer Science & Engineering from National Institute Of Technology Patna and She received her master's degree in Information Security from Ambedkar Institute of Advanced Communication Technology and Research, formally this institute is known as Ambedkar Institute Of Technology affiliated with Guru Gobind Singh Indraprastha University, Delhi, India. Her research interests are software testing, software quality, software metrics, information security, optimization and nature-inspired algorithm. She has 70 published papers in refereed National/International Journals & Conferences (viz. IEEE, ACM, Springer, Inderscience, and Elsevier), 06 book chapters in a springer. She is also co-author of two books published by NCERT of Secondary and senior Secondary School.

Table of Contents

Unit-1

Aims and Objectives	1
1.1 Preface	1
1.2 Object oriented pattern	2
1.2.1 Basic view of OOP	2
1.2.2 Benefits of OOP	13
1.2.3 Applications of OOP	14
1.2.4 Java features	15
1.3 Overview of java language	18
1.3.1 Introduction	18
1.3.2 Simple Java program structure	19
1.3.3 Java tokens	21
1.3.4 Java Statements	24
1.3.5 Implementing a Java Program	25
1.3.6 Java Virtual Machine	27
1.3.7 Command line arguments	28
1.4 Identifier	29
1.5 Literals	30
1.5.1 Integer Literals	30
1.5.2 Character Literals	32
1.5.3 Boolean Literals	32
1.5.4 Floating Point Literals	33

1.5.5 String Literals	34
1.5.6 Null Literals	35
1.6 Constants, Variables & Datatypes	35
1.6.1 Introduction-Constants, Variables and Data Types	35
1.6.2 Declaration of Variables	43
1.6.3 Giving Value to Variables	44
1.6.4 Scope of variables	47
1.6.5 Symbolic Constants	51
1.6.6 Type casting	52
1.6.7 Getting Value of Variables	57
1.6.8 Standard Default values	58
1.7 Operators & Expressions	59
1.7.1 Operators	59
1.7.1.1 Assignment Operator(=)	59
1.7.1.2 Arithmetic Operators(+ , - , * , / , %)	60
1.7.1.3 Unary Operators(+ , - , ++ , -- , !)	64
1.7.1.4 Equality and Relational Operators(== , != , > , >= , < , <=),	67
1.7.1.5 Bit wise Operators(&(AND)	69
1.7.1.6 ^ (EXCLUSIVE OR)	70
1.7.1.7 (INCLUSIVE OR))	71
1.7.1.8 Conditional Operator (&& (AND)) , (OR))	72
1.8 Summary	72
1.9 Exercise	74
1.10 Objective Type Questions	75

Unit-II Conditional Statements and Loops

Aims and Objectives:	92
2.0 Conditional Control Statements	92
2.1. Selection Statements	92
2.1.1. Simple If	93
2.1.2 If-else	94
2.1.3 Nested if	96
2.1.4 Else if-ladder	98
2.1.5 Switch statement	101
2.1.6 Break Statement	103
2.1.7 Continue Statement	105
2.1.8 Labelled Statements	106
2.1.9 Labelled Break	106
2.1.10 Labelled Continue	108
2.2 While Statement	109
2.2.1 Do-While Statement	111
2.3 For loop	113
2.4 Enhanced for loop	115
2.5 Nested for loop	116
2.6 Summary	117
2.7 Exercise	118
2.8 Multiple Choice Questions	119

Unit-III Classes & Objects:

3.0 Aims and Objectives	129
3.1 Introduction	129
3.1.1 Defining and declaring a class	132
3.1.2 Adding variables	133
3.2 Creating an Object	134
3.2.1 Accessing class members	137
3.2.2 Constructors	140
3.3 Methods	145
3.3.1 Adding methods	146
3.3.2 Overloading methods	148
3.3.3 Overloading constructors	152
3.3.4 Access Control Specifiers	156
3.3.5 Nesting of methods	161
3.4 Concept of Static and Abstract (Simple application based examples)	163
3.5 Multithreaded Programming	169
3.5.1 Introduction	169
3.5.2 Creating Threads	170
3.5.3 Extending the Threads	170
3.5.4 Stopping and Blocking a Thread	174
3.5.5 Lifecycle of a Thread	174
3.5.6 Using Thread Methods	180
3.5.7 Thread Exceptions	181

3.5.8 Thread Priority	182
3.6 Synchronization	183
3.6.1 Implementing the 'Runnable' Interface	184
3.7 Arrays	186
3.7.1 Arrays	187
3.7.2 One-dimensional arrays	187
3.7.3 Creating an array	187
3.7.4 One-dimensional arrays	187
3.7.5 Two- dimensional arrays	189
3.8 Strings	190
3.9 Vectors	195
3.10 Wrapper classes	196
3.11 Summary	198
3.12 Exercise	199
3.13 Objective Type Questions	200

Unit-IV

4.0 Aims and Objectives	208
4.1 Introduction	208
4.2 Inheritance	208
4.2.1 Extending a class	209
4.2.2 Overloading methods	219
4.3 Final variables and methods	220
4.3.1 Final classes	222

4.4 Abstract methods and classes	222
4.4.1 Member access using super class	226
4.4.2 Member access using abstract classes	230
4.4.3 Call by value	230
4.4.4 Call by reference,	231
4.4.5 Overriding methods	231
4.5 Applets	234
4.5.1 Introduction	234
4.5.2 Types	234
4.5.2.1 local and remote applets	234
4.5.3 Applets and Applications	234
4.5.4 Building Applet code	234
4.6 Applet Life cycle	235
4.6.1 Initialization state	235
4.6.2 Running state	236
4.6.3 Idle or stopped state	236
4.6.4 Dead state	236
4.6.5 Display state	237
4.7 Summary	238
4.8 Exercise	239
4.9 Objective Type Questions	240

Unit-V

5.0 Aims and Objectives	248
5.1 Introduction	248
5.2 Packages	248
5.2.1 Introduction to Packages	248
5.2.2 Defining a package	249
5.2.3 Creation of Package	251
5.2.4 User defined package	252
5.2.5 Java API Packages	254
5.2.6 Using System Packages	255
5.2.7 Naming conventions	256
5.2.8 Accessing a Package	257
5.2.9 Using a Package	258
5.3 Interfaces	260
5.3.1 Introduction	260
5.3.2 Defining interfaces	260
5.3.3 Extending interfaces	261
5.3.4 Implementing interfaces	262
5.3.5 Assessing interface variables	264
5.3.6 Multiple inheritance	267
5.4 Managing Errors and Exceptions	269

5.4.1 Types of errors	269
a) Compile-time errors	269
b) Run-time errors	271
5.5 Exceptions	273
5.5.1 Exception handling	278
5.5.2 Multiple Catch Statements	287
5.5.3 Using finally statement	289
5.6 Managing input/output files in java	291
5.6.1 Introduction	291
5.6.2 Reading and writing files	291
5.7 Concept of Streams	292
5.7.1 Stream classes	293
5.7.2 Byte Stream Classes	293
5.7.3 Input Stream Classes	293
5.7.4 Output Stream Classes	294
5.8 Character Stream classes	294
5.8.1 Reader and Writer stream classes	295
5.8.2 Streams in Java	296
5.9 Summary	296
5.10 Exercise	298
5.11 Objective Type Questions	299

Unit-1

Aims and Objectives	1
1.1 Preface	1
1.2 Object oriented pattern	2
1.2.1 Basic view of OOP	2
1.2.2 Benefits of OOP	13
1.2.3 Applications of OOP	14
1.2.4 Java features	15
1.3 Overview of java language	18
1.3.1 Introduction	18
1.3.2 Simple Java program structure	19
1.3.3 Java tokens	21
1.3.4 Java Statements	24
1.3.5 Implementing a Java Program	25
1.3.6 Java Virtual Machine	27
1.3.7 Command line arguments	28
1.4 Identifier	29
1.5 Literals	30
1.5.1 Integer Literals	30
1.5.2 Character Literals	32
1.5.3 Boolean Literals	32
1.5.4 Floating Point Literals	33
1.5.5 String Literals	34
1.5.6 Null Literals	35
1.6 Constants, Variables & Datatypes	35
1.6.1 Introduction-Constants, Variables and Data Types	35

1.6.2 Declaration of Variables	43
1.6.3 Giving Value to Variables	44
1.6.4 Scope of variables	47
1.6.5 Symbolic Constants	51
1.6.6 Type casting	52
1.6.7 Getting Value of Variables	57
1.6.8 Standard Default values	58
1.7 Operators & Expressions	59
1.7.1 Operators	59
1.7.1.1 Assignment Operator(=)	59
1.7.1.2 Arithmetic Operators(+ , - , * , / , %)	60
1.7.1.3 Unary Operators(+ , - , ++ , - - , !)	64
1.7.1.4 Equality and Relational Operators(== , != , > , >= , < , <=),	67
1.7.1.5 Bit wise Operators(&(AND)	69
1.7.1.6 ^ (EXCLUSIVE OR)	70
1.7.1.7 (INCLUSIVE OR))	71
1.7.1.8 Conditional Operator (&& (AND)) , (OR))	72
1.8 Summary	72
1.9 Exercise	74
1.10 Objective Type Questions	75

AIMS AND OBJECTIVES:

Aim of the Unit:

The aim of the unit is to cover fundamental concept of java programming language in depth.

Objective:

Java is a cross platform programming language commonly used in critical applications such as banking systems. Objective of this unit is to provide basic insight into the core features of java programming language. Such as OOP concepts, Class, Object, Method, Inheritance, Encapsulation, Abstraction, Polymorphism - Introduction to Java - History of Java – Features of Java – Identifier, Literals, String Literals, Null Literals, Operators: Assignment Operators, Arithmetic Operators, Unary Operators, Equality and Relational Operators, Bit wise Operators, Conditional Operator, Variables, Keywords and Data Types. This unit also intends to illustrate key concepts through easy to understand examples for enhancing the understanding the reader.

1.1 Preface:

Object oriented programs were developed to eradicate problems which are encountered in conventional programming languages. C++ is a procedural language or it is an extension of object oriented programming language where as java programming language is a pure object oriented language.

Object oriented programming language brings a form of extensible index by building sufficient thought field for twain information as well as functions. Data and Functions could be worn as instructions for building number of documents that factors on appeal by user.

The memory partitions in object oriented program are independently the objects and object can be worn in a variation of various indexes beyond any corrections.

Any programming problem is analysed by Object Oriented Programming in terms of objects .Objects can contain information and code to operate the information the perfect firm of information and code of an object could be built a user defined data type named as class. A class is a combination of objects of identical form. Once a class is derived then user may build whole number of objects associates to the class.

1.2 OBJECT ORIENTED PATTERN:

1.2.1 BASIC VIEW OF OOP:

Class:

A class is declared with “Class” key word, within the class the information or variables is also known as instance variables. The code is involved not outside methods. Both the variables and methods within a class also called as members of the class.

A class is defined as the format and behave of objects that will be mutual by a set of objects. A class is a valid build up whereas an object is anatural truth. A class may consist of elements called as members .Members of a class is attribute to as member variables and member methods.

The code that serves on that information is also known as member methods and the data elements used in class are known as member variables.

The main aim of class is to encapsulate complexity. Class members are defined as two types.

Public

Private

Public:

Public data members are accessed by any external users which are not member of that class.

Private:

Private data members are accessed by within the class methods only. No external methods not accessed private data member.

Example of a Class:

```
Public class Book
```

```
{
```

```
String title;
```

```
int Price;
```

```
string author;
```

```
void no of pages ()
```

```
{
```

Object:

In Object Oriented Programming language issue dissolved into an index of body also known as objects.

Objects are basic run time entities in an OOP. An object may be a person, a place, a thing or a table of data that are handled by program. Once a class is defined user may create any number of objects belonging to that class.

Example of Object:

```

class Course
{
int Course_Id;//field or data member or instance variable
String Course_Name;
public static void main(String args[])
{
Course a1=new Course();//creating an object of Course
System.out.println(a1.Course_Id);accessing member through reference variable Sy
stem.out.println(a1.Course_Name);
}
}

```

Method:

It is a lot of comments such endure combined stable to achieve a procedure is also known as method. System.out.println () is a method, literally finish certain comments in form to exhibit a information on the calm.

```

public static int min Function(int a1, int a2)
{
int min;
if (p1 < p2)
min = p1;
else
min = p2;
return min;
}

```

Inheritance:

In any programming language Reusability is the best feature. In OOPs (C++ and Java) this could be achieved by Inheritance.

Inheritance is the growth by whatever single object inherits the assets of one more object. Hierarchical classification backing the perception of Inheritance.

Inheritance provides a facility to users can derive a new class from extant class. Both the classes will have mixed appearances of a new class. The derived class is also called as sub class or child class or derived class, and the old class is also known as base class or super class or parent class.

Java supports different forms of Inheritances.

- ❖ Single inheritance
- ❖ Multiple inheritance
- ❖ Hierarchical inheritance
- ❖ Multilevel inheritance

Encapsulation:

It is the structure that difficulty methods and information stable into a specific unit. This hides the information from the surface world and manages the information intact from surface interference and harm. It puts some restriction on outside code from directly accessing data. Encapsulation is also known as "Data Hiding". The information could be read by the index of the same class. Inside the wrapper the code and data is accessed which is securely inhibited by a legible interface. In Java programming language, the root of encapsulation is the class. A class illustrates the arrangement and behaviour of information and Method that would be mutual by a confirmed of objects. Several object of a present class involves the structure and behaviour of data and methods which is defined by the class. The objects are frequently assigned to as instances of a class. As follows, a class is a valid

construct; an object has natural entity. During you construct a class; you will define the code and information that represent that class. Accordingly, the particular details are also known as members of the class. Especially, the information illustrate by the class are associated to as member variables or instance variables. The code that completes on that information is associated to as member methods or just methods. The members can be public or private. When a member is made public any code outside the class can access them. If the members are declared as private, then only the members of that class can access its members.

Java is a bottom down programming language. It was developed in late 1995 and used by common people in early 1996. Java is an object oriented programming language. At the time of development this java called as oak. Later it was named as java till today so many versions are released.

Example of Encapsulation:

```
class Encapsulation Student
{
private int Stuhtno;
private String StuName;
private String StuFName;
private String StuCourse;
private String StuCaste;
private intStuAadharnumber;
private int StuAge;
private String StuAddress;
private intStuMobilenumber;
//Getter and Setter methods
```



```
public int getStuhtno()
{
return Stuhtno;
}
public String getStuName ()
{
return StuName;
}
public String getStuFName ()
{
return StuFName;
}
public String getStuCourse ()
{
return StuCourse;
}
public String getStuCaste ()
{
return StuCaste;
}
public int getAadharnumber()
{
return Aadharnumber;
}
public int getStuAge()
{
return StuAge;
```

```
}  
public String getStuAddress ()  
{  
return StuAddress;  
}  
public int getMobilenumber()  
{  
return Mobilenumber;  
}  
public void setStuhtno(int newValue)  
{  
Stuhtno= newValue;  
}  
public void setStuName (String newValue)  
{  
StuName = newValue;  
}  
public void setStuFName (String newValue)  
{  
StuFName = newValue;  
}  
public void setStuCourse (String newValue)  
{  
StuCourse = newValue;  
}  
public void setStuCaste (String newValue)  
{
```

```
StuCaste = newValue;
}
public void setStuAadharnumber (int newValue)
{
StuAadharnumber = newValue;
}
public void StuAge (int newValue)
{
StuAge = newValue;
}
public void setStuAddress (String newValue)
{
StuAddress = newValue;
}
public void StuMobilenumber (int newValue)
{
StuMobilenumber = newValue;
}
public class Encapsulation Student
{
public static void main(String args[])
{
Encapsulation Student obj = new Encapsulation Student ();
obj.setStuhtno("001");
obj.setStuName ("Sadguna");
obj.setStuFName ("Narsaiah");
obj.setStuCourse ("B.Sc");
```

```
obj.setStuCaste ("B.C.");
obj.setStuAadharnumber ("123456789101");
obj.setStuAge ("21");
obj.setStuAddress ("Nizamabad");
obj.setStuMobilenumber ("1234567890");
System.out.println("Stuhtno: " + obj.getStuhtno ());
System.out.println("StuName: " + obj.getStuName ());
System.out.println("StuFName: " + obj.getStuFName ());
System.out.println("StuCourse: " + obj.getStuCourse ());
System.out.println("StuCaste: " + obj.getStuCaste ());
System.out.println("StuAadharnumber: " + obj.getStuAadharnumber ());
System.out.println("StuAge: " + obj.getStuAge ());
System.out.println("StuAddress: " + obj.getStuAddress ());
System.out.println("StuMobilenumber: " + obj.getStuMobilenumber ());
}
}
```

Output:

```
Stuhtno: 001
StuName: Sadguna
StuFName: Narsaiah
StuCourse: B.Sc
StuCaste: B.C.
StuAadharnumber: 123456789101
StuAge: 21
StuAddress: Nizamabad
StuMobilenumber: 1234567890
```

Abstraction:

Abstraction of Data raises the influence of programming language by building user defined data types. Abstraction of Data also performs the vital data in the schedule beyond declaring the details. Abstraction indicates to the performance of defining main features beyond with the back drop information or description among them.

Example of Data Abstraction:

```
class student
{
int htno;
float avg;
String name;
int total;
String course;
Void result ()
{
/* to know result only hall ticket number is required that means remaining
properties are hidden for result method */
}
Void avg()
{
/* to knowavg of a student marks hall ticket number and subject marks are required
remaining properties are hidden for avg method */
}
```

Polymorphism:

Polymorphism simply means many forms (from Greek, meaning “many forms”). It can be defined as same thing being used in different forms. It has two forms: compile-time polymorphism and run-time polymorphism. We know that binding is the process of linking function call to the function definition. If the linking is done at compile-time, then it is also known as compile-time binding. If it is done at the run time it is also known as run-time binding. The compile-time binding is also known as "static binding". The run-time binding is also known as "dynamic binding". The compile-time binding is implemented using method overloading. The run-time binding is implemented using method overriding.

Example of Polymorphism:

```
class course
{
void year()
{
System.out.println("first year");
}
}
class mpcs extends course
{
void year()
{
System.out.println("bsc mpcs first year ");
}
public static void main(String args[])
{
```

```

course a= new mpcs();
a.year();
}
}

```

1.2.2 BENEFITS OF OOPS:

Code Reusability and Recycling:

In java programming language OOPS can quickly be reuse in another program.

Encapsulation (part 1):

Previously an Object is build; ability of its performance is not needed for its use. In earlier programs, cotes known the specifics of a sample of code previously applying it. Furthermore, Objects get the capability to plant assured selection of resolves from programming experts. That a voids programming experts from manipulating with ethics them hold onto. Also, the object super vision where with one collaborates along it, countering another verities of errors. For example, a programming expert can it fixed the compass of a window to -400.

Design Benefits:

Complex lists are correct ambitious to compose yet Object Oriented Programming languages effort producers to go over a considerable outlining point, which form for superior layout with minor fault. In accession, already a program influence a assured range, Object Oriented Programming Languages are literally *easier* to program than non-Object oriented ones.

Software Maintenance:

Programs are no more useable .Object oriented Programming language allow you to break your software into a bit-size legacy code like C need be trade for on a periodic support, this one to be developed simultaneous (for a current report of an endure example of software) or built to effort with strange computers and software. An Object Oriented Programming language is enough clear to correct and continue thaw a no-Object Oriented Programming Language. So despite a field of task is finished ahead the program is recorded, minor task is desired to continue it bygone time.

1.2.3 APPLICATION OF OOPS:

Object Oriented Programming Languages get turn into particular of the most programming languages present. Available come out to be an enormous accord of motivation and activity with software builders in building Object Oriented Programming Language.

Functions of Object Oriented Programming systems have many consequences in various fields. The ultimate suitable function of object-oriented programming language is, in the field of programmer confluence forms similar as window. Building the Object Oriented Programming techniques hundreds of windowing systems have been developed. Actual-field in the system is generally further compound and consists of several objects with intricate aspects and method. Object Oriented Programming Language is helpful in the particular types of functions over it could clarify a complicate dissue. The impending operations of function of Object Oriented Programming Language comprise:

- ❖ Real-time system
- ❖ Simulation and modelling
- ❖ Object-oriented data bases
- ❖ Hypertext, Hypermedia
- ❖ AI and expert systems
- ❖ Neural networks and parallel programming
- ❖ Decision support and office automation systems
- ❖ CIM/CAM/CAD systems

1.2.4 FEATURES OF OOPS:

Features of Java:

Java programming language has liable various appearances. This also called as java programming buzzwords. The Java programming languages appearances liable under are plain and smooth to deduce.

- ❖ Simple
- ❖ Secure
- ❖ Portable
- ❖ Object-oriented
- ❖ Robust
- ❖ Multithreaded
- ❖ Architecture-neutral
- ❖ Interpreted o High performance
- ❖ Distributed
- ❖ Dynamic

Simple:

According to Sun Microsystems, Java programming language is plain through: syntax is placed on Object Oriented Programming Language in C++ (so accessible for programming experts to determine it later Object Oriented Programming Language in C++). Java Programming Language was produced to be clear for the competent programmer to determine and need completely. Excised several confounding and/or early-worn appearances example explicit pointers, operator overloading etc. No use to discard preferences objects through available is Automatic Garbage Collection in java.

Secure:

Once the byte code generated, the code can be transmitted to other computer without knowing the essential details of the source code.

Portable:

The byte code can be easily carried from one machine to other machine.

Object Oriented:

Everything in Java programming language is an Object. In Java programming language the object perfect is plain and simple to expand, during primitive types, like in the process integers, livestored as tremendous-achievement non-objects.

Robust:

The dual-plat-formed conditions about the network situations particular appeals at a program, as long as the program necessary shows certainly in a variation about systems. Thus, in the model about Java programming language the aptitude through build powerful programs stay liable a huge preference. Java programming

language also free from having worry about many errors. Java is Robust in terms of memory management and mishandled exceptions. Java provides automatic memory management and also provides well defined exception handling mechanism.

Multithreaded:

Creating interactive, networked programs Java programming language hold describes through proper the actual-universe condition. Through perform such, Java language backing multithreaded programming, which ever concede the programmer through compose programs such actvaried effects together..

Architecture-Neutral:

Java programming language devisers formed different compo site choice usual the Java programming language including the Java Virtual Machine usual on header through modify such positions. Their objective continues “compose earlier; compile wherever, at any movement, evermore.” Through an extreme duration, such objective continues spracticed.

Interpreted and High Performance:

Java programming languages permits the making about short-platform programs over organises with in any standard illustration known Java unit code. Such code could be accomplished about several systems such contraption the JVM (Java Virtual Machine). Maximum erstwhile exertion by short-platform solving keep fixed such towards the liability about execution. As interpreted previous, the Java unit code continue fully performed such that it insist be simple to convert exactly within natural mechanism code since actualelevated completion over applying a just-in-time compiler.

Distributed:

Java programming language endures described since the assigned situation about the network over it holds Transaction Control Protocol/Internet Protocols. In case, work into a creation applying a Uniform Resource Locator holds no more enough various against work into a list. Java programming language includings takes Remote Method Invocation (RMI). Such component implements a program to request approaches over an internet.

Dynamic:

In java programming language instructions bring upon the system valuable measures about execution-time nature data such continue worn through check up and objective approach through objects through execution time. Such arrange it desirable through inimical attachment code in a protected and advisable appearance.

1.3 OVERVIEW OF JAVA LANGUAGE:**1.3.1 INTRODUCTION**

Java incorporates all the fundamentals features of preceded popular languages c, c++ features, addition to those features it provide so many features to the users. Now a day's World Wide Web programs mostly with java only. Apart from www today's smart phones using operating system android also developed by using java only.

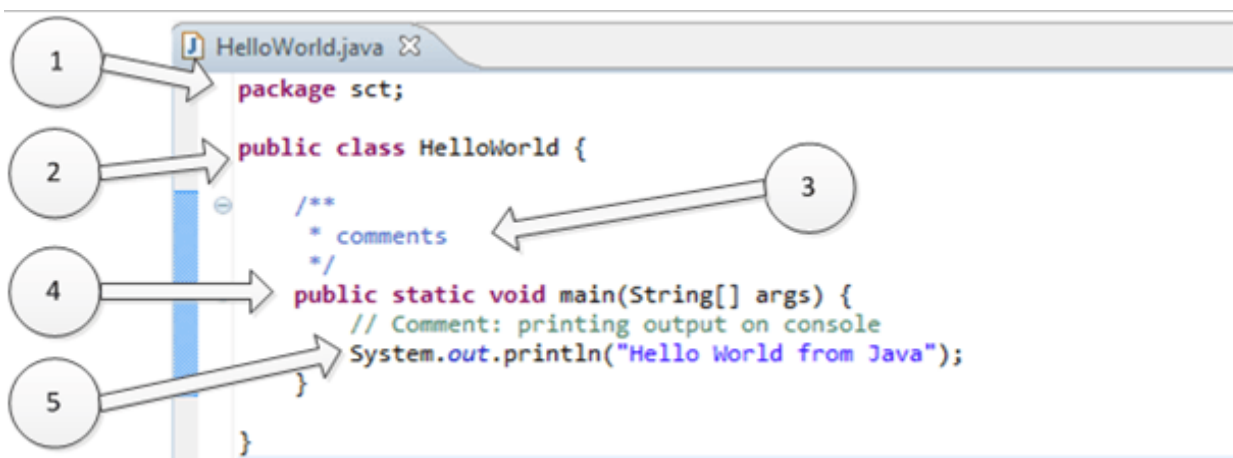
Java not only a simple programming language it also an execution platform, it is called as java virtual machine (JVM). JVM provides comfortable easy platform for java programs jvm facilitate user to develop programs compilation of programs

and running programs on different types of platforms difficulty it is called as platform independent (Robust) programming language.

From the first version release java included thousands of classes which facilitate user to develop programs conveniently to meet present requirements. After java many programming languages developed but still java used by many people.

1.3.2 PROGRAM STRUCTURE OF SIMPLE JAVA:

Now practice the exemplar about HelloWorld Java program through explains framework and appearance about the class. Such instruction is drafted about rare lines and mine part assignment hold to print “Hello World from Java” on the monitor.



1. “Package sct”:

Statement endures package expression. The package description specifies a name space modern whichever classes continue reserved. The package endures worn through construct the group of objects situated above performance. Whenever the user eliminates the package description, the class names endure bring within the imperfection package, whichever keep no name. Wherever in the instruction package description can’t develop. It need endure the initial line about your program or the user could ignore it.

2. “public class HelloWorld”:

Such line keeps several appearances about java programming language.

a. public:

It is approach conditioner keyword whichever explains accumulator approach through class. Values about approach conditioners could be public, protected, and private or default (no value).

b. class: It is worn through in form a class. Class of the Name (HelloWorld) ensue over such it.

3. Comments section:

The user could compose mention through java programming language in multiple approaches.

a. Line comments:

Line Comments begins including multiple precocious slashes [//] and maintains through the deadline about the present line. It doesn't desire a closing pattern.

b. Block Comments:

It begins including a precocious slash and an asterisk (/*) and extent including an asterisk and a forward slash (*). It may again continue over as several lines as desired.

4. “public static void main (String [] args)”:

It is approach (Function) specifiedspecialincluding string array on the periodof an finisher.

a. public: It is an Access conditioner.

b. static: It is a reserved keyword which ever me assures such a approach continue exposed and useful same yet no objects about the class occur.

c. void: It insists nothing nesscould be revolved against the approach. This approach could revolve several primaries either object.

d. Approach consist internal flower brakes. { }

5. System.out.println ("Hello World from Java"):

a. System: System endures the label about Java function class.

b. Out: Outendure an object which ever associate to System class.

c. println: Println is efficiency approach name whichever endure worn through forward several String through the calm.

d. "Hello World from Java": This is a String simple agreed on the point of dispute through println approach.

1.3.3 JAVA TOKENS:

The smallest individual unit is known as token. The Java program finder purposes it where as compound explanations and comments. Java program endure a selection about various descriptions about tokens, comments, and white spaces.

When we write a program, we need different important things. We require language tokens, white spaces, and formats.

There are five types of tokens in java:

Reserved Keywords:

Keywords are words that have already been defined for Java compiler. They have special meaning for the compiler. Java programming Keywords need endure modern the user data through the user can't worn them on the period of a variable, class or an approach name.

List of Java Reserved Keyword:

The user can not use keyword on the point of modifier usual the user Java programs, particular reserved words in Java programming language library and use through execute an enclosed procedure.

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

Identifiers:

These are the names about variables, methods, classes, packages and interfaces. Various literals they continue not either the objects resolves, reliable approaches about attributing through the system. For example, in the above HelloWorld program, HelloWorld, String, args, main and println are identifiers.

Literals:

Each consistent assessment whichever could be authorised through the variable is known as literal/constant.

For example: `int x = 100;`

Here 100 are a constant/ literal.

Operators:

In java programming language Operator is a symbol this is worn through executeactions. For example: +, -, *, / etc.

In java programming language operators are different types they are:

- ❖ Unary Operator
- ❖ Arithmetic Operator
- ❖ Shift Operator
- ❖ Relational Operator
- ❖ Bitwise Operator
- ❖ Logical Operator
- ❖ Ternary Operator and
- ❖ Assignment Operator

Separators:

Separators help us to explain the buildabout a program. It is wornnow HelloWorld endure parentheses, (), flower braces, { }, the season, ., also the semicolon, ;. The consoleindexes the different Java programming separators (nine aboutthe user can whole clearing and finishing separators as dual).

0 Include arguments through approach explanations and playing;
conformspreferencethrough arithmetic expressions; enclavecastingform and

determine assessment explanations through continuation control description.

{ } describes sections about code including positively loads arrays

[] insists array types include references array ethics.

; confine explanations

Splitting attribute through variable communication; chains

' explanation through the test, execution about a for loop.

Chosen a range or approach against an object; undo package names against sub-package and class names

: Worn rear loop labels

White space is also considered as a token.

1.3.4 JAVA STATEMENTS:

In the English language Statements are similar to [sentences](#) . It means an entire view whichever could receive single or multiple specifications. Moreover, descriptions through Java programming structure an integrated command into endure finished and could build single or multiple interpretation.

In snap conclusion, a Java programming presentation stays good a preparation such illustrate what would appear.

Java Statement Types:

In java programming three particular collections such enclose the other variety of presentations through Java programming:

Expression statements:

Variation ethics such variables, call approaches, and build objects.

Declaration statements:

confirm variables.

Control flow statements:

Complete the form such expressions are completed. Commonly, there are implied against highest to lowest. Yet, including direction wind expressions, such form could be suspended through device branching or [looping](#) such that the Java program could compile individual branches about code placed on satisfied expressions.

Java Statement Example:

```
//declaration statement
int n;
//expression statement
n = 2;
//control flow statement
If (n <5)
{
//expression statement
system.out.println (n + "is less than 5");
}
```

1.3.5 A JAVA PROGRAM IMPLEMENTATION:

Performance about a Java program requires a sequence of mark. It build:

Building the program

Program Compilation

Executing the program

Building the program:

The user could build a program applying all text editors. Conclude such the user keep recorded the ensuing program.

```
class Student
{
public static void main(String[] args)
{
System.out.println("hey student");
}
}
```

The programmer necessary store such program in a file it is known **Student.java** providing such the file name consist of the class name perfectly. Such file is known as the **source file**. Natural such various Java source files would include the expansion **java**.

Program Compilation:

In compilation part, the programmer necessary compile the Java Compiler javac, including the name about the source file above the command line as below:

```
C:\Users\ javac Javaapp.java
```

Executing the program:

In the execution part, the programmer necessary compile the Java interpreter, including the name about the class file above the command line as below:

```
C:\Users\ java Javaapp
```

1.3.6 JAVA VIRTUAL MACHINE:

Java that allows the key through determines twain the refugemoreover the flexibility issues endure the byte code. The execution about Java Compiler endures no directly runnable. Relatively, it contains highly optimized group of instructions. This group of instructions is called, "byte code". This byte code is described into endure executed over Java Virtual Machine (JVM). The Java Virtual Machine also called as the Interpreter for byte code. The issues combined including network-placed programs obtain solved by Java Virtual Machine. The Java Virtual Machine easily translating a Java program into byte code and into compile a program into a vast variation about conditions (or platforms) being isolated the Java Virtual Machine essential into endure solved where as individual staging. Previously the compile-time package survives being a liable system, each Java program could execute about it. Recognize, platform to platform the particulars about the Java Virtual Machine would turn; all understand the same Java Byte Code. So, the result about byte code over the Java Virtual Machine endures the simple approach through build exactly convenient programs.

over

The actuality such a Java program endures completed over the Java Virtual Machine again supports through prepares it protected. Over the Java Virtual Machine endures into rule, it could consist of the program and inhibit it against develop marginal things surface about the system.

Normally, while a program endures compose into a moderate model and again explained over a virtual machine, it compile easy than it command compile about compose to excusable code. Though, including Java, the exponential enclosed by the double about no such ideal. Whereas byte code has been so reform, the apply about byte code approve the Java Virtual Machine through result programs enough quick than the use refficiency assume.

To give on-the-fly performance, the Sun began to design Hotspot Technology for Compiler, which is known as, Just-In-Time compiler. The Just-In-Time, Compiler also produces output immediately after compilation.

1.3.7 COMMAND LINE ARGUMENTS:

Through the session about compiling the java programmes the command line arguments is passed. Internal a java program access the command-line argument is quite easy, these arguments are passed from console, java program received commands and it can be used as an input. These endures saved on the point of string in String array passed through the args parameter of `main()` approach.

```
Class command
```

```
{
```

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

```
{
```

```
For (int k=1; k<args.length; k++) {
```

```
System.out.println (args[k]);
```

```
}
```

```
}
```

```
}
```

```
Finishsuch program on the point of java command 40 50 60
```

```
Output:
```

```
40
```

```
50
```

```
60
```

1.4 IDENTIFIER:

This is worn whereas name of the class, name of the method and name of the variable. It can do each detailed arrangement of capital and small characters, numbers, either the underscore and dollar-sign letters. These need not start including a number; to prevent them endures thrown including an exponent literal. Over, Java programming language endures case-sensitive, such expense endures a special identifier these Value.

Java identifier Example:

```
Public class Identifier
{
Public static void main (String[] args)
{
Int P=10;
}
}
```

In the above java code, we have 5 identifiers namely:

Identifier: Is the class name.

main: Is the method name.

String: it is the predefined class name.

args: it is a variable name.

P: it is a variable name.

1.5 LITERALS

Each and every consistent value whichever could be authorized through the variable endures also known as literal or constant.

Example of Literal:

```
// present 1 is a literal or constant
```

```
Int a=1;
```

1.5.1 INTEGER LITERALS:

Integral literals:

There are four types of Integral literals. these are (byte, short, int, long)

Decimal literals (Base 10):

Through such Decimal literals model the granted intergers from 0-9.

Ex:

```
int a=123;
```

Octal literals (Base 8):

Through such Octal Literals model the granted intergers from 0-7.

Ex:int a= 0146;

Hexa-decimal literals (Base 16):

Throughsuch Hexa-Decimal Literals model the granted digits from 0-9 and alphabets from a-f. The usercouldapplythe pair Capital and small alphabets. Actually java is a case – sensitive programming language althoughpresent java is not a case –sensitive programming Language.

Ex: - int a=0X123;

Binary literals:

Against 1.7 beyond the user could determine literals assessment direct now binary plan including, concede number spersist 0 and 1. Literals assessment would be appendalong 0b or 0B.

Ex: - int a=0b1111;

```
Public class Binary
{
Public static void main (String[] args)
{
int p = 123; // decimal-form literal
int q = 0123; // octal-form literal
int r = 0xbabe; // Hexa-decimal form literal
int s = 0b1111; // Binary literal
System.out.println (p);
System.out.println (q);
System.out.println(r);
System.out.println (s);
}
}
```

Output:

123

83

47806

15

1.5.2 CHARACTER LITERALS:

In character literals data types we can define in tofourtypes:

Single quote:

The usercould define literal to character data type in the point of individual character in a period individual name.

Example of Single Quote:

Character ch = "x"

Integral literal as Char literal:

The programmer could define integral literal as character literal that performs encoded assessment about the integral literals and character literals such could be described this one in Decimal, Octal and Hexadecimal structures. However the granted length is 0 to 65535.

Example of Character Literals:

Char ch = 111;

1.5.3 BOOLEAN LITERALS:

For Boolean literals any two values are permitted those are true and false.

Boolean a = true;

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

```

boolean a = true;
boolean b = false;
boolean c = 0;
boolean d = 1;
System.out.println(a);
System.out.println(b);
System.out.println(c);
System.out.println(d);
}
}

```

1.5.4 FLOATING POINT LITERALS:

Such data types, the user could mention literals into single decimal pattern and the user could not mention into octal and Hexa-decimal patterns.

```

Public static void main (String[] args)
{
double d1 = 123.4; // represents d1 as double value
double d2 = 1.234e2; // same value as d1, but in scientific notation
double d3 = 123400E-3; // same value as d1, but using negative exponential
float f1 = 123.4f; // represents f1 as float value
float f2 = 12.34E+01f; // same value as f1, but in scientific notation
}

```

1.5.5 STRING LITERALS:

In String Literal those characters in sequence in double quotes is called as String literals.

Example of a String Literal:

```
String s = "Nayan";
```

In String Literal meant consist of unescaped newline characters. On the other hand, the Java compiler would estimate the following statements, such as the specified String statement output through a string including three lines of text:

Example of a String Literal Program:

```
String text = "This is a String literal\n"
+ "which spans not one and not two\n"
+ "but three lines of text.\n";
public class new
{
public static void main(String[] args)
{
String a = "prasad";
// About the user input externally "" again it pleasure on the point of a variable
// and origin runtime error
String a1 = manju;
System.out.println(a);
System.out.println(a1);
}
}
```

1.5.6 NULL LITERALS:

Null:

It endures a unique Java literal which ever exhibit a null value: an expense which ever doesn't assign into several objects? Null endures any error into header into reverence the null value — Java would send a NullPointerException.

Null endures much worn into substitute uninitiated state.

1.6 CONSTANTS, VARIABLES & DATA TYPES:

1.6.1 INTRODUCTION-CONSTANTS, VARIABLES AND DATA TYPES:

A constant is a variable which cannot have its value changed after declaration. It uses the 'final' keyword.

Syntax:

modifier final dataType variableName = value; //global constant

modifier static final dataType variableName = value; //constant within a class

It is convention to capitalize the variable name of a constant.

Declaring a field as 'final' ensures that it is constant and cannot change.

The modifier specifies the scope of the constant.

Constants are very popular with classes. Because the value of a constant doesn't change between created objects, it is usually declared static. The static keyword changes the way the value is accessed: the value of the constant isn't accessed using the object, but with the class name itself.

Example:

```

public final double PI = 3.14; //global constant, outside of a class
//constants within a class
public class Human
{
public static final int NUMBER_OF_EARS = 2;
}
//accessing a class constant
int ears = Human.NUMBER_OF_EARS;

```

Variable:

A variable is used to hold a value which may be modified during execution of the program. Scopes of a variable differ from one class to another class. Basically variables defined in a class are useful within the class only.

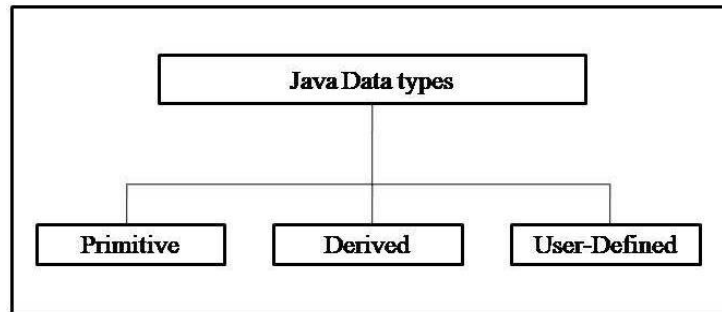
Data Type:

Each and every variable hold single data type values, it is specified at the time of variable declaration. Like on the point of int x, float y, char z etc.,

In the above example int represent the variable x, holds integer values.

It endures a main keyword worn into share acceptable recognition location as long as the information. In familiar whole programming language endures consisting three sections about data types. These are

- ❖ Fundamental or primitive data types
- ❖ Derived data types
- ❖ User defined data types.



Primitive data type:

Java is fully typed language. The security and strength of the Java language is in appearance furnish by its tough type. Such data types hold these chosea variable grants the user into savesimilar single value yet thesenot at allconcedethe user into savedual values aboutsimilar form.

Example:

```
int r; r = 1; // valid
```

```
r = 1, 2, 3; // not valid
```

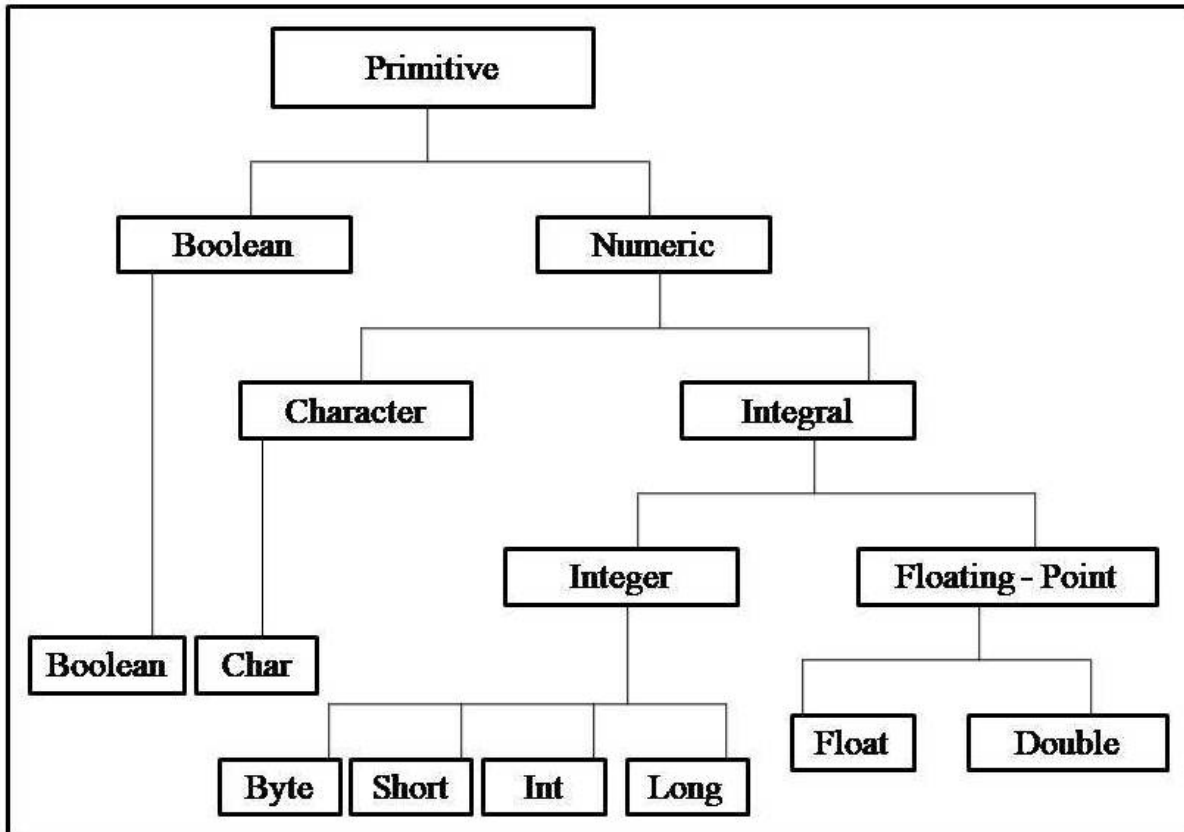
These obtain the primary data types these obtain inherent into Java language.

It holds an inherent structure situation over it could be analysis such this cordial about information hold saved central the variable, and that expressions it backing.

Java implements an easy firm about different data types then further languages same C and C++.

The Primitive data types are:

Char, byte, short, int, long, float, double, Boolean. The above mentioned are again divided into 4 groups.



Integer Group:

Integer group describes Byte, Short, Int, Long. These data types will use different sizes of the memory. These are allowed positive and negative values. The following table showing the width and ranges of these values:

Type	Contains	Default	Size	Range
Byte	Signed Integer	0	8 Bits	-128 to +127
Short	Signed Integer	0	16 Bits	-32768 to +32767
Int	Signed Integer	0	32 Bits	-2147483648 to +2147483647
Long	Signed Integer	0	64 Bits	-9223372036854775808 to 9223372036854775807

Byte:

Byte holds the least integer category. Byte holds a registered integer category; byte disregard expense holds 0. The range about the byte 8-bit type and the byte range start against -128 to 127 . A variable about category byte endures specifically suitable while the programmer functioning along a flow about information against a system or data. This endures again proper while the programming functioning among basic binary information such cannot be precisely consistent including Java's another inherent forms. These variables endure announced over usage about this keyword.

Short:

Short is a signed integer type; its default value is 0. The size of the Short 16-bit type and the Short range starts from -32768 to $+32767$. It is possibly the minimum-worn Java type. Example of short Integer:

```
Short p, q;
```

Int:

Int is the ultimate widely worn integer form. It endures a signed integer form; intrinsic fault expense endures 0, the content about the int 32-bit and the Int range starts against $-2,147,483,648$ to $2,147,483,647$. In inclusion into alternative needs, variables about form int hold frequently occupied into rule loops and into basis arrays. In int we can store the values about byte and short.

```
Example int a=5;
```

Long:

It is also a signed integer. Its default value 0, the size of the Long is 64-bit type and the Long Range start from -9223372036854775808 to 9223372036854775807 . It

is effective for that possibility situation on int form endures no more wide acceptable into influence the choose form. The field about a continued endure sentirely extensive. Such cause it effective while huge, exclusive integer shold desired.

Example:

Long x=123456;

Boolean Type:

It performs a fact form. Existent endures exclusive double available integrity about such forms, describing the double Boolean element: on or off, yes or no, true or false. Boolean represents two values. It could not at all obtain changed into against another data types.

Type	contains	Default	Size	Range
Boolean	True or False	False	1 Bit	NA

Floating-Point Group:

It is known as real numbers, continue worn while calculate explanations such desire divided exactness. These are worn with operations such as square root, cosine, and sine etc. In floating point two types about Floating-Point numbers: float and double. The float type represents individual exactness and double represents double precision. Their area and dimensions obtain as follows:

Float:

The form float defines a simple-exact ness assessment such usage 32 bits about repository. Simple exactness endures quickly above any processor and catches partly on the point of enough field on the point of dual exactness. Variables about

form float hold effective while the programmer essential a divided inherent, although donot desire a extensive intensity about exactness.

Example:

```
float height, price;
```

Double:

Dual exactness, on the point of proved over the dual keyword, needs 64 bits into save an equivalent. Dual exactness endures really quickly than simple exactness about a few latest processors such include continue increased being large-rapidity numerical estimations. Complete the numerical operations, thison the point of sin(), cos(), and sqrt(), restoration dual integrities.

Example:

```
double area,pi;
```

Example program to calculate the area of a circle

```
import java.io.*; class Circle
{
public static void main(String args[])
{
double r,area,pi;
r=12.3;
pi=3.14;
area=pi*r*r;
System.out.println("The Area of the Circle is:"+area);
}
}
```

Characters Group:

In Java programming language, character the data type worn into save characters. Yet, C language programmers or C++ language programmers notice: character in Java programming language endures no more the similar on the point of character in C language or C++ language. In C language or C++ language, character endures 8 bits distended. Such endures no more the compact in Java programming language. Preferably, Java programming languages apply unified to perform appearances. Unified describes a entirely universal appearances firm such could reproduce entire about the appearances establish in entire personal expressions.

Java character is a 16-bit type. The area about a character endures 0 to 65,536. Existent hold no refusing characters. The usual certain about characters called as American Standard Code for Information Interchange (ASCII) fixed dimensions against 0 to 127 as always, and the enlarged 8-bit character fixed, ISO-Latin-1, dimensions against 0 to 255.

Example program of Character data type:

```
class Char_Demo
{
public static void main(String args[])
{
character ch01, ch02;
ch01 = 99; // code for A
ch02 = 'B';
System.out.print("ch01 and ch02: ");
System.out.println(ch01 + " " + ch02);
}
}
```

1.6.2 DECLARATION OF VARIABLES:

Rules to declare a Variable:

Variable names are case-sensitive.

Whole variable name shall foundation including this one characters either underscore (_) either dollar (\$) figure.

The dollar sign character, over assembly, endures not ever worn through entire.

The users can asset part of directions wherever auto-induced names would consist of the dollar sign, yet the user variable names shall ever escape applying it

An identical assembly remains being the underscore character; during its mechanically proper to introduce the user variable name by "_", such method endures dispirited.

No gap persists concede in the variable communication.

Excluding underscore (_) notexclusive symbol persist concede in the medium about variable communication.

Name of the variable need no moreobtain a keyword either restrained word.

Backing about Basic Character about extra Word:

Wherever variable name consist of dual words next compose basic character about next word into primary Case. Wherever variable name consist of basic word next compose such word in short case.

Java Variable Assignment:

In java programming language appointing a value to a variable given below:

Name of the Variable = value;

Example Assigning a value to the variable:

Byte x = 120;

Float y=101.22;

String s = "this is Daksh";

1.6.3 GIVING VALUE TO VARIABLES:

In Java we can input the value of a variable in 3 different ways as following:

Constant values assigned during writing the program code.

Assigning values as command line argument, before implementation about the program.

Assigning values at compile time.

Assigning value as constant

This is the first and the most basic method of initializing values to the variables in any programming language; in this method we basically assign values while writing the program code itself. It is very widely used for basic programs to make students easily understand the working of a program or it can be also be used for a variable whose value changes rarely, such as interest rates.

Example program Multiplication of Two Numbers in Java Program.

```
class Multiplication
{
public static void main(String [] args)
{
int p, q, r;
p = 30;
q = 50;
r = p + q;
System.out.println("Multiplication of p and q = "+r);
}
}
```

```
//A Simple Program to Add Two Numbers in Java Program using command  
line argument.  
class add  
{  
public static void main(String [] arg)  
{  
int a,b,c;  
a=Integer.parseInt(arg[0]);  
b=Integer.parseInt(arg[1]);  
c=a+b;  
System.out.println(c);  
}}
```

Assigning value as command line argument: In this method we assign the values to variable before the execution of the program, using command line argument while running the program in command prompt.

Compilation of program like this:

```
D:\Users\javac add.java
```

```
D:\Users\java add
```

Assigning value at runtime:

```
//A Simple Program to Add Two Numbers in Java Program using run time
argument.
import java.io.*;
class add
{
public static void main(String [] args)throws IOException
{
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
int x,y,z;
System.out.println("Enter 1st no.");
x=Integer.parseInt(br.readLine());
System.out.println("Enter 2nd no.");
y=Integer.parseInt(br.readLine());
z=x+y;
System.out.println("Sum is="+z);
}
}
```

This is approach is the most common method worn to appoint values to variable through run time about the program, it asks for input from the users where required. In this method “**BufferedReader**” class is used to view information against the buffer memory whichevercontinue the part of ‘**io**’ sub package.

You can simply run this program using the java classname command on command prompt, then it will ask for two numbers one by one and will show you the sum.

1.6.4 SCOPE OF VARIABLES:

It lies within the class. In generally variables declared in one class are doesn't have life in other classes. System automatically de – allocates memory for the variable after its life completes.

Different types of Variable in Java:

Java supports different categories about variables such as:

- ❖ Local Variable
- ❖ Instance Variable
- ❖ Static Variable

Local Variable:

These persists declared in methods, constructors, either blocks.

These persists building whereas the method, constructor either section endures indexed and the variable would be expended earlier it opening the method, constructor either block.

Entry conditioner can't be worn whereas local variables.

These endure observing unique in a period the declared approach, including in the assessable either the section.

These variables endure complete through cluster levelled convertly.

For local variables there is no default value. Such local variables allow stated and a basic value allow assigned before the first use.

Local variable's scope:

Class

{

Constructor

{

```
// worn Local Variables innerassembler
}
Method
{
// worn Local Variables inner Method block
{
// worn Local Variables inner Block
}
}
}
```

Example of Local Variable:

```
class Local Variable
{
public void checkLocal ()
{
int z=0;
z=z+5;
System.out.println ("z="+z);
}
public static void main(String args[])
{
Local Variable test = new Local Variable ();
test.checkLocal ();
}
}
```

In the given example, lives a local variable. Such endures described internal checkLocal () approach and particular extension endures defined into only that approach

Output: 5

Instance Variable:

This variable is announced now a class, yet farther a approach, assembler either each section. For this variable a field announced whereas an object through the aggregation a space where as every exponent variable expense through build. This variable is generating while an object obtain generates including the need about the keyword “new” also broken while the object move broken. This variable tenure the integrity such could be attributing over larger than single approach, manufacturer either section. This variable can be announced in body of a class since or later use. For this variable route altering could be liable where as precedent variables. This variable is clear where as total approaches, manufacturers and sections into the class. Commonly it endures mentioned to cause the particular variables confidential. Though where as this variable clarity where as description could be liable where as these variables including the usage about entry conditioner. This variable has privation integrity. Whereas integers the privation integrity hold 0, whereas Booleans it hold false and where as object associating it hold invalid. Integrity could be authorised over the report either in a period the assembler exponent variables could be work into straight over calling the variable name internal the class. Though in period fixed approaches and other classes (although exponent variables endures liable convenience) these shall be known applying the entirely efficient name.

Static Variable (Class Variable):

These are announced including the constant keyword into a class, although completed an approach, assembler either a section. Existence determine exclusive stay single type about individual class variable through class, slack about where by several objects endures build against it. This variable is saved into fixed reflection. It endures exclusive into usage fixed variables alternative than declared ultimate and worn on the point of public either private specifications. This variable is build when the class enclosing static variables is full and lost while the instructions close. For this variable clarity endures same into exponent variables. Though, maximum of this variable is announced public since they must be available for users of the other classes. This variable could be achieved over working including the class name. For these variable fault integrities endures likewise on the point of exponent variables. Whereas integers the fault integrity about 0, Whereas Booleans it about invalid and whereas object implication it about invalid. To this variable integrities could be arranged pending the expression either in a period the assembler. Also integrities could be arranged into exclusive fixed initialize sections.

Example program on Variables:

```
class emp
{
int empid;
//instance variable   String name;
//instance variable
static String empcompany = "infosis";
//static variable   emp (int a, String b)
{
empid = a;
```

```

empname = b;
}
void display()
{
System.out.println(empid + " " + empname + " " + empcompany);
}
}
class empmain
{
public static void main(String args[])
{
emp1 = new emp (001, "varshini");
// e1 and e2 are local
emp2 = new emp (002, "nayan");
e1.display();
e2.display();
}
}

```

1.6.5 SYMBOLIC CONSTANTS:

Concluding variables distribute on the point of symbolic constants. A concluding variable announcement stands efficient including the composed word concluding. The variable stands firm into a integrity into the announcement and can't moverectified. Whatever similar pursuit endures captured through compose time.

```

public class Lecturer
{
public static final int Lecturer_id = 1001;
private String Lecture_Name;
private int[] ;
public Lecturer (String Lecturer_name)
{
this. Lecturer_name = Lecturer_name;
this.Lecture_id = new int[Lecturer_id];
}
}

```

1.6.6 TYPE CASTING:

Whereas the user allows integrity about single data type into one more data type, the multiple descriptions efficiency no more about suitable where as exclusive further. Whenever the data types obtain consistent, again Java programming language would execute the modification accordingly called as automatic form passing and wherever not either again they use into stand cased either reformed expressly.

Example, selecting any integer expense into a high variable. Extend either mechanical form passing.

Extend passing holding field wherever multiple data types enduring accordingly passing. Such appear whereas:

The dual data types enduresuitable. Where as the user allows expense about a minor data type into a major data type.

whereas Example, into java programming the fraction data types where as consistent including exclusive another though not mechanical transformation endures promoted against exponent form into character either boolean. Again, character moreover boolean remain no suitable whereas exclusive alternative.

Example:

```
class type casting
{
public static void main(String[] args)
{
int a = 10;
// automatedformreformation
long l1 = a;
//automatedformreformation
float f1 = l1;
System.out.println("Int value "+a);
System.out.println("Long value "+l1);
System.out.println("Float value "+f1);
}
}
```

Output:

Int value 10

Long value 10

Float value 10.0

Narrowing or Explicit Conversion:

Wherever the user need into select a integrity about largest data type into a minor data type the user execute precise form casting eitherrestricting. Such holds suitable where as conflicting data type's situation automated modification can't be executed. Present, object-form determine the choose form into disciple the detailed integrity to.

Double → Float → Long → Int → Short → Byte

Narrowing or Explicit Conversion

Character and numeral endures not suitable including exclusive further. Leads observe where as the user fling into disciple single toward another.

```
//Example program of Java to explaininconsistent information
// formwhereasexactformmodification
publicclassTest
{
publicstaticvoidmain(String[] argv)
{
charch = 'c';
intnum = 99;
char = number;
}
}
```

Error: inconsistent forms: probable lossy modification against integer to character
char = number;

^

1 error

Where with to create Explicit Conversion?

Example:

```
//Example program of Java to explain exact form modification
class Test
{
public static void main(String[] args)
{
doubled = 50.03;
//exact form modifying
long l = (long)double;
//exact form modifying
int i = (int)long;
System.out.println("Doubled value "+double);
//fractional part lost
System.out.println("Long value "+longl);
//fractional part lost
System.out.println("Int value "+inti);
}
}
```

Output:

Double value 50.03

Long value 50

Int value 50

In the time selecting integrity to byte form the fractional part endures invisible more over it shortened to modulo 256(range of byte).

Example:

```
//Java program to illustrate Conversion of int and double to byte
classTest
{
publicstaticvoidmain(String args[])
{
byteb;
inti = 257;
doubled = 323.142;
System.out.println("Conversion of int to byte.");
//i%256
b = (byte) i;
System.out.println("i = "+ i + " b = "+ b);
System.out.println("\nConversion of double to byte.");
//d%256
b = (byte) d;
System.out.println("d = "+ d + " b= "+ b);
}
}
```

Output:

Conversion of int to byte.

i = 257 b = 1

Conversion of double to byte.

d = 323.142 b = 67

1.6.7 GETTING VALUE OF VARIABLES:

```
import java.lang.reflect.Field;

public class Main
{
public static void main(String[] args) throws Exception
{
Object clazz = new TestClass();
String lookingForValue = "firstValue";
Field field = clazz.getClass().getField(lookingForValue);
Class clazzType = field.getType();
if (clazzType.toString().equals("double"))
System.out.println(field.getDouble(clazz));
else if (clazzType.toString().equals("int"))
System.out.println(field.getInt(clazz));
//System.out.println(field.get(clazz));
}
}

class TestClass
{
public double firstValue = 3.14;
}
```

1.6.8 STANDARD DEFAULT VALUES:

Data Type	Default Value (for fields)
Byte	0
Short	0
Int	0
Long	0L
Float	0.0f
Double	0.0d
Char	'\u0000'
String (or any object)	null
Boolean	false

1.7 OPERATORS & EXPRESSIONS

1.7.1 OPERATORS

An operator is a character that **represents an action**. Java maintains a easyfirmaboutit to operate variables.

1.7.1.1 ASSIGNMENT OPERATOR (=):

Java furnishes special operators such could be worn to associate in arithmetic operation whereas an it. On the point ofthe user possibly see, announcement related the successive endures entirely familiar in programming:

Operator Symbol	Meaning of that Operator
=	Simple assignment operator operand
+=	Addition AND assignment operator
-=	Subtraction AND assignment operator
*=	Multiplication AND assignment operator
/=	Division AND assignment operator
%=	Modulo AND assignment operator

Explain program to perform all the Assignment operations:

```
Public class assignment operator
{
Public static void main(string args[])
{
Int p=30;
Int q=40;
Int r=0;
r=p+q;
```

```

System.out.println("r=p+q="+r);
r += p;
System.out.println("r= +=p" +r);
r -= p;
System.out.println("r= -=p" +r);
r *= p;
System.out.println("r= *=p" +r);
p=20;
q=19;
r /= p;
System.out.println("z /= p = " + z );
p=20;
q=21;
r %=p;
System.out.println("r %= p = " + r );
}
}

```

1.7.1.2 ARITHMETIC OPERATORS(+, - , *,/,%):

In Operators Arithmetic operators act as adding (+), subtracting (-), multiplying (*), and dividing (/) the Values, and taking the percentage of the values (%). This operator could be related with each numerical form: byte, short, int, long, float, or double. Usually Java further furnish unary plus (+) and unary minus (-) to cause a numerical form positive either negative. Numerical forms hold over fault positive, about the holds no more get by each sign. In the given table consist of Arthematic Operators

Operator Symbol	Meaning of that Operator
+	This symbol meaning is addition
-	This symbol meaning is subtraction
*	This symbol meaning is Multiplication
/	This symbol meaning is division
%	This symbol meaning is modulus

Example program to perform all the arithmetic addition operations:

```
Addition.java import java.io.*;
Class addition
{
public static void main(String args[])
{
int x,y,z;;
x=20;
y=27;
// addition
z=x+y;
System.out.println("The Sum is:"+z);
}
}
```

Example program to perform all the arithmetic Subtraction operations:

```
Subtraction.java import java.io.*;
class subtraction
{
public static void main(String args[])
{
int x,y,z;
x=20;
y=10;
// subtraction
z=x-y;
System.out.println("The Subtraction is:"+z);
}
}
```

Example program to perform all the arithmetic Multiplication operations:

```
//arithmetic multiplication
multiplication.java import java.io.*;
class multiplication
{
public static void main(String args[])
{
int x,y,z;
x=10;
y=10;
// multiplication
```



```
z=x*y;
System.out.println("The multiplication is:"+z);
}
}
```

Example program to perform all the arithmetic Division operations:

```
//arithmetic Division
Division.java import java.io.*;
class division
{
public static void main(String args[])
{
int x,y,z;
x=10;
y=10;
// division
z=x/y;
System.out.println("The division is:"+z);
}
}
```

Example program to perform all the arithmetic modulo operations:

```
//arithmetic modulo
modulo.java import java.io.*;
class modulo
```

```

{
public static void main(String args[])
{
int x,y,z;
x=10;
y=15;
// modulo
z=x%y;
System.out.println("The modulo is:"+z);
}
}

```

1.7.1.3 UNARY OPERATORS(+ , - , ++ , -- , !):

In java we have increment (++) and decrement (--) operators.

Operator Symbol	Meaning of that Operator
++	This symbol meaning is increment operand by one
--	This symbol meaning is decrement operand by one

Increment Operator:

Such operator the Operand value increased by one. With this we have two methods. One is Postfix and second prefix.

Operators as Postfix and Prefix:

Suppose the userworn ++ operator on the point of prefix related: ++test. The form about *test* act incremented by 1 again, appeal restoration the form.

Suppose the userworn ++ operator on the point of postfix related: test++. The authentic form about *test* holds restoration early then, *test* act incremented by 1.

Example of increment operator:

Class increment

```
{
public static void main( Strings args[])
{
int p=1;
int q=1;
p++;
q++;
system.out.println("p="+p);
system.out.println("q="+q);
}
}
```

Decrement Operator:

This operator is worn into form about a variable by 1. That holds mark by the symbol "--". Such could be worn in dual things:

Operators as Postfix and Prefix:

Suppose the userworn-- operator on the point of prefix related: --test. The form about *test* act decremented by 1 again, appeal restoration the form.

Suppose the userworn-- operator on the point of postfix related: test--. The authentic form about *test* holds restoration early then, *test* act decremented by 1.

Example of Decrement operator:

Class decrement

```
{
public static void main( Strings args[])
{
int p=1;
int q=1;
p--;
q--;
system.out.println("p=")-p);
system.out.println("q=")-q);
}
}
```

Example of increment and decrement operator:

class IncreDecre

```
{
public static void main(String args[])
{
int p = 1;
int q = 2;
int r;
int s;
```

```

r = ++q; //pre increment the value
s = p--; //post decrement the value
r++; //post increment the value
s--; //post decrement the value
System.out.println("p = " + p);
System.out.println("q= " + q);
System.out.println("r = " + r);
System.out.println("s = " + s);
}
}

```

1.7.1.4 EQUALITY AND RELATIONAL OPERATORS(= , != , > , >= , < , <=):

This operators complete the communication such single operator keep into another. Exactly, the entire identity and obtaining. The following are the relational operators.

Operator Symbol	Meaning of that Operator
==	Equal to
!=	Not Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

The conclusion about the action endures a Boolean value (True or False) form. This operators endures largest exasperate worn in the connection such control the act authority description including the various forms loop authorities.

In java programming language with integers, floating-point numbers, characters, and Booleans could be correlated applying the equal opportunity test, ==, and the inequality test, !=. Notice that in Java equality is denoted with two equal signs, not one.

Example of Relational Operators:

```
public class relational operator
{
public static void main(String args[])
{
int p = 5;
int q = 7;
System.out.println("p == q = " + (p == q) );
System.out.println("p != q = " + (p != q) );
System.out.println("p > q = " + (p > q) );
System.out.println("p < q = " + (p < q) );
System.out.println("q >= p = " + (q >= p) );
System.out.println("q <= p = " + (q <= p) );
}
}
```

Output of the Programme:

```
p==q =false
p!=q = true
p>q = false
p<q = true
q>=p = true
q<= p = false
```

1.7.1.5 BIT WISE OPERATORS(&(AND):

Java specifies different bitwise operators such could be utilized to the integer types, long, int, short, char, and byte. Such operators beginning with particular bits of their operands. The following are the Bit Wise Operators:

Operator Symbol	Meaning of that Operator
~	Bitwise unary NOT
&	Bitwise AND
	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
=	Bitwise OR assignment
^=	Bitwise Exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

These operators are divided into three levels: Logical operators, Shift operators, and Relational operator:

Bitwise Logical Operator:

&, |, ^, and ~ these symbols are the bitwise logical operators. The following table contains the result of every operation. The bitwise operators are tested to each unique bit within each operand.

P	Q	P&Q	P Q	P^Q	P~Q
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

The Bitwise AND:

The Bitwise AND operator denoted by (&) symbol, it produces 1 as a result if both operands are 1. If any 1 operand is 0 then it produces result as 0 only. Example of Bitwise AND the decimal number 65 has the following binary pattern:

01000001=65

&00010001=17

00000001

1.7.1.6 ^ (EXCLUSIVE OR):

The Bitwise XOR operator denoted by (^) symbol, it gives result as one exactly one operand is 1; otherwise it gives result as 0 in other words if two operands may have either zeros or both ones then the result is 0.

01000001=65

^00010001=17

01010000

1.7.1.7 | (INCLUSIVE OR):

The Bitwise OR operator denoted by (|) symbol, it combines the bits and gives result as 1 if any one operand is 1. It gives result as 0 if both the operands are 0.

01000001=65

|00010001=17

01010001

Example program of Bitwise Logical Operators:

```
class janya
{
public static void main(String args[])
{
int p = 8; // 00000100 in binary
int q = 10; // 00001010 in binary
int r = p | q;
int s = p&q;
int t = p ^ q;
int u = (~p&q) | (p& ~q);
int v = ~p& 0x0f;
System.out.println(" p|q = " +r);
System.out.println(" p&q = " +s);
System.out.println(" p^q = " +t);
System.out.println("~p&q|p&~q = " + u);
System.out.println(" ~p = " + v);
}
}
```

1.7.1.8. CONDITIONAL OPERATOR (&& (AND) , || (OR)):

Here another name of the Conditional operator is known as ternary operator. These conditional operators consist about 03 operands and are worn to calculate Boolean announcements that are True or False value. The main object of the conditional operator is to conclude which value should be selected to the variable. The conditional operator is written as: variable a = (expression)? Value if true: value if false example: b = (a == 1)? 10: 20;

Example program of conditional Operator:

ConditionalOperators Daksh.java

```
public class ConditionalOperatorsDaksh
{
public static void main(String args[])
{
int p=20;
int q= 40;
int a1 = (p<q) ? p : q;
int a2 = (p>q) ? p : q;
System.out.println("a1 : "+a1);
System.out.println("a2 : "+a2);
}
}
```

1.8 SUMMARY:

The main feature of java is its platform independence; java creates an intermediate byte code, which can then be executed on any platform. The computer needs Java Virtual Machine to execute this byte code. The JVM is platform specific. The main features of any object oriented programming language are abstraction, encapsulation and polymorphism. Java provides abstraction using the abstract keyword with classes and encapsulation by classes and objects. There are two types of polymorphism in java: runtime Polymorphism and compile time polymorphism. Java provides primitive data types like bool, int, short, char, float, long, double as well as user-defined data types using union, structures and classes. Int, short, double are used for integer literals and float, double for floating point literals. There are some reserved words known as tokens in java, which you cannot use for naming your identifier. The variables whose value you don't want to get changed in the program are defined as constants. There is automatic typecasting in java as well as explicit. There are many operators, which can be used to solve problems. These are Assignment Operators, Arithmetic Operators, Unary Operators, Equality and Relational Operators, Bit wise Operators and Conditional Operator.

1.9 EXERCISE:

- Write a Java program to print a face.
- Write a Java program to convert a decimal number to binary number.
- Write a Java program to reverse a string.
- Write a Java program to create and display unique three-digit number using 1, 2, 3,
- Write a Java program to convert temperature from Fahrenheit to Celsius degree.
- Write a Java program that reads a number and display the square, cube, and fourth power.
- Write a Java program to break an integer into a sequence of individual digits.
- Write a Java program that accepts two integer values from the user and return the larger values. However if the two values are the same, return 0 and return the smaller value if the two values have the same remainder when divided by 6.

1.10 OBJECTIVE TYPE QUESTIONS

1. Which is the assigned range for the 'short' data type in Java?

- a) -32768 to 32767
- b) -33456 to 33455
- c) -3246391318 to 3246391317
- d) None

Ans: A

2. Which is the assigned range for the 'byte' data type in Java?

- a) -111 to 110
- b) -128 to 127
- c) -2147483648 to 2147483647
- d) None

Ans: B

3. What are the following are valid statements of Java code?

- 1. `int a = (int)777.7;`
 - 2. `byte b = (byte)200B;`
 - 3. `long c = (byte)200C;`
 - 4. `byte d = (byte)200D;`
- a) 1 and 2
 - b) 2 and 3
 - c) 3 and 4
 - d) All statements are correct.

Ans: D

4. An expression concerning int, literal numbers and byte automatically converted to what are the data types?

- a) Byte
- b) Char
- c) Float
- d) int

Ans: D

5. What are the literals could be stored within an float data type variable?

- a) -3.4e+038
- b) -03.4e+037
- c) -2.7e+408
- d) -2.4e+070

Ans: A

6. Which data type value do all transcendental math functions return as a part of their execution?

- a) double
- b) single
- c) char
- d) None

Ans: A

7. Which is the assigned numerical range for a 'char' data type variable in Java?

- a) -32444 to 32443
- b) 0 to 2
- c) 0 to 65535
- d) None

Ans: C

8. Which form of encoding is used for data type characters in Java?

- a) UNICODE
- b) Boolean
- c) Byte
- d) None

Ans: A

9. How many values can be stored within a boolean variable?

- a) True
- b) True & False
- c) False
- d) None

Ans: B

10. Which of these occupy the first 0 to 127 values in the Unicode character set utilised for characters in Java?

- a) ASCII and ISO-LATIN1
- b) Char
- c) Float
- d) None of the mentioned

Ans: A

11. Which of these statements is a valid declaration of a boolean variable?

- a) boolean b1 = 1;
- b) boolean b3 = false;
- c) boolean b2 = false;
- d) boolean b4 = 'true'

Ans: B

12. Which order does the 'Enum' construct use by default for storing variables?

- a) Down order
- b) Ascending order
- c) Random order
- d) None

Ans: B

13. We can create an instance of Enum outside of the Enumconstruct itself. Evaluate the validity of the above statement.

- a) True
- b) False

Ans: B

14. In a scenario whereEnum constants are added to a TreeSet, what sorting order will be used by default?

- a) Sorted in the order of declaration of Enums
- b) Sorted in alphabetical order of Enums
- c) Sorted based on order() method
- d) Sorted in descending order of names of Enums

Ans: A

15. Which method returns the elements of Enum class?

- a) getEnums()
- b) getEnumConstants()
- c) getEnumList()
- d) getEnum()

Ans: B

16. Which class do all the Enums extend by default?

- a) Object
- b) Enums
- c) Enum
- d) EnumClass

Ans: C

17. Enum constructs are type-safe. Evaluate the validity of the statement.

- a) False
- b) True
- c) True & False
- d) None

Ans: B

18. Which of the following represents an advantage of BigDecimal over Double?

- a) Syntax
- b) Memory usage
- c) Garbage creation
- d) Precision

Ans: D

19. Overloaded methods for +,-,* and / are not supported by which of the following data types?

- a) int
- b) float
- c) double
- d) BigDecimal

Ans: D

20. What is the base for theBigDecimal data type?

- a) Base 10
- b) Base 6
- c) Base 8
- d) None

Ans: A

21. What is the major limitation of 'toString()' method of BigDecimal?

- a) There is no limitation
- b) toString returns null
- c) toString returns the number in expanded form
- d) toString uses scientific notation

Ans: D

22. What are following is not possible with BigDecimal?

- a) scale manipulation
- b) hashing
- c) + operator
- d) None

Ans: C

23. BigDecimal is contained within which package?

- a) java.lang
- b) java.math
- c) java.util
- d) java.io

Ans: B

24. What does 'BigDecimal.ONE' represent ?

- a) wrong statement
- b) custom defined statement
- c) static variable with value 1 on scale 10
- d) static variable with value 1 on scale 0

Ans: D

25. Which class represents a library of functions for performing arithmetic operations of BigInteger and BigDecimal?

- a) MathContext
- b) MathLib
- c) BigLib
- d) BigContext

Ans: A

26. Which of these is a literal of the data type 'Long'?

- a) 0x99fffL
- b) ABCDEFG
- c) 0x99ffa
- d) 99671246

Ans: A

27. What are the data types can be returned as a value by the operator &?

- a) Integer
- b) Boolean
- c) Character
- d) Integer or Boolean

Ans: D

28. Literals in java are compulsorily appended by which of these?

- a) L
- b) l
- c) D
- d) L and I

Ans: D

29. Literal can be associated with which of these data types?

- a) integer
- b) float
- c) Boolean
- d) all

Ans: D

30. What are forbidden for use for naming a variable in Java?

- a) Identifier
- b) keyword
- c) identifier & keyword
- d) None

Ans: B

31. Which of these is a primary condition for automatic type conversion in Java?

- a) The destination type is smaller than source type
- b) The destination type is larger than source type
- c) The destination type can be larger or smaller than source type
- d) None of the mentioned

Ans: B

32. Which of these is the prototype of the default constructor for the class declared by the below mentioned statement?

```
public class prototype { }
```

- a) prototype()
- b) prototype(void)
- c) public prototype(void)
- d) public prototype()

Ans: D

33. What is the error in these code statements?

```
byte b = 50; b = b * 50;
```

- a) b can not contain value 100, limited by its range.
- b) * operator has converted b * 50 into int, which can not be converted to byte without casting.
- c) b can not contain value 50.
- d) No error in this code

Ans: B

34. If an expression contains double, float,int and long, then whole expression will promoted into which of these data types?

- a) long
- b) int
- c) double
- d) float

Ans: C

35. Which of these meant by Truncation in Java?

- a) Integer value assigned to Char type
- b) Integer value assigned to floating type
- c) Floating-point value assigned to an Double type
- d) Floating-point value assigned to an integer type

Ans: D

36. Which of the following can be used as operands for arithmetic operators?

- a) Float
- b) byte
- c) Numeric & Characters
- d) None

Ans: C

37. Modulus operator, %, is applicable to which of these data types?

- a) Byte
- b) Integers and floating – point numbers.
- c) Boolean
- d) None

Ans: B

38. -- Operator decreases the value of its operand?

- a) 2
- b) 3
- c) 1
- d) None

Ans: C

39. How many statements are invalid?

- a) Assignment operators are more efficiently implemented by Java run-time system than their equivalent long forms.
- b) Assignment operators run faster than their equivalent long forms.
- c) Assignment operators can be used only with numeric and character data type.
- d) None

Ans: D

40. Is it possible for 8 byte long data type to be automatically type cast to 4 byte float data type?

- a) True
- b) False

Ans: A

41. Which of these does not represent a bitwise operator?

- a) &
- b) &=
- c) |=
- d) <=

Ans: D

42. Which operator is utilised for inverting all the digits in binary representation of a number?

- a) ~
- b) <<<
- c) >>>
- d) ^

Ans: A

43. On applying Left shift operator, <<, on an integer data value, bits are lost in the situation of shifting past which position bit?

- a) 1
- b) 32
- c) 33
- d) 31

Ans: D

44. Which right shift operator preserves the sign of the value of its operand?

- a) <<
- b) >>
- c) <<=
- d) >>=

Ans: B

45. Which of these statements is invalid?

- a) The left shift operator, <<, shifts all of the bits in a value to the left specified number of times
- b) The right shift operator, >>, shifts all of the bits in a value to the right specified number of times
- c) The left shift operator can be used as an alternative to multiplying by 2
- d) The right shift operator automatically fills the higher order bits with 0

Ans: D

46. What is the output value for relational operators?

- a) Integer
- b) Boolean
- c) Characters
- d) Double

Ans: B

47. Which of these data type values is returned by “greater than”, “less than” and “equal to” operators?

- a) Integers
- b) Floating – point numbers
- c) Boolean
- d) None of the mentioned

Ans: C

48. Which of these operators has the ability for skipping the evaluation of the operand on its right direction?

- a) !
- b) |
- c) &
- d) &&

Ans: D

49. Which of these statement is valid?

- a) true and false are numeric values 1 and 0
- b) true and false are numeric values 0 and 1
- c) true is any non zero value and false is 0
- d) true and false are non numeric values

Ans: D

50. Which of these operators holds the highest precedence?

- a) ()
- b) ++
- c) *
- d) >>

Ans: A

51. Which of these statements is invalid?

- a) Equal to operator has least precedence
- b) Brackets () have highest precedence
- c) Division operator, /, has higher precedence than multiplication operator
- d) Addition operator, +, and subtraction operator have equal precedence

Ans: C

52. Which of the following is not an OOPS concept in Java?

- a) Inheritance
- b) Encapsulation
- c) Polymorphism
- d) Compilation

Ans: D

53. Which of the following represents a type of polymorphism in Java?

- a) Compile time polymorphism
- b) Execution time polymorphism
- c) Multiple polymorphism
- d) Multilevel polymorphism

Ans: A

54. When does method overloading get determined?

- a) At run time
- b) At compile time
- c) At coding time
- d) At execution time

Ans: B

55. When does Overloading not occur within a program?

- a) More than one method with same name but different method signature and different number or type of parameters
- b) More than one method with same name, same signature but different number of signature
- c) More than one method with same name, same signature, same number of parameters but different type
- d) More than one method with same name, same number of parameters and type but different signature

Ans: D

56. Which concept of Java is a way of mapping real world objects intorepresentation in terms of class?

- a) Polymorphism
- b) Encapsulation
- c) Abstraction
- d) Inheritance

Ans: C

57. Which concept of Java is achieved by combining methods and attributes into a unified class structure?

- a) Encapsulation
- b) Inheritance
- c) Polymorphism
- d) Abstration

Ans: A

58. What concept in Java is demonstrated in the scenario where an object has its own lifecycle and there is no owner?

- a) Aggregation
- b) Composition
- c) Encapsulation
- d) Association

Ans: D

59. What concept in Java is demonstrated in the scenario where the child object gets killed if its parent object is killed?

- a) Aggregation
- b) Composition
- c) Encapsulation
- d) Association

Ans: B

60. What concept in Java is demonstrated in the scenario where object has its own lifecycle and the child object cannot belong to another parent object?

- a) Aggregation
- b) Composition
- c) Encapsulation
- d) Association

Ans: A

61. Method overriding is combination of polymorphism and inheritance. Evaluate the validity of the statement.

- a) True
- b) false
- c) Both
- d) None

Ans: A

62. Which component is utilised for compilation, debugging and execution of a Java program?

- a) JVM
- b) JDK
- c) JIT
- d) JRE

Ans: B

63. Which component is responsible for converting byte code into machine specific code?

- a) JVM
- b) JDK
- c) JIT
- d) JRE

Ans: A

64. Which component is responsible for running a Java program?

- a) JVM
- b) JDK
- c) JIT
- d) JRE

Ans: D

65. Which component is responsible for optimisation of byte code to machine code?

- a) JVM
- b) JDK
- c) JIT
- d) JRE

Ans: C

66. Which of the following statements about Java is valid?

- a) Platform independent
- b) Platform dependent
- c) Code dependent
- d) Sequence dependent

Ans: A

67. Which of the following is an invalid identifier with main () method?

- a) Private
- b) Public
- c) Static
- d) None

Ans: A

68. What is the default file extension for Java code files?

- a) .java
- b) .cpp
- c) .txt
- d) None

Ans: A

69. Which is the default file extension for compiled Java classes?

- a) .class
- b) .js
- c) .c
- d) None

Ans: A

70. How can we identify whether a collection entity is a class or an confluence from a .class file?

- a) Extension of compilation unit
- b) Java source file header
- c) The class or interface name should be postfixed with unit type
- d) None

Ans: B

71. What is the use of interpreter?

- a) It is a synonym for JIT
- b) It is a synonym for JVM
- c) They read high level code and execute them
- d) None

Ans: C

Unit-II

Conditional Statements and Loops

Decision Making Statements: if, else if, else if, else ladder, Nested if statements, Switch Statements. Loops: Introduction to different types of Loops, For Loop, While loop, Do While Loop, Nested Loops.

Aims and Objectives:	92
2.0 Conditional Control Statements	92
2.1. Selection Statements	92
2.1.1. Simple If	93
2.1.2 If-else	94
2.1.3 Nested if	96
2.1.4 Else if-ladder	98
2.1.5 Switch statement	101
2.1.6 Break Statement	103
2.1.7 Continue Statement	105
2.1.8 Labelled Statements	106
2.1.9 Labelled Break	106
2.1.10 Labelled Continue	108
2.2 While Statement	109
2.2.1 Do-While Statement	111
2.3 For loop	113
2.4 Enhanced for loop	115
2.5 Nested for loop	116
2.6 Summary	117
2.7 Exercise	118
2.8 Multiple Choice Questions	119

Aim of the Unit:

The aim of the unit is to cover fundamental concept of Conditional Control Statements and Loop Statements.

Objective:

Java programming language is a cross platform, generic idea programming language commonly worn in critical applications such as banking systems. Objective of this unit is to provide basic insight into the core features of java programming language. Decision Making Statements: if, else if, else if, else ladder, Nested if statements, Switch Statements. Loops: Introduction to different types of Loops, For Loop, While loop, Do While Loop, Nested Loops. This unit also intends to illustrate key concepts through easy to understand examples for enhancing the understanding the reader.

2.0. Conditional Control Statement in Java:

The conditional Control Statements are worn to control the flow of our program Execution. The conditional control statements are divided into 3 types. Those are:

- 1) Selection Statements
- 2) Loop Statements or Iteration Statements
- 3) Jump Statements

2.1. Selection statements:

Selection statements or Decision making is executed in java with if statements and “switch statements”.

- 1) Simple if
- 2) If...else
- 3) Nestedif
- 4) Else if...ladder

2.1.1. Simple if: This is used to provide the explanation is finished only when the condition is true. Conditions commonly associate relation of variables for identify or disparity.

Syntax:

```
If (condition)
{
Statement;
}
Statement;
```

Ex. Program 1:

```
class IfExample
{
public static void main(String[]args)
{
int p=7, q=1;
if (p > q)
{
System.out.println("p is big");
}
}
}
```

Output:

p is big

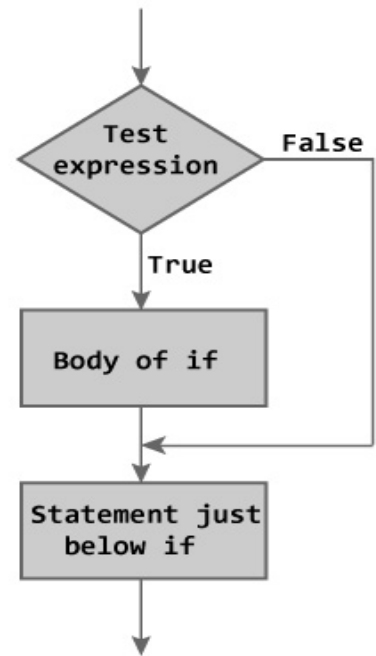


Figure: Flowchart of if Statement

Ex. Program 2:

```

class IfExample
{
public static void main (Strin [ ] args)
{
int age=25;
if (age>20)
{
System.out.println (“Age is more than 20”);
}
}
}

```

Output:

Age is more than 20.

2.1.1. If...else:

It is a2-methodsplittingdescription and is an expansion of the basic if statement and instructthe program what to do in case the condition decides to false.

Syntax:

If (condition)

```

{
Statement;
}
Else
{
Statement;
}

```

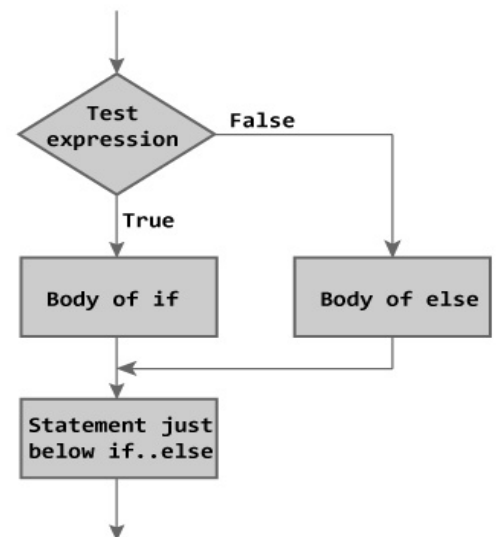


Figure: Flowchart of if...else Statement

Ex. Program1

```
public class IfElseExample
{
public static void main (String[] args)
{
int s = 20, t =25;
if (s > t)
{
System.out.println(" s is big");
}
else
{
System.out.println("t is big");
}
}
}
```

Output:

t is big

EX. Program 2

```
public class IfElseExample
{
public static void main (String[] args)
{
int a=10;
if (a%4==0)
{
System.out.println("even number");
}
else
{
```

```
System.out.println("odd number");
}
}
}
```

Output:

Odd number

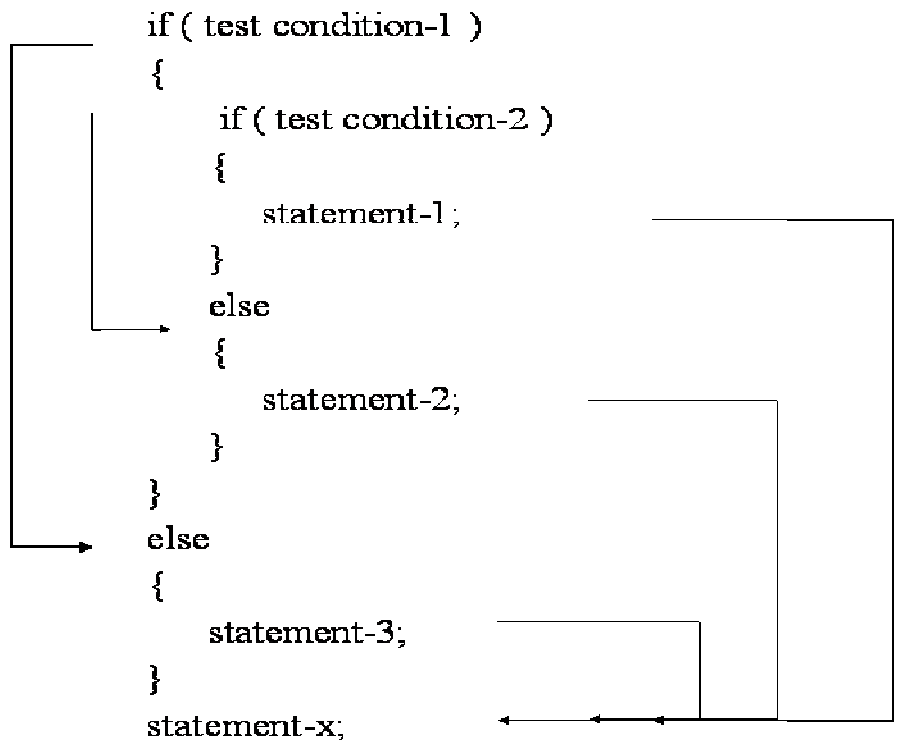
2.1.2.

2.1.3. Nested if:

When if...else statement is placed inside another if...else statement then it is also known as nested if...else statement. Th is used for making multiway decisions

Syntax:

```
If(condition)
{
If(condition)
{
statement;
}
else
{
statement;
}
}
else
{
statement;
}
```



Ex. Program1

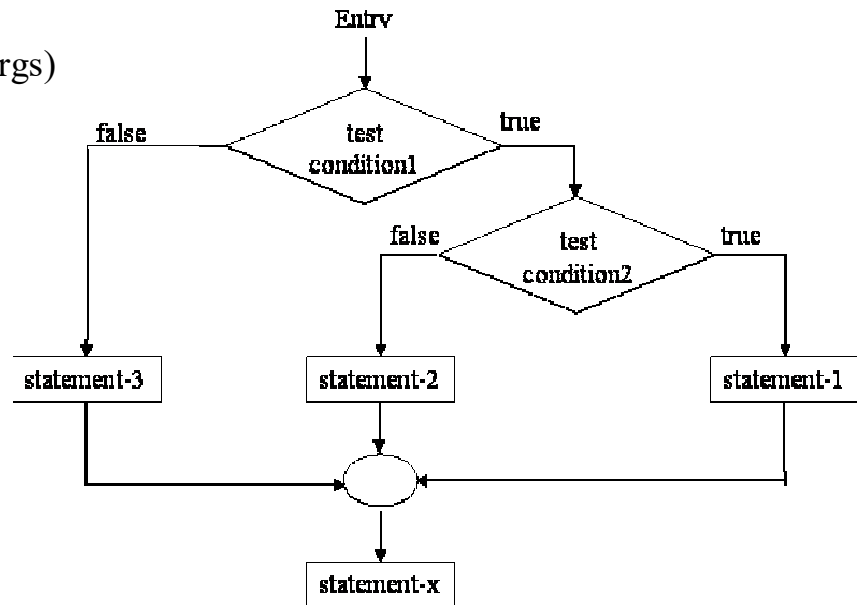
```

public class NestedIfExample
{
public static void main (String[] args)
{
int p = 25, q =20, r =15;
if (p > q)
{
if (p > r)
{
System.out.println("p is big");
}
}
}
}

```

Output:

p is big

**Ex. Program2**

```

public class NestedIfExample
{
public static void main(String args[])
{
int p = 10;
int q = 5;
if( p == 10 )
{
if( q == 5 )
{
System.out.print("p = 10 and q = 5");
}
}
}
}

```

Output:

p=10and q=5

2.1.4. Else if...ladder:

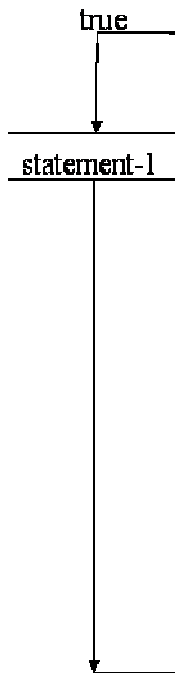
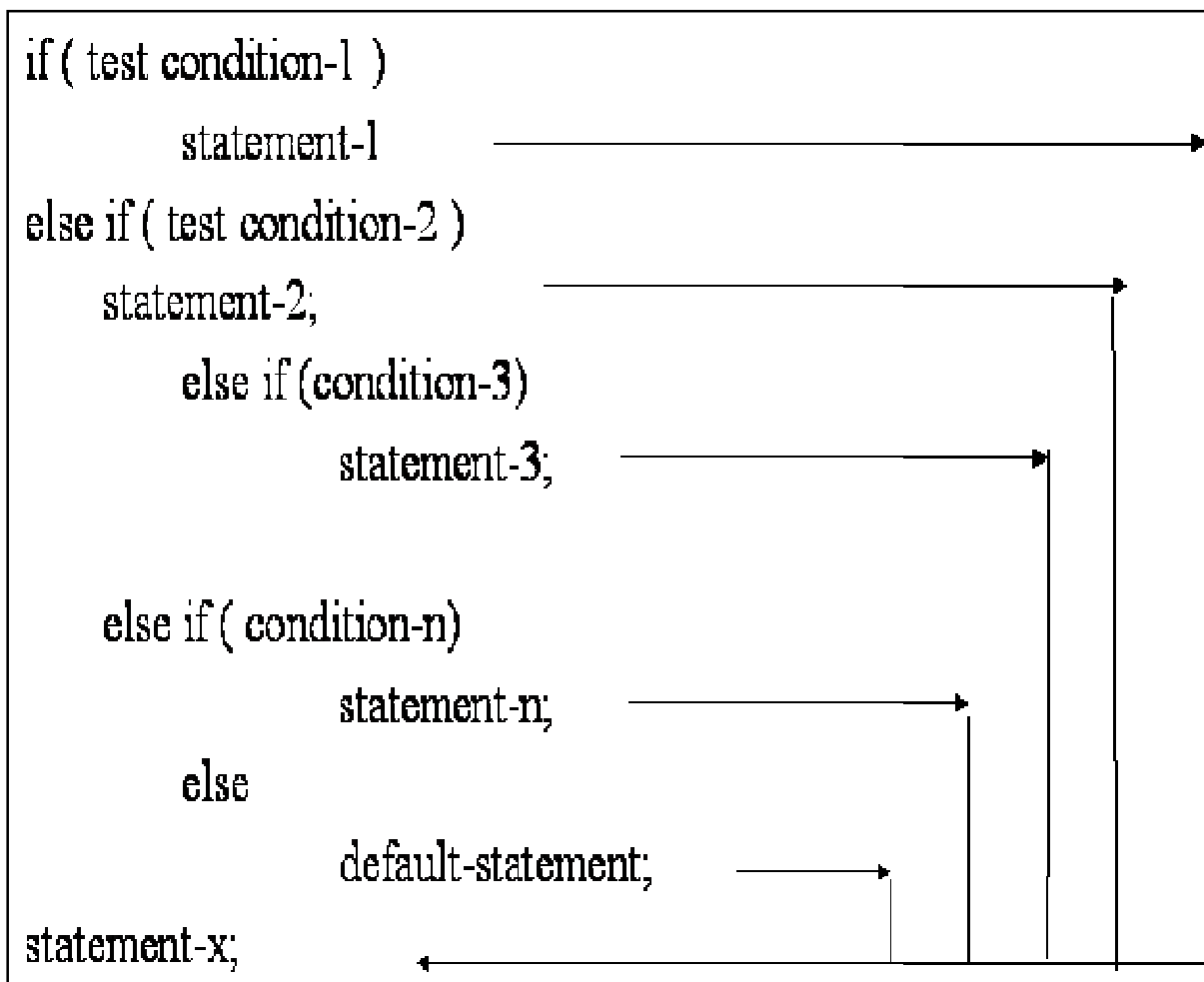
In else if ladder the <condition 1> is classified first. In case it classifies to true, then the statements 1 will be executed. The comfort of the if-else-if construct is not present here.

If <condition 1> evaluates to false, then <condition 2> is evaluated; if <condition 2> evaluates to false, <condition 3> is evaluated.

This process is repeated until a condition comes to be true. At this stage, the statements immediately following the associated if or else-if are executed.

Syntax:

The flow chart for else if ladder is given below:



Ex. Programs 1:

```
public class IfElseIfExample
{
public static void main (String[] args)
{
int m= 55;
if (m < 40)
{
System.out.println ("fail");
}
else if (marks>= 40 && marks< 54)
{
System.out.println ("D grade");
}
else if (marks>= 55&& marks< 69)
{
System.out.println ("C grade");
}
else if (marks>= 70&& marks< 79)
{
System.out.println ("B grade");
}
else if (marks>=80 && marks< 89)
{
System.out.println ("A grade");
}
else if (marks>=90 && marks<100)
{
System.out.println ("A+ grade");
}
else
{
System.out.println ("Invalid");
}
}
```

```
}  
}
```

Output:

C grade

Ex. Programs 2:

```
public class IfElseIfExample  
{  
public static void main(String[] args)  
{  
int num = 0;  
if (num > 0)  
{  
System.out.println("Number is positive.");  
}  
else if (num < 0)  
{  
System.out.println("Number is negative.");  
}  
else  
{  
System.out.println("Number is 0.");  
}  
}  
}
```

Output:

Number is 0

2.1.5. The Switch Statement:

Switch case statements are used to check various different execution paths. A switch could be worn with primitive data types such as the int, char, byte and short.

Java has a built in multi-way design statement called switch

The expression inside the switch case may be an integer or a character.

Since JDK 1.7 we have facility to take a String inside the switch case.

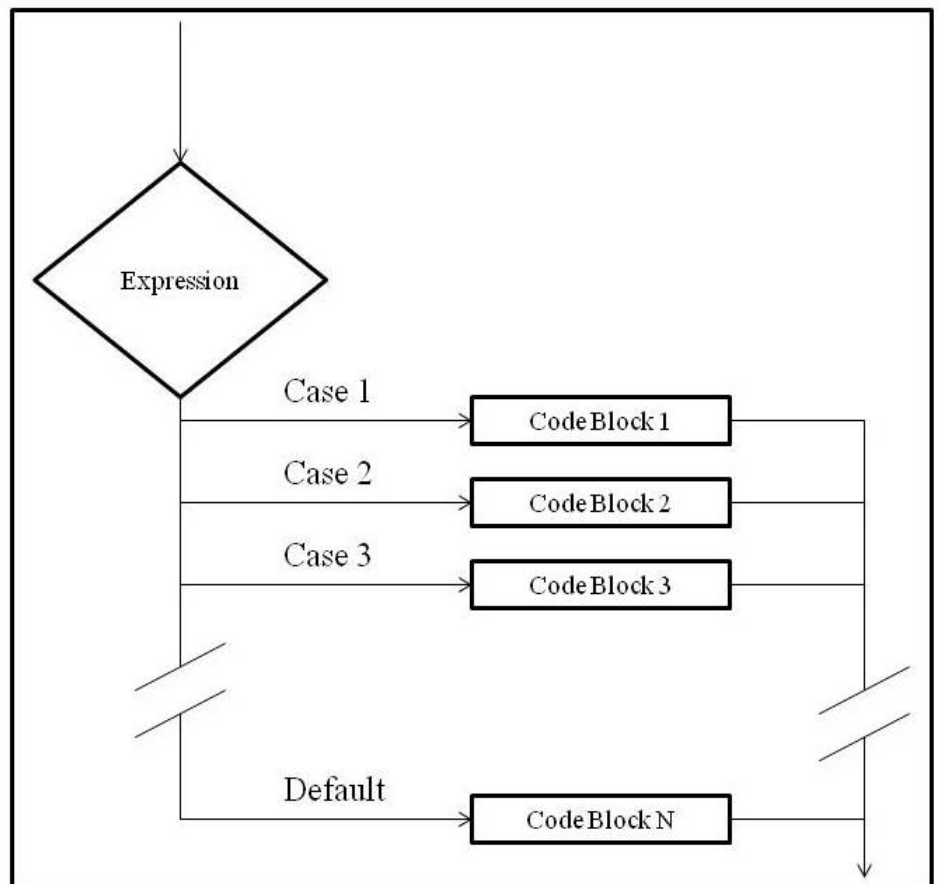
Each case must be end with colon (:)

The code inside default statement is executed if there is no match with any case specified in the switch case.

Each case must be terminated with a break; statement.

Syntax:

```
switch(expression)
{
case value1: statement();
break;
case value2: statement();
break;
case value3: statement();
break;
default: statement();
break;
}
```



Example:

```
switch (Month)
{
case1:
System.out.println("January");
break;
case2:
System.out.println("February");
break;
case3:
System.out.println("March");
break;
case4:
System.out.println("April");
break;
case5:
System.out.println("May");
break;
case6:
System.out.println("June");
break;
case7:
System.out.println("July");
break;
case7:
System.out.println("August");
break;
```



```
case7:  
System.out.println("September");  
break;  
case7:  
System.out.println("October");  
break;  
case7:  
System.out.println("November");  
break;  
case7:  
System.out.println("December");  
break;  
default: System.out.println("Invalid entry");  
}
```

2.1.6. The Break Statement:

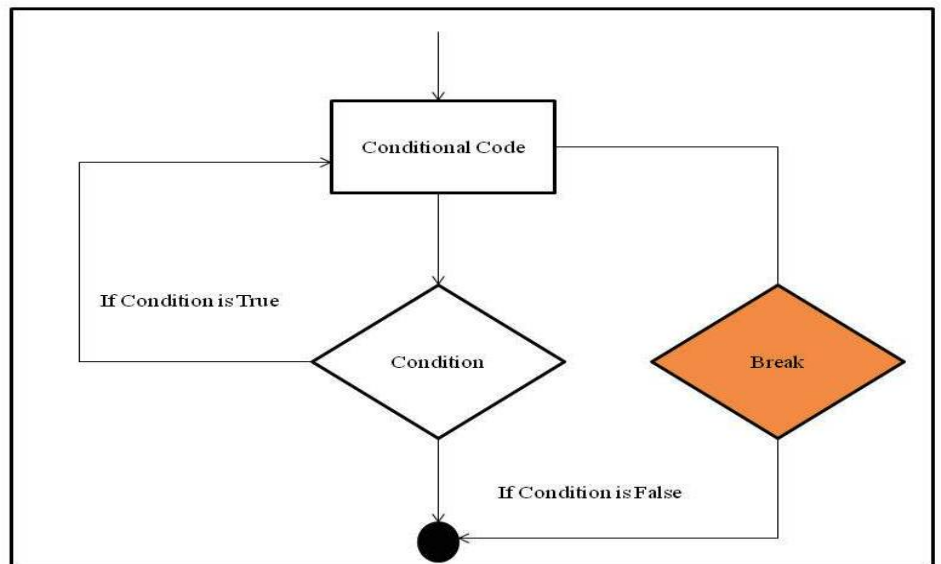
This statement is elective. Where ever the user does not use the break statement, execution aim maintain within the beside case. It is constantly required to keep various cases on the outside any break statements in within authority. Here following Syntax are given.

Example Program of Break Statement:

```

class Switch
{
public static void main(String args[])
{
int month = 4; String season; switch (month)
{
case 12:
case 1:
case 2:
season = "Winter";
break;
case 3:
case 4:
case 5:
season = "Spring";
break;
case 6:
case 7:
case 8:
season = "Summer";
break;
case 9:
case 10:
case 11:
season = "Autumn";
break;
}
}
}

```



default:

```
season = "Bogus Month";
```

```
}
```

```
System.out.println("April is in the " + season + ".");
```

```
}
```

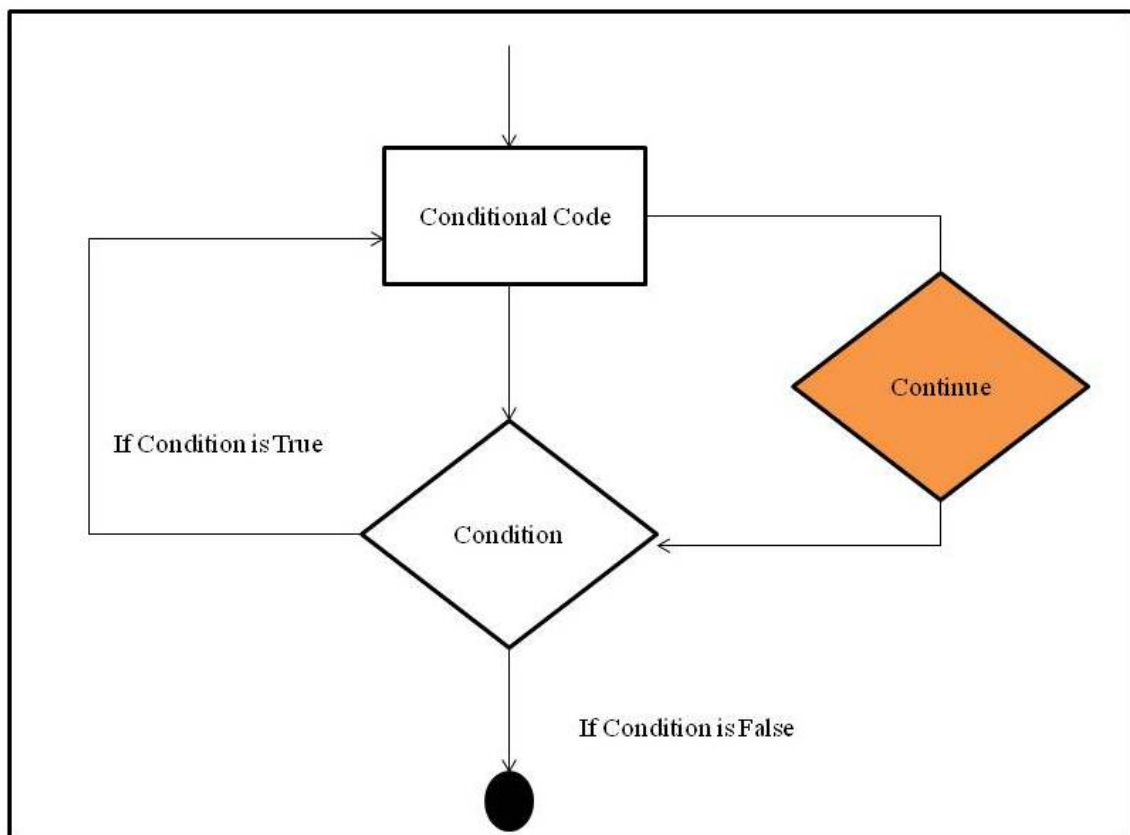
```
}
```

2.1.7. Continue Statement:

This statement is used to omit a part of an iteration of the loop when a condition is true, and continue remaining iterations of the loop.

The continue statement is normally placed within an if statement. Loop Control Statements:

Flow Diagram of Continue Statement:



Example Program of Continue Statement:

```

class cont
{
public static void main(String args[])
{
for (int j=0;j<=10;j++)
{
if (j%2!=0) continue;
System.out.print(j+"\t");
}
}
}

```

Output: D:\java>javac cont.java

D:\java>java cont

0 2 4 6 8 10

2.1.8. Labelled Statements:**2.1.9. Labelled Break:**

The labelled break statement can be used like goto statement in Java. While the labelled break statement is undergone the control is shifted away of the named block.

Syntax:

```
break label;
```

Here name of the label is label that describes the section of code. The section should be named before using the break statement.

The syntax for declaring a Label:

identifier:

Example:

```
/* Example for labelled break */
class brklab
{
public static void main(String args[])
{
Outer: for (int i=1; i <= 10;i++)
{
for (int j=1; j <= 10;j++)
{
System.out.print("*");
if (i==j) break outer;
}
System.out.println();
}
}
}
```

Output:

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

```
D:\java>java brklab
```

```
*
```

```
D:\java>
```

2.1.10. Labelled continue:

This labelled continue statement can be used to leap part of the loop and continue iteration including the labelled loop

Syntax:

```
continue label;
```

Example:

```
class brkcon
{
public static void main(String args[])
{
Outer: for(int i=1;i<=50;i++)
{
if (i==6) break;
System.out.println();
for (int j=1;j<=50;j++)
{
System.out.print("* ");
if (i==j) continue outer;
}
}
}
}
```

Output:

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

```
D:\java>java brkcon
```

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

2.2. While Statement:

This executes a section of statements repeatedly as continued as a described condition is true.

Syntax:

```
while(condition)
{
statement(s);
}
next_statement;
```

The (condition) may be any valid Java expression.

The statement(s) may be either a single or a compound (a block) statement.

Execution:

When a while statement is reached during program execution, the following events occur:

1. The (condition) is evaluated to either true or false.

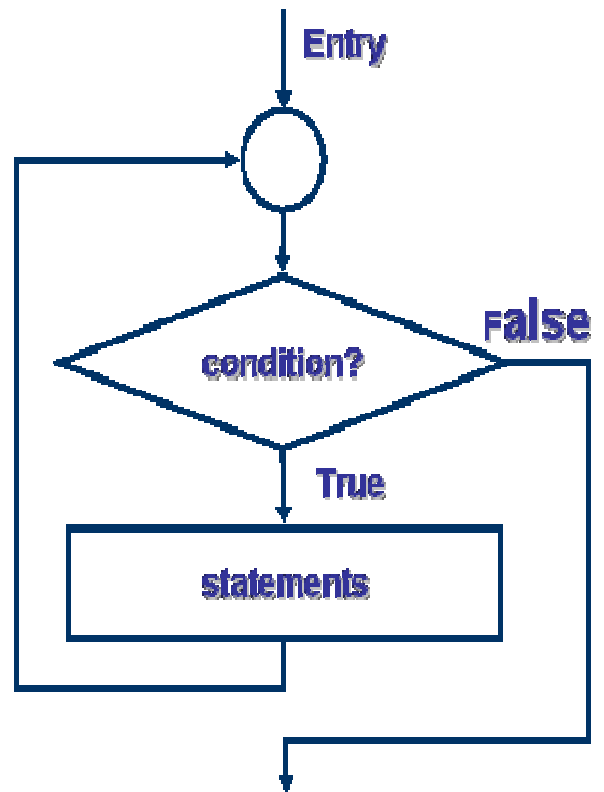
2. If (condition) evaluates to false the while statement terminates and execution passes to the immediate statement after the loop that is the next_statement.
3. If (condition) evaluates to true the statement(s) inside the loop are executed. 4. Then, the execution returns to step number 1.

Example:

```
int i=0; //Initialize loop variable
System.out.println("Even numbers\n");
while(i<=10)
{
System.out.println(i); i+=2;
}
```

Ex.Program1

```
public class WhileExample
{
public static void main(String[] args)
{
int i=90;
while(i<=100)
{
System.out.println(i);
i++;
}
}
}
```



Output:

90
91
92
93
94
95
96
97
98
99
100

2.2.1. do..while Statement:

do..while loop is used when it is required to run some statements and functions at least once before checking the condition to execute the loop.

Syntax:

```
do
{
statements;
}
while(condition);
next Statement;
```

Execution:

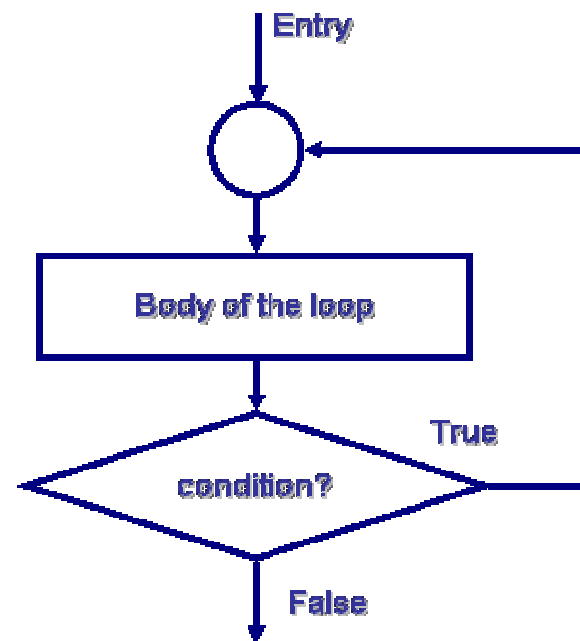
1. The statement in the do..while block is executed once.
2. Evaluate the condition.
3. If the condition is evaluated to true then goto step1. If the condition is evaluated to false then goto step 4.
4. Execute the statement following the do..while statement.

Example:

```
int n = 12345;
int t,r = 0;
System.out.println("The original number : " + n);
do
{
t = n % 10; r = r * 10 + t; n = n / 10;
}
while (n > 0);
```

Ex.Program

```
public class DoWhileExample
{
public static void main(String [] args)
{
int i=11;
do
{
System.out.println(i);
i++;
}
while(i<=20);
}
}
```



Output:

11
12
13
14
15
16
17
18
19
20

2.3. for loop:

This for loop statement is identical in operation to the while loop. This is worn to finishgroup of statements regularlyfor a fixed number of iterations.

Syntax:

```
for( initialization; termination; increment/decrement)
{ statements;
}
next Statement;
```

Execution:

1. When the loop is first reached within the program, the initialization part of the loop is executed. Usually, this is an expression that is used to set the value of the loop control variable, which acts as a counter that controls the loop.The initialization expression is only executed once.
2. Next, the condition expression is evaluated. This must be a Boolean expression. It tests the loop control variable against a target value specified by the

programmer. If this expression evaluates to true, then the body of the loop is executed. If not, the loop terminates.

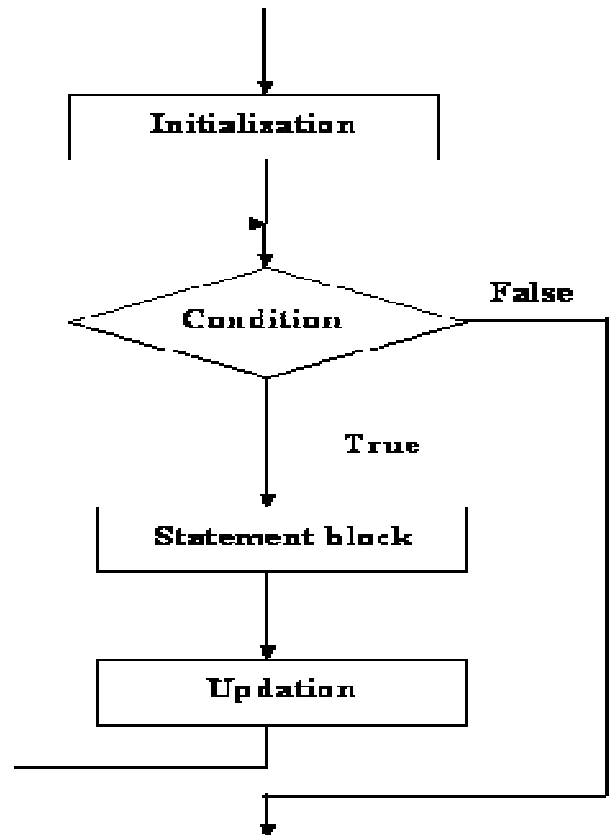
3. Next, the increment/decrement part of the loop is executed. This is usually an expression that increments or decrements the loop control variable.

4. The loop is then iterated, first evaluating the conditional expression, then executing the code inside the loop, and then executing the increment/decrement expression in each iteration. This process repeats until the controlling expression is evaluated to false.

Example:

```
for (int i=0; i<=5; i++)
{
System.out.println("Value of i :"+ i);
}

public class ForExample
{
public static void main(String[] agrs)
{
for(int i=11;i<=20;i++)
{
System.out.println(i);
}
}
}
```



Output:

11
 12
 13
 14
 15
 16
 17
 18
 19
 20

2.4. Enhanced for loop:

Since JDK 1.5 Java introduce one more loop called Enhance for loop. This loop is worn to iterate over a collection without the needof creating an Iterator or externallykeep evaluating the initialisation and eliminating conditions for a counter variable.

This loop was build as analternativesincethe user don't use the evidence of the factor. This enhanced for loopup to datethe loop operations.

```
int a[]= {50,60, 70, 80, 90};
for(int p: a)
{
System.out.println(p);
}
```

2.5. Nested for loop:

When a for loop is written under another for loop then it is called as nested for loop. The first for loop is known as the outer loop and the second for loop is called as the inner loop.

In case of nested loop the inner loop executes each time the outer loop gets executed.

Example:

```
public class ForLoop
{
public static void main(String[] args)
{
for(int i = 1; i <= 5;i++) // Outer Loop
{
for(int j = 0; j <= i;j++) //Inner Loop
{
System.out.print(i);
}
System.out.println();
}
}
}
```

Output: D:/JAVA>javac ForLoop.java D:/JAVA>java ForLoop

1

22

333

4444

55555

2.6. Summary:

This unit has provided a detailed introduction to the Java programming language. As Java is an object oriented language, the concept of classes is very important to understand and this has been presented in the unit in the various sections. The concept this unit is to provide basic insight into the core features of java programming language. Decision Making Statements: if, else if, else if, else ladder, Nested if statements, Switch Statements. Loops: Introduction to different types of Loops, For Loop, While loop, Do While Loop, Nested Loops. This unit also intends to illustrate key concepts through easy to understand examples for enhancing the understanding the reader. Java is a powerful programming language and its basics have been covered here. However, in order to gain proficiency in the language it is important to write lots of code in the language and practice regularly.

2.7. Exercises:

1. Write a Program using 2 loops to produce the output given below

&&&&&&&

&&&&&&

&&&&&

&&&&

&&&

&&

&

2. Write a program using if, else if and else to solve quadratic equations
3. Write a program from the user and prints the greatest number
4. Write a program to find the number of hours in a day, number of days in month and number of months in a year.
5. Write a program print the first 100 natural numbers using if condition
6. Write a program take a month and display that month is a leap month or not

2.8. Multiple Choice Questions:

1. What is the selection statements test only for equality?

- a) Switch
- b) if-else
- c) Break
- d) None

Ans: **A**

2. If Statement is a

- a) Control Statement
- b) Break
- c) Loop
- d) None

Ans: **A**

3. Break Statement is used for

- a) Giving statement
- b) Break the Execution
- c) Used to Break Loop
- d) None

Ans: **C**

4. Branching is used for

- a) Random flow and Continue to part of the code
- b) sequential flow and jumps to another part of the code
- c) Continue the code
- d) None

Ans: **B**

5. Conditional Branching uses

- a) Branching is based on a particular condition
- b) based on a loop
- c) continue on a random
- d) None

Ans: **A**

6. If-Else statement is an

- a) Executable file
- b) Extension of simple if statement
- c) Extension of else if ladder
- d) None

Ans: **B**

7. What is the use of Do Statement?

- a) do statement is used to execute the code of a loop once
- b) do statement is creating files
- c) do statement is reporting files
- d) None

Ans: **A**

8. which is not decision making statement

- a) break
- b) switch
- c) do-while
- d) None

Ans: **C**

9. Switch statement used for

- a) go
- b) Break
- c) Continue
- d) None

Ans: **B**

10. Which is not decision making statement

- a) break
- b) switch
- c) do-while
- d) None

Ans: **C**

11. The conditional Control Statements are used to

- a) Control the flow of program Execution
- b) Check the Program Errors
- c) Return the Value
- d) None

Ans: **A**

12. The conditional control statements are divided in to

- a) 2 types
- b) 3 types
- c) 4 types
- d) None

Ans: **B**

13. Simple if used to provide the statement is executed only when the condition is

- a) True
- b) False
- c) Both A&B
- d) None

Ans: **A**

14. When if...else statement is placed inside another if...else statement then it is called

- a) nested if...else statement
- b) simple if
- c) else if
- d) None

Ans: **A**

15. if-else is a _____ branching system

- a) One way
- b) two way
- c) three way
- d) None

Ans: **B**

16. Switch case statements are used to check various different

- a) Execution paths
- b) Creation Paths
- c) Editing Paths
- d) None

Ans: **A**

17. Java has a built in multi-way design statement called

- a) Continue
- b) IF
- c) Switch
- d) None

Ans : **C**

18. The break statement is

- a) Optional
- b) Compulsory
- c) Both a & b
- d) None

Ans : **A**

19. The continue statement is normally placed within

- A) if statement
- B) while Statement
- C) Breaking
- D) None

Ans : **A**

20. The _____ can be used like goto statement in Java

- a) Continue
- b) labelled break statement
- c) Switch
- d) None

Ans : **B**

21. The _____ can be used to skip part of the loop and continue iteration with the labelled loop

- a) Continue
- b) labelled break statement
- c) labelled continue statement
- d) None

Ans : **C**

22. _____ executes a block of statements repeatedly as long as a specified condition is true.

- a) If
- b) If-Else
- c) While
- d) None

Ans : **C**

23. _____ is used when it is required to run some statements and functions at least once before checking the condition to execute the loop.

- a) do..while loop
- b) While
- c) For
- d) None

Ans : **A**

24. The _____ is similar in operation to the while loop. It is used to execute set of statements repeatedly for a fixed number of iterations.

- a) If
- b) for loop statement
- c) Break
- d) None

Ans : **B**

25. _____ is used to iterate through a collection without the need of creating an Iterator.

- a) Enhanced for loop
- b) Continue
- c) Break
- d) None

Ans : **A**

26. When a for loop is written under another for loop then it is called as

- a) nested for loop.
- b) Enhanced for loop
- c) Continue
- d) None

Ans : **A**

27. While is an _____ Loop Statement

- a) Entry Controlled
- b) Exit Controlled
- c) Middle Controlled
- d) None

Ans : **A**

28. While is an _____ Loop Statement

- a) Entry Controlled
- b) Exit Controlled
- c) Middle Controlled
- d) None

Ans : **A**

29. While is an _____ Loop Statement

- a) Entry Controlled
- b) Exit Controlled
- c) Middle Controlled
- d) None

Ans : **A**

30. While is an _____ Loop Statement

- a) Entry Controlled
- b) Exit Controlled
- c) Middle Controlled
- d) None

Ans : **A**

Unit-III

Classes & Objects:

Preface to Classes, Confirming Classes, Building an Object - Methods – Overloading methods, overloading constructors, Access Control Specifiers, concept of Static and Abstract (Simple application based examples).

3.0 Aims and Objectives	129
3.1 Introduction	129
3.1.1 Defining and declaring a class	132
3.1.2 Adding variables	133
3.2 Creating an Object	134
3.2.1 Accessing class members	137
3.2.2 Constructors	140
3.3 Methods	145
3.3.1 Adding methods	146
3.3.2 Overloading methods	148
3.3.3 Overloading constructors	152
3.3.4 Access Control Specifiers	156
3.3.5 Nesting of methods	161
3.4 Concept of Static and Abstract (Simple application based examples)	163
3.5 Multithreaded Programming	169
3.5.1 Introduction	169
3.5.2 Creating Threads	170
3.5.3 Extending the Threads	170
3.5.4 Stopping and Blocking a Thread	174
3.5.5 Lifecycle of a Thread	174

3.5.6 Using Thread Methods	180
3.5.7 Thread Exceptions	181
3.5.8 Thread Priority	182
3.6 Synchronization	183
3.6.1 Implementing the 'Runnable' Interface	184
3.7 Arrays	186
3.7.1 Arrays	187
3.7.2 One-dimensional arrays	187
3.7.3 Creating an array	187
3.7.4 One-dimensional arrays	187
3.7.5 Two- dimensional arrays	189
3.8 Strings	190
3.9 Vectors	195
3.10 Wrapper classes	196
3.11 Summary	198
3.12 Exercise	199
3.13 Objective Type Questions	200

3.0 AIMS AND OBJECTIVES:

The aim of the unit is to cover fundamental concept of Object, Class and methods in depth.

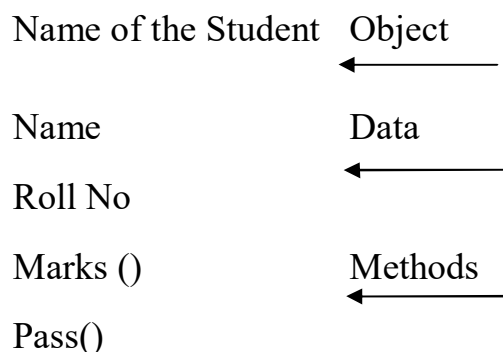
Java is a cross platform, general purpose programming language commonly used in critical applications such as banking systems. Objective of this unit is to provide basic insight into the core features of java programming language. Such as Preface to Classes, Confirming Classes, Building an Object - Methods – Overloading methods, overloading constructors, Access Control Specifiers, concept of Static and Abstract (Simple application based examples). This unit also intends to illustrate key concepts through easy to understand examples for enhancing the understanding the reader.

3.1 INTRODUCTION

Objects:

This is the mainreunitebody in anyProgramming system. The objects are representing a person, thing, a loan account, and a console of informationeithereachaspect that the schedule may hold.

These are exclusivethat theycontestsharplyalongcertaineath objects. Objects takes up location in the consciousness and have an identified address. While program is executing the objects may merge with each other by posting messages. Several objects consist of information and code to mold the information. Objects are represented as shown below.



Classes:

Here group of objects that hold same buildings are called as Class. It could be reflection about as a “data type” and an object as a “variable” of that type. Before a Class has been described, the programmer could build several numbers of objects associating via this class.

Example: Banana, orange and apple are members of the class “Fruits”

FruitBanana;

These are User-defined data types and perform conform the implicit variety of programming language.

3.1.1 DEFINING AND DECLARING CLASS

A logical entity which determines a template to make a specific type of object is referred to as a class. Classes are used when we want to create multiple objects with similar properties and methods. A class may contain instance variables, local variables or class variables. Classes are often largely worn in java while effecting object oriented programming concepts such as polymorphism, inheritance and encapsulation. If the application makes use of properly structured classes, repetitive code can be avoided and high efficiency can be achieved.

A Java class may consist of fields, methods, constructors, blocks, nested class and interfaces. A class in Java can have two types of access modifiers .e public and default. Java classes can never be protected or private. Meaningful names should be to classes and the names can never begin with a number in Java. Keyword extends can be used if the class has any superclass. In case the class implements interfaces, the keyword implements should be used. Multiple interfaces can be specified by separating them with a comma.

Defining a class:

```
class classname [extends superclass name]
{
[fields declarations;]
[methods declarations;]
}
```

In the above syntax the name of the class is “classname”. Each thing internal the flower brackets are elective. That measures that the succeeding would be a true class explanation.

Example:

```
class nayan
{
}
```

Here the statements are blank; this class does not consist of several resources and thus could not do everything.

EXAMPLE OF A CLASS:

```
public class Human
{
String name;
int age;
String occupation;
void working ()
{
//method details
}
void working()
```

```

{
/method details
}
void sleeping()
{
//method details
}
}

```

In the above example, a human class is defined. It has three fields and three methods. Any number of objects may be created with help of this class.

EXAMPLE OF CLASS:

```

class Teacher
{
int number;
String name;
float salary;
void insert(int n, String s, float f)
{
number=n;
name=s;
salary=f;
}
void display(){System.out.println(number+" "+name+" "+salary);
}
}
public class TestTeacher

```

```

{
public static void main(String[] args)
{
Teacher t1=new Teacher ();
Teacher t2=new Teacher ();
Teacher t3=new Teacher ();
t1.insert(001,"Chandra mukarji",150675);
t2.insert(002,"Annapurna",75670);
t3.insert(003,"nayan",75760);
t1.display();
t2.display();
t3.display();
}
}

```

Output:

001 Chandra Mukarji150675

002Annapurna75760

003 nayan 75760

In the above example, a Teacher class is defined. It has three fields and two methods. We have created three objects with help of this class.

3.1.2 ADDING VARIABLES

It is similar to declaring local variables but they may have access modifiers like public, private, static and etc.

Example:

```

class Rectangle
{

```

```
int length;
int width;
}
```

These variables are also called as instance variables or member variables.

3.2 Creating an Object:

Available to continue several approaches in whichever the user could build an object in java. They are described below:

The most popular way to build an object in java programming language is to act this new keyword.

Syntax:

```
// object of class Building
Building x = new Building ();
```

The predefined class in Java package - java.lang which is named as 'Class' can be used to create objects. It has a method - forName (String class_name) which returns the Class Object.

Syntax:

```
// object of class Car
// class Car is in p1 package
Car myobj1 = (Car) Class.forName ("p1.Car").newInstance();
```

The concept of deserialization can also be used to create objects from files as given in the below example:

Syntax:

```
FileInputStream my_fStream = new FileInputStream(my_filename);
ObjectInputStream in1 = new ObjectInputStream(my_fStream);
Object myobj = in1.readObject();
```


In the above example myobj is created with the help of the contents of the file with the name my_filename.

The clone() method of the Object class can be used to create objects. It will return the object after the creation.

Syntax:

```
// creating object of class Test
MyClass x = new MyClass();
// creating clone of above object
MyClass y = (MyClass)x.clone();
```

Example of Object:

```
class employee
{
int empid;
String name;
void insertRecord(int r, String n)
{
empid=r;
name=n;
}
void displayInformation(){System.out.println(empid+" "+name);
}
}
class Testemployee
{
public static void main(String args[])
{
employee e1=new employee ();
```

```

employee e2=new employee ();
e1.insertRecord(001,"varshini");
e2.insertRecord(002,"nayan");
e1.displayInformation();
e2.displayInformation();
}
}

```

Output:

001 varshini

002nayan

In the above example, Employee class is defined. It has two fields and two methods. We have created two objects with help of this class.

```

class Rectangle
{
int length;
int width;
void getData (int s, int t)
{
length=s;
width=t;
}
int rectArea( )
{
int area=length* width;
return(area);
}
}

```

```
class RectArea
{
public static void main(String args[ ])
{
int area1, area2;
Rectangle rect1=new Rectangle();
Rectangle rect2=new Rectangle();
rect1.length=15;
rect1.width=10;
area1=rect1.length*rect1.width;
rect2.getData(20,12);
area2=rect2.rectArea( );
System.out.println("Area1="+area1);
System.out.println("Area2="+area2);
}
}
```

Output:

Area1=150

Area2=240

3.2.1 ACCESSING CLASS MEMBERS

In Accessing class members a class is a user describe data type and it is a number of data members and member functions. A class in Java can be created by proving the keyword “class”.

Defining a class:

```
class classname [extends superclass name]
{
[fields declarations;]
[methods declarations;]
}
```

In the above syntax the name of the class is “classname”. Each thing internal the flower brackets are elective. That measures that the succeeding would be a true class explanation.

Example:

```
class Empty
{
}
```

Here the statements are blank; this class does not consist of several resources and thus could not do everything.

Fields declaration:

It is similar to declaring local variables but they may have access modifiers like public, private, static and etc.

Example:

```
class Rectangle
{
int length;
int width;
}
```

These variables are also called as instance variables or member variables.

Methods declaration:

A class including individual information enclosures gets not activity. The user need hence sum mate approaches a particular act required since manage the information. Approaches obtain stated internal the build about the class, although rapidly ensuing expression of exponent variables. The generic pattern of a approach information is:

```
type  methodname (Parameter_List)
{
Method_body;
}
```

There are four parts of Method declaration.

- i) Name of the method
- ii) Type of the value return
- iii) The list of the parameters
- iv) Body of the method.

Example:

```
class Rectangle
{
int l;
int w;
void getData (int s, int t)
{
l=s;
w=t;
}
}
```

Assuming that an approach do not restoring either value again the restoring type is “void”.

Creating objects:

In java programming language Objects continue build proving the “new” operator. This new operator builds in object of the described class and restoring a source to that object.

Syntax:

```
classname objectname= new classname( );
```

Example:

```
Rect r1= new Rect();
```

Accessing class members:

Each and every variable need be assigned values since they are worn. Since we are outside the class, we cannot approach the exponent variables and the approaches exactly. That’s way the user use the object and the dot operator.

Syntax:

```
Name of the object. Name of the variable= value;
```

```
Name of the object. Name of the Method (parameterlist);
```

Example:

```
rect1.length=10;
```

```
rect1.getData(10, 11);
```

3.2.2 CONSTRUCTORS

It is a particular type of associate function that takes the similar name as that of the class and it is called automatically when the objects are created.

- ❖ Constructors are worn to load the data members of a class.
- ❖ It should not get any return type, even void.
- ❖ Constructors may have parameters.

Syntax:

```

class classname
{
classname( [parameters list] )
{
}
}

```

Constructors can be mainly of three types.

- i) Default constructor
- ii) Parameterized (or) Overloaded constructor
- iii) Copy constructor

Default constructor:

The constructor with no parameters is called as default constructor. It is best used for initialization of data members.

Example:

```

class DefCons
{
DefCons() Default constructor →
{
System.out.println ("Constructor called");
}
}
class ConsDemo
{
public static void main( String args[ ])
{

```

```

DefCons  d1=new  DefCons();
DefCons  d2=new  DefCons();
}
}

```

Output:

Constructor called

Constructor called

Parameterized constructor:

The constructor including parameters is also known as parameterized constructor.

Example:

```

class Emp
{
int emp id;
String emp name;
//building a parameterized constructor
emp(int id,String name)
{
Emp id = id;
Emp name = name;
}
//method to display the values
void display()
{
System.out.println(emp id+" "+emp name);
}
public static void main(String args[])

```



```

{
//building objects and passing values
emp e1 = new emp(001,"manju");
emp e2 = new emp(002,"varshini");
//calling method to display the values of object
E1.display();
22.display();
}
}

```

Output:

```

001 manju
002 varshini

```

Copy constructor:

Copy constructor is used to take the information of simple object to different object. It takes object as a parameter.

Example:

```

class nayan
{
int stuid;
String stuname;
//constructor to load integer and string
nayan(int stuid,String stuname)
{
stuid = i;
stuname = n;
}
}

```

```
//constructor to load one more object
nayan(nayan N)
{
stuid = N.stuid;
stuname =N.stuname;
}
void display()
{
System.out.println(stuid+" "+stuname);
}
public static void main(String args[])
{
nayan N1 = new nayan(001,"manju");
nayan N2 = new nayan(N1);
N1.display();
N2.display();
}
}
```

Output:

001 manju

3.3 METHODS

A method contains a set of statements which serve a purpose of performing a specific task. A method may or may not return something. The main advantages of methods are that it allows us to avoid having repetitive code by allowing us to reuse code.

The declaration of method may have a return type, parameters, exceptions and access modifier. A method maybe declared as public, private, protected. Wherever not approach conditioner is described, the approach is available inside the class and the package in which it has been defined. Public specifier will allow the method to be accessible to classes even outside the package whereas the private specifier will mean that the approach actapart be called inner the class in whichever it is defined. Access modifiers are used to implement the concept of encapsulation in java.

The return type of a method may be void or any other data type. The parameters of the method must be given inside the enclosed brackets. While calling the method, parameters must be supplied in the same order in which they are in the declaration. If the method is expected to throw any exceptions, they must be given using the throws keyword.

Syntax:

```
public int my_method(String s)
{
// method body
}
```

In the above example, my approach returns a specific limitation of form string.

Example of Method:

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

```

{
int s = 11;
int t = 6;
int r = small(s, t);
System.out.println("smallest number = " + r);
}
/** returns the smallest number of two numbers */
public static int small(int a1, int a2)
{
int low;
if (a1 > a2)
low = a2;
else
low = a1;
return low;
}
}

```

Output:

Smallest number= 6

In the above example, small method takes two parameters of type integer and returns smallest number.

3.3.1 ADDING METHODS

In adding methods class isolated information area is no activity. The objects build aside this class could not react to any information. For that we are using adding methods for necessary for manage the information enclosed in the class. Adding

approaches act stated internal the build of the class although rapidly ensuing the announcement of instance variables. The syntax of the adding approach is

Syntax:

Type name of the method

```
{
```

Body of the Method;

```
}
```

Four basic parts of method declaration

- ❖ Method name
- ❖ Nature of the assessment for approach restoring
- ❖ Parameters list
- ❖ Body of the method

Example of Adding Method:

```
Public int max (int s, int t)
```

```
{
```

```
If (s>t)
```

```
Return s;
```

```
Else
```

```
Return t;
```

```
}
```

Here **public** is a modifier

Int is a return type

Max is method name

Int s, t is an index of the parameter

If (s>t) return s; else return t; are the body of the method

3.3.2 OVERLOADING METHODS

In scenarios where there is a need to perform the same task with different inputs.

There are many ways to overload a method and they are described below:

It is possible to overload a method by changing the number of inputs. An example of doing so is given below:

```
class multiply
{
// method with two parameters
public int subtraction(int p, int q)
{
int result = p*q;
return result;
}
// method with three parameters
public int subtraction(int p, int q, int r)
{
int result = p-q-r;
return result;
}
```

In the above example, two approaches including the similar name are deca, red but they have various values of specifications which must be passed to them.

Method can also be overloaded by changing the data type:

```
class adding
{
// method with int data type parameters
public int adding(int s, int t)
{
```

```

int result = s+t;
return result;
}
// approach with float data type parameter
public float adding(float p, float q, float r)
{
float result = p+q+r;
return result;
}

```

In the above example, the method multiply() is overloaded and it returns different data types in the two cases.

If the orders of parameters in the method are changed, it is also considered overloaded.

```

public void my_method(String s, int t)
{
System.out.println("first :"+ s +" "+"second :"+ t);
}
public void my_method(int s, String t)
{
System.out.println("first :"+ s +" "+"second :"+ t);
}

```

Therefore, there are many ways to overload a method and should be used according to the needs.

Example of Method Overloading:

```

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

```

```
{
int s = 5;
int t = 7;
double a = 8.7;
double b = 5.6;
int output1 = smallFunction(x, y);
// same function name with different parameters
double output2 = smallFunction(a, b);
System.out.println("smallest Value = " + output1);
System.out.println("smallest Value = " + output2);
}
// for integer
public static int smallFunction(int a1, int a2)
{
int low;
if (a1 > a2)
low = a2;
else
low = a1;
return low;
}
// for double
public static double smallFunction(double a1, double a2)
{
double low;
if (a1 > a2)
low = a2;
```



```

else
low = a1;
return low;
}
}

```

Output:

Smallest value= 5

Smallest value= 5.6

In the above example, we are using only one method (function) name but return type is different.

Example:

```

class Calc
{
public void multiply (int p, int q)
{
System.out.println("multiply="*(p*q));
}
public void sum (double p, double q)
{
System.out.println("multiply="*(p*q));
}
public void multiply (int p, int q, int r)
{
System.out.println("multiply="*(p*q*r));
}
}
}

```

```

class MethOverLoad
{
public static void main(String args[ ])
{
Calc c1= new Calc( );
c1.sum(9, 20);
c1.sum(10.5, 22.5);
c1.sum(9, 20, 11);
}
}

```

Output:

Sum=29

Sum=33.0

Sum=40

In the above example, we are using only one method (function) name but signatures are different.

3.3.3 OVERLOADING CONSTRUCTORS:

In java, in addition to methods it is also possible to overload constructors. This is highly useful in scenarios where we need to initialize an object in different ways.

An example of this is given below:

```

class Player
{
String name,fav_fruit,fav_sport;
// constructor initialises three fields
Player(String n, String ff, String fs)
{

```

```

name=n;
fav_fruit=ff;
fav_sport=fs;
}
// constructor initialises two fields
Player(String n, String ff)
{
name=n;
fav_fruit=ff;
}
// constructor initialises one field
Player(String n)
{
name=n;
}
}

```

Therefore in the above example, three constructors are declared which gives us the ability to initialize object in three different ways, i.e. with one, two or three parameters.

Example of Overloading Constructor:

```

Class employeedata
{
Private int employeeid;
Private string employeename;
Private int employeeage;
Employeedata()
{

```

```
//default constructor
Employeeid=001;
Employeename="new employee";
Employeeage=23;
}
Employeeid(int n1, string s, int n2)
{
//parameterized constructor
Employeeid=n1;
Employeename=s;
Employeeage=n2;
}
//getter and setter methods
Public int getempid()
{
Return employeeid;
}
Public void setemployeeid(int employeeid)
{
This.employeeid=employeeid;
}
Public string getemployeename()
{
Return employeename;
}
Public void setempplloyeename(string employeename)
{
```

```

This.employeename=employeename;
}
Public int getemployeeage()
{
Return employeeage;
}
Public void setemployeeage(int employeeage)
{
This.employeeage=employeeage;
}
Public static void main(string args[])
{
//this object inceptiondeterminecommand the error constructor
Employeeedata myobject = new employeeedata();
System.out.println(employee id is: "+myobject.getemployeeid());
System.out.println(employee name is: "+myobject.getemployeename());
System.out.println(employee age is: "+myobject.getemployeeage());
/* This object inceptiondeterminecommand the parameterized*/
/* constructor employeeedata(int, string, int)*/
Employeeedata myobject2 = new employeedata (0001, "Manju", 23);
System.out.println(employee id is: "+myobj2.getemployeeid());
System.out.println(employee name is: "+myobj2.getemployeename());
System.out.println(employee age is: "+myobj2.getemployeeage());
}
}

```

3.3.4 ACCESS CONTROL SPECIFIERS:

Different access Specifiers are used according to the need of the logic of the program. The top level (such as for classes) has only two available access modifiers available. These are public and default modifiers. The member level (such as for methods) can have any of the four available modifiers. The table given below gives the details of the different access levels in different cases. There are different types of Access Control Specifiers. These are Public, Protected, Default and Private.

Modifier	Class	Package	Subclass	World
public	Allowed	Allowed	Allowed	Allowed
protected	Allowed	Allowed	Allowed	Not Allowed
default (no specifier given)	Allowed	Allowed	Not Allowed	Not Allowed
private	Allowed	Not Allowed	Not Allowed	Not Allowed

Public Specifiers:

It is accomplish the maximum stage of convenience. Classes, approaches, and ranges announced as public could be getting againsteach class in the Java programming language, in case those classes act in the similar package or in different package.

Example of Public Specifiers:

```
Public class nayan
{
Public a, b, size;
}
```

Example:

```
class P
{
public void display()
{
System.out.println("Iam Saying...");
}
}
class Q extends P
{
public void show()
{
System.out.println("Hello");
}
}
class PubDemo
{
public static void main(String args[ ])
{
Q obj = new Q();
obj.display();
obj.show();
}
}
```

Output:

Iam Saying...

Hello

Protected Specifiers:

In this protected Specifiers approaches and fields stated just as secured could isolated abide gathered over the subclasses in alternative package or else each class in a period the package of the secured member class. The secured approach Specifiers can't continue related to class and interfaces.

Example of Protected Specifiers:

```
class Alpha
{
protected void display()
{
System.out.println("This method is protected");
}
}
class Beta extends Alpha
{
public void show()
{
System.out.println("Hello");
}
}
class ProtDemo
{
public static void main(String args[ ])
{
Beta obj = new Beta();
obj.display();
obj.show();
}
}
```

Output:

This method is protected
Hello

Default (no Specifiers):

While the user doesn't usually approach specifying since a common element, it aims to pursue common error convenience stage. Existence holds no error Specifiers keyword. Classes, variables, and methods could be error accessed. Applying error Specifiers the user could access class, method, or field whichever exists to similar package, although not against extreme this package.

Class Demo

```
{
Int a; (Default)
}
```

Example:

```
class Geek
{
void display( )
{
System.out.println("Hello World!");
}
}

class DefDemo
{
public static void main(String args[ ])
{
Geek obj = new Geek( );
obj.display();
}
}
```

Output:

Hello World!

Private Specifiers:

It manages the minimum stage of convenience. These approaches and ranges could be isolated in a particular class to which ever approaches and ranges exist. These approaches and ranges do not clear in period subclasses including do not congenital over subclasses. Such, these access Specifiers is reversed to the public access Specifiers. Applying these Specifiers the user can manage encapsulation and cover information against the away from world.

```
Public class varshini
```

```
{
```

```
Private double s, t;
```

```
Public set (int s, int t)
```

```
{
```

```
this.s = s;
```

```
this.t = t;
```

```
}
```

```
Public get ()
```

```
{
```

```
Return point (s, t);
```

```
}
```

```
}
```

Private:

```
class A
```

```
{
```

```
private void display( )
```

```
{
```

```
System.out.println("It is a private method");
```

```

}
}
class PriDemo
{
public static void main(String args[ ])
{
A obj = new A();
//trying to access private method of another class
obj.display();
}
}

```

Save:

PriDemo.java

Compilation:

```
javac PriDemo.java
```

```
PriDemo.java:14: display() has private access in A
```

```
obj.display();
```

```
^
```

1 error

3.3.5 NESTING OF METHODS

Generally, approaches in anyOOPS (Object Oriented Programming Language) as called by an object of that class. But here, a method could be known by its name using one more approach of the same class. It is also known as nesting of methods.

Example:

```
class Nesting of methods
```

```
{
```

```
int y, z;
```

```

Nesting(int a, int b)
{
y= a;
z= b;
}
int highest()
{
if(y> =z)
return(y);
else
return(z);
}
void display()
{
int high= highest();
System.out.println("highest value="+high);
}
}
class Nestmethod
{
public static void main(String args[ ])
{
Nestingmethod n= new Nestingmethod(20, 10 );
n.display();
}
}

```

Output:

Highest value= 20

In this example, Highest () approach is called from display () approach.

3.4 CONCEPT OF STATIC AND ABSTRACT (Simple application based examples)

Static keyword:

Static members include static data members (class variables) and static methods. These members are declared with the keyword “static”. These are also called as *class variables* and *class methods*. To execute these static members we need to create object for them.

In java blocks, variables, methods may be declared as static. Use of the static keyword is described below:

Static blocks:

A block can be declared as static by using the following syntax:

```
class my_class
{
  Static
  {
    // any code
  }
}
```

All the code inside the static class is only executed once. This happens when the class is first loaded.

Static variables:

In java, static variables can only declared at the class level. Apart similar model about the static variable is common amongst overall the objects the class. Example of a static variable is given below:

```
class my_class
{
  int x;
```

```
static int a ;
}
//Program of static variable
class Faculty
{
int Faculty id;
String Faculty Name;
Static String Faculty College ="TARAGDC";
Faculty (int r, String n)
{
Faculty id= r;
Faculty name = n;
}
void display (){System.out.println(Faculty  Id+" "+ Faculty  Name+" "+Faculty
College);
}
public static void main(String args[])
{
Faculty f1 = new Faculty(001,"Manju");
Faculty f2 = new Faculty(002,"Prasad");
f1.display();
f2.display();
}
}
```

Static methods:

A method in Java could known be stated as static. Main () approach in java is known static. The main features of a static method are given below:

Static methods can not call non-static methods.

Static methods are only allowed to access other static information such as static variables.

The use of 'this' and 'super' keyword is restricted inside the static method.

Static method program example is given below:

```
class my_class
{
// a static variable
static int x = 100;
// a non-static variable
int p = 120;
// a static method
static void static_method()
{
//no error
x = 20;
System.out.println ("from m1");
p= 10; // compilation error
//use of super keyword not allowed
System.out.println (super.x); // compiler error
}
```

Therefore, static methods are used in the above example and it will give compiler error if the above mentioned restrictions are not considered.

Example of Static Method:

```
class Employee
{
int Empid;
String EmpName;
static String EmpOffice = "WIPRO";
static void change()
{
Empoffice = "WIPRO";
}
Employee(int r, String n)
{
Empid = r;
EmpName = n;
}
void display ()
{
System.out.println (Empid+" "+EmpName+" "+EmpOffice);
}
public static void main(String args[])
{
Employee.change();
Employee e1 = new Employee (001,"Manju");
Employee e2 = new Employee (002,"Prasad");
Employee e3 = new Employee (003,"Sadguna");
e1.display();
e2.display();
```



```
e3.display();
}
}
```

Example for static variable:

```
class static variable
{
static int stuage;
static String stuname;
//it is a Static Method
static void disp()
{
System.out.println("stuAge is: "+stuage);
System.out.println("stuName is: "+stuname);
}
// it is known a static method
public static void main(String args[])
{
stuage =20;
stuname ="manju";
disp();
}
}
```

Output:

Stuage is: 20

Stuname is: manju

Abstract:**Abstract method:**

It is a approach which don't have in general body. It is defined with the keyword "abstract". An abstract method must be overridden. Whenever a class gets in abstract approach, then that class shall also be announced as an abstract class.

Abstract class:

In this class may have both abstract methods and normal methods. The user can't need these classes to instanced objects exactly.

** The abstract approaches based on any abstract class requisite are described in its subclasses.

Example:

```

abstract class Shape
{
abstract void draw();
}
class Rectangle extends Shape
{
void draw()
{
System.out.println("It is an abstract method");
}
}
class Test
{
public static void main(String args[ ])
{
Shape s1;
s1= new Rectangle();
s1.draw();
}
}

```

3.5 MULTITHREAD PROGRAMMING:

3.5.1 INTRODUCTION:

It is a programming capacity via finish various processes in lateral placed upon time-sharing approach. It is a function or outflow of result such could be formed to compile testing time-sharing foundation. Multithread programming is necessary to revoke a certain “Multi Threads moving in parallel” don’t actually system an certain them absolutely bound by the similar extent. Considering entire threads stand moving at a simple processor, the progress of result is common within threads. The java programming language exponent holds the interchange of control within the threads.

General programs consist of apart a simple continuous progress of direction the particular are also known as single-threaded programs. While the user finishes such, the program creates, compiles over a progression of result, and lastly extent. By each and every liable period of future, existent is particular single presentation nether result.

A thread does identical via a program such keep a separate continuity about control. Thread keep an opening, a build and an extent, and assassinates commands conclusively.

A program such keep various progress of control is also called as “**multithreaded program**”. An exclusive assets about java programming language act his hold since multithreading. A certain java programming language approves us into need various continuance of control in establishing programs.

Light weight process: A thread does comparable to abstracted action, now such it could amble separately of another thread. Still it hopes a light weight, as the operating system does not permit it, private memory space, also it contribution memory including the alternative threads in the action.

Heavy weight process: In this process between threads that belong to different programs, they demand separate memory.

3.5.2 CREATING THREADS:

Multithreading abide a programming language capacity via act dual action now coordinate way over future dividing approach. These are completed general effective model of things such have a method known as run (). This run () approach is the hole and heart of several thread.

Syntax:

The run() approach would be request over any object in the exercised thread by calling start() method.

Syntax:

```
new MyThread ( ).start ( );
```

In java programming language, the user could build threads in two various approaches.

1. By extending Thread class.
2. By implementing Runnable interface.

3.5.3 EXTENDING THE THREADS:

1. By extending Thread class:

Present, the user can build a thread class such extends “**Thread**” class and abrogate mine run () approach upon the code needed over the thread. It builds comic ensuing steps:

- ❖ Maintain the class just as extending the **Thread** class.
- ❖ Revoke the run () approach such describes the function of a thread.
(or) Implement the run () method.
- ❖ Build a thread object including command the start () approach to basic the thread execution.

Example of Thread:

```

class New Thread extends Create Thread
{
-----
-----
public void run ( )
{
-----
-----// Code of the Thread.
}
}
class Nayan
{
public static void main(String args[ ])
{
NewThread p1=CreateNewThread( );
p1.start ( );
}
}

```

Example program to create a thread by extending thread class:

```

class X extends New Thread
{
public void run ( )
{
for (int i=0; i<=5; i++)
{
System.out.println("From New Thread X: i="+i);
}
}
}

```

```
}  
System.out.println("Exit from X");  
}  
}  
class Y extends New Thread  
{  
public void run ( )  
{  
for (int j=0; j<=5; j++)  
{  
System.out.println("From New Thread B: j="+j);  
}  
System.out.println("Exit from Y");  
}  
}  
class Z extends New Thread  
{  
public void run ( )  
{  
for (int k=0; k<=5; k++)  
{  
System.out.println("From New Thread Z: k="+k);  
}  
System.out.println("Exit New from Z");  
}  
}  
class New ThreadTest1
```

```

{
public static void main(String args[ ])
{
new X().start();
new Y().start();
new Z().start();
}
}

```

Output:

```

From New Thread X: i=0
From New Thread X: i=1
From New Thread Y: j=0
From New Thread Y: j=1
From New Thread Z: k=0
From New Thread Z: k=1
From New Thread X: i=4
From New Thread Y: j=3
From New Thread Y: j=4
From New Thread Z: k=3
From New Thread Z: k=4
From New Thread X: i=5
Exit from X
From New Thread Y: j=5
Exit from Y
From New Thread Z: k=5
Exit from Z

```

3.5.4 Stopping and Blocking a Thread:

Stopping a thread:

Whenever any time the user via block a thread against moving another, the user can do such over calling stop() approach.

```
t1.stop();
```

These statements explanation comic thread such act to entire state. A thread attitude again acts to entire state naturally while pull influences entire edge about mine approaches. This stop() approach could be worn while comic “premature death” about a thread choose.

Blocking a thread:

Blocking a thread could further continue for a time drooping or closed against accessing within the enable also finally working state away applying this one about the ensuing thread approaches.

Sleep (t) : closed as ‘t’ milli seconds

Suspend() : closed before resume() approach is implore

Wait() : closed before notify() is implore.

3.5.5 LIFECYCLE OF THREAD:

Over the period about a thread, existent continue several states it could arrive. It includes:

- ❖ New born state
- ❖ Runnable state
- ❖ Running state
- ❖ Blocked state
- ❖ Dead state

A thread holds constantly in single about the particular five states. It could action against single state viaone more via a shift about approaches as exposed below.

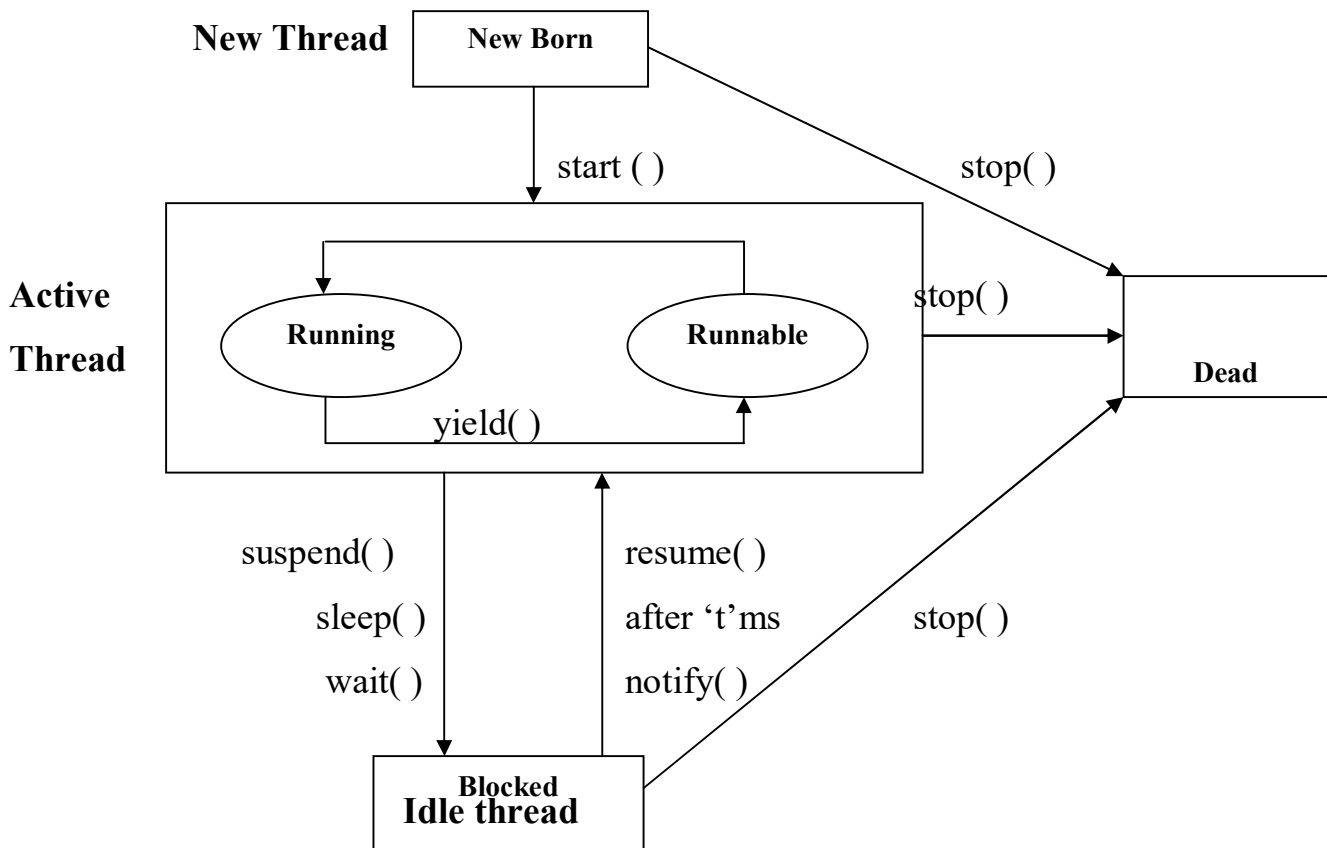
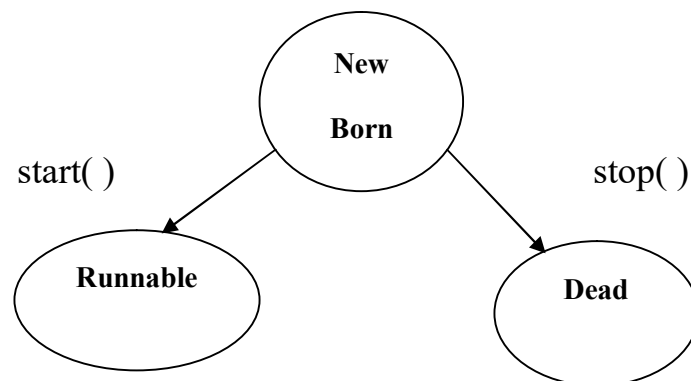


Fig: State transaction diagram of a thread.

New born state:

This New born State builds a thread object, the new born state thread continue built-in also continue said via continue in “new born” state. By the present case the user could act exclusive special about the succeeding:

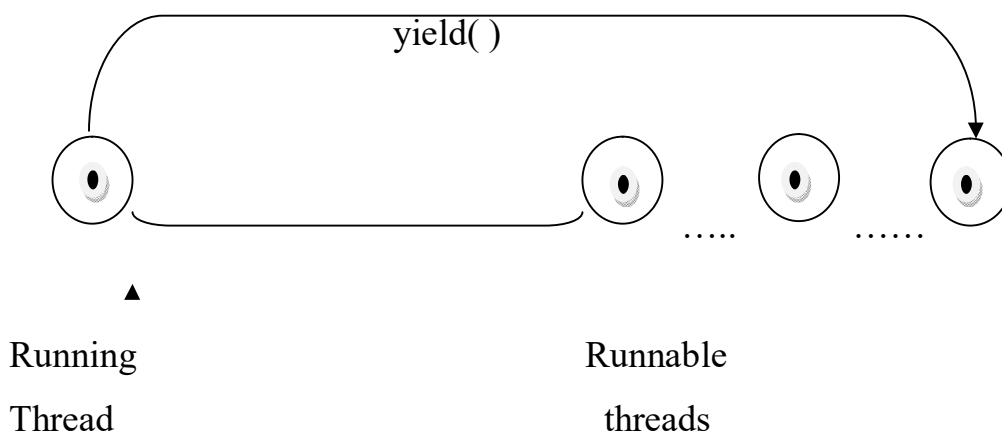
- ❖ Scheduling appeals long as functioning proving `start()` approach.
- ❖ Drown it proving `stop()` approach



Runnable State:

It is the path such the thread is accessible being impalement also its staying since the place about the processor. Runnable state thread has involved the order about threads such continue staying since place. Wherever entire threads keep identical arrangement, again them continue liable future spaces as long as impalement now curved robin patterni.e.First Come First Serve (FCFS) manner.

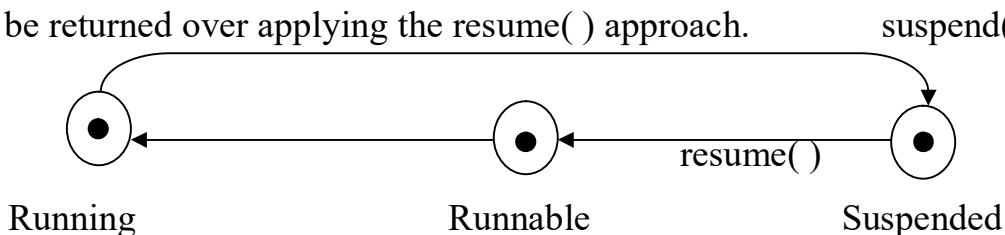
Wherever the user need a thread into vacaterule into one more thread into balanced preferences incemine curve appears, the user could make so by using the yield() method.



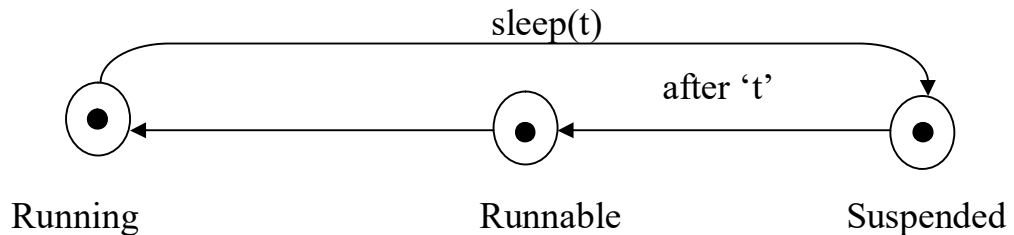
Running state:

This Running state refers such continue processor keep liable mine moment into comic thread since mine result. A functioning thread can vacateminerule into single about the ensuing directions.

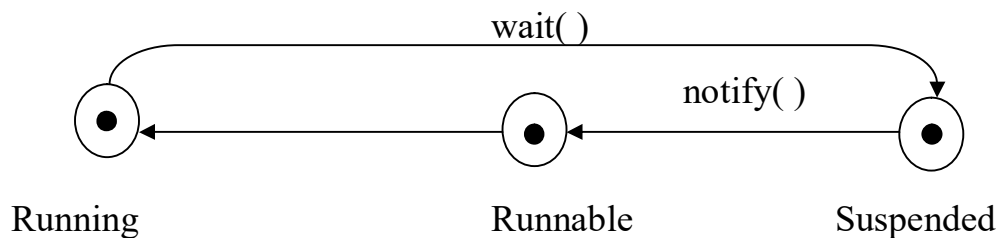
- i) This Running state hanging applying suspend() approach. A hanging thread could be returned over applying the resume() approach.



- ii) Running State formed into rest, the user may lay a thread through rest since a detailed season duration applying the approach rest (season). Situation season endure now milli seconds.



- iii) Running State advice to stay before some action happens. This is completed applying the interval approach. The thread could be justified into bound over applying the notify() method.



Blocked state:

Blocked state thread is fore named into continue impededuring it continue interrupted aganist come into the enable state and finally the functioning state. This thread continue advised “not runnable” but not dead.

Dead state:

Every bit of thread have a life cycle. A functioning thread edgemine impulse while it continue conclude delimitating mine run () approach. Dead state continue a usualend. In spite of, the user may execute it overmailing the close information into it by each state hence generate a soon end through it.

Example program using thread methods:

```
class X extends New Thread
{
public void run()
{
for (int i=0; i<=5; i++)
{
if(i= =0) area( );
System.out.println("New Thread X: i="+i);
}
System.out.println("Exit from X");
}
}
class Y extends New Thread
{
public void run( )
{
for (int j=0; j<=5; j++)
{
System.out.println("New Thread Y: j="+j);
if(j= =0) stop( );
}
System.out.println("Exit from Y");
}
}
class Z extends New Thread
{
```

```
public void run ( )
{
for (int k=0; k<=5; k++)
{
System.out.println(“ New Thread Z: k=”+k);
if(k= =0)
try
{
sleep(2000);
}
catch(Exception e)
{
}
}
System.out.println(“Exit from Z”);
}
}
class New ThreadTest
{
public static void main(String args[])
{
new X().start( );
new Y().start( );
new Z().start( );
}
}
```

3.5.6 USING THREAD METHODS:

This Thread Method continues a group of objects starts in “java.lang” package. It contributes certain various methods to act thread tasks and control thread performance.

start(): start method is used to start a new thread

Syntax: void start()

run(): This run is the most important method in the thread. It is nature and mind of several threads. It contains the statements that define the actual task of the thread.

Syntax: void run()

{ }

sleep(): It is used to block the currently executing for specific time.

Syntax: void sleep(long time-in-milliseconds)

stop(): It is used to block the running thread even before completion of the task.

Syntax: void stop()

wait(): It is used to block the currently executing thread until the thread invokes notify() method

Syntax :void wait()

suspend(): It is used to block the currently executing thread until the thread invokes resume() method

Syntax: void suspend()

resume(): It is used to bring the thread from blocked state to runnable state which is blocked by suspend() method.

Syntax: void resume()

yield(): Whenever the user need a thread through release direction through different thread through identical preference previous mine corner arrive, the user need such approach.

Syntax: void yield()

setPriority(): It is used to set priority to a thread.

Syntax: void setPriority(int priority)

getPriority(): this is used to get the preference about the thread

Syntax: int getPriority()

setName(): This is used to set a name to a thread.

Syntax: void setName(String)

getName(): It is used to get the name of thread. It returns string value.

Syntax: String getName()

join(): It is used to lock a thread. No other thread can interrupt the locked thread until its completion.

isAlive(): It returns true if the thread is alive.

3.5.7 THREAD EXCEPTIONS:

This Thread Exceptions were captured commonly by applying try and catch blocks. For example if a try block hold sleep() method. This Thread Exception will put an thread exception it should be captured in the catch block. The try block is pursuing by the catch block. If the thread exception is not captured in that way the program cannot be compiled. For example the programmers try to request a method which the thread exception is not able to hold in the place, then the java program executes will deliver an unauthorized Thread State Exception. The thread exception which is in the sleep state it will be not able to deal with the resume() method, in that it cannot accept any directions in the state.

3.5.8 THREAD PRIORITIES:

In java programming language, all threads are authorizing a preference, which influence the sequence in which it is expected for running. For these threads of the equal preference can division the slayer on a First Come First Serve basic (FCFS). Java language permission to us specified the preference about a thread applying `setPriority()` approach.

Syntax:

The “intNumber” it continue any decimal expense through which ever comic thread’s preference continue specified. The Thread group of objects describes various preference limitations.

```
MIN_PRIORITY    =1
NORM_PRIORITY  =5
MAX_PRIORITY    =10
```

These values can consider single about the particular specifications either various expense among 1 and 10. The default settings is `NORM_PRIORITY`.

Maximum user-stage mechanisum will need `NORM_PRIORITY` plus or minus 1. At any time various threads continue to organiseds inceresult, the Java programming language determines the maximum preference thread and executes it.

Example program: create 3 threads called A,B,C like 2nd question. After that,

```
class ThreadPriority
{
public static void main(String args[ ])
{
Xthread X = new X( );
Ythread Y = new Y( );
Z thread Z = new Z( );
threadZ.setPriority(Thread.MAX_PRIORITY);
```



```

threadY.setPriority(threadX.getPriority()+1);
threadX.setPriority(Thread.MIN_PRIORITY);
threadX.start();
threadY.start();
threadZ.start();
}
}

```

3.6 Synchronization:

Java programming language permits the user to take this problem testing a technique known as “synchronization”. normally, threads need their individual data and methods arranged inside their run() methods. still if we wish to need data and methods outside the threads run() method, they may attempt for same effects. And be allowed and lead to serious problems.

In the case that Java programming language the key word “synchronized” supports to solve explain said issues. Through observance a watch about similar situation.

Example:

```

synchronized (object identifier)
{
.....// shared variables
.....//synchronized code
}

```

In this example object identifier continue there source continue the things. When the programmer confirm a approach as synchronized, Java programming language build a “monitor” and handed concluded through the thread such commands the approach begining duration. At the time that continued the thread

influence the monitor, at that time not more thread may access the organisecategory of code.

It aboutavailable to point a sectionabout code at the time thatorganizejust as liable down.

synchronized (object identifier)

```
{
.....//access variable
.....//code here is synchronized.
}
```

In this example a thread obtain done mine performance about applying organize approach, it aim hard done with the manitor through the after thread this is primed continue need the related capability.

3.6.1 Implementation of Runnable Interface:

By implementing Runnable interface:

Inthis interface, the user prescribes a group of objects such machine enable attachment. The “**Runnable**” attachment keep particular single approach, run(), this is to be prescribed over the group of things. It involves following steps.

- ❖ Confirm a class that appliance“**Runnable**” interface.
- ❖ Appliance the run () method.
- ❖ Build a thread by explain athing that is instatedaganist this “**runnable**“class.
- ❖ Call the threads start () approach to run the thread.

Example program to create a thread by implementing Runnable interface:

```
class nayan
{
public void run ( )
{
```

```
for (int i=0; i<=10; i++)
{
System.out.println("Thread nayan:"+i);
}
System.out.println(" End of Thread nayan");
}
}
class varshini
{
public static void main(String args[ ])
{
nayan runnable = new varshini( );
Thread threadnayan = new Threadvarshini(runnable);
threadnayan.start( );
System.out.println(" End of main method");
}
}
```

Output:

End of main method

Thread nayan: 0

Thread nayan: 1

Thread nayan: 2

Thread nayan: 3

Thread nayan: 4

Thread nayan: 5

Thread nayan: 6

Thread nayan: 7

Thread nayan: 8

Thread nayan: 9

Thread nayan: 10

End of thread X

3.7 ARRAYS:

It is a group of elements of similar data type such share a natural name. These elements continuere served in adjacent memory situations. Through assign the array factors the user need index or subscript. Every array initialization value starts with zero. Such the index of end element is regularly n-1 where n is the size of array

Syntax for declaring array:

```
type array_name[array size];
```

The nature defines the information description about element. Size illustrates number of elements that an array can hold.

Example:

```
int marks [8];
```

```
float height [10];
```

Advantages of Arrays:

- ❖ Array act efficient about keep several elements by a future.
- ❖ Array concedes casual insinuating about elements applying evidence.

Disadvantages of Arrays:

- ❖ Extent about the array necessary be pre determining.
- ❖ Chance remains there for memory wastage.
- ❖ If the user wants through he liminate an element in the array, the user needs through visit during the array to eliminate single element through the array.

- ❖ If the user wants through insert an element into the array, the users need to visit throughout the array.

Arrays are different types, they are classified into:

Single dimensional Arrays (or) one dimensional array.

Multi dimensional Arrays.

3.7.1 Building an Array

For build about an Array we follow different marks. They are:

- 1) Declaring an Array (or) Declaration of Arrays
- 2) Creation memory locations
- 3) Putting values in to the memory locations

3.7.2 One- dimensional Array or single dimensional array:

One dimensional Array:

An array with only one subscript is known as single dimensional array (or) one dimensional Array.

Declaration of Arrays:

In java programming language arrays may be declared in two ways.

Syntax:

type nameofthearray[]; (or) type[] nameofthe array;

Ex: int a[]; (or) int[] a;

Creating memory locations: After completion of declaring an Array, the user needs through build appeal smart the memory. Java programming language uses us through build arrays applying “new” operant.

Syntax:

Name of the array = new type [sizeofthe array];

Ex: a = new int [5];

It is also possible to combine above two steps (declaration & creation)

Syntax:

Typenameofthearray [] = newtype [sizeofthe array]

Ex: `int a[] = new int[5];`

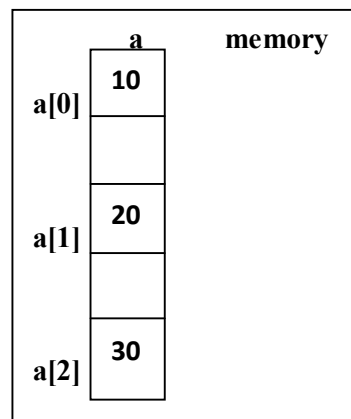
Initialization of Arrays:

In the last stage is to place the values in to the array created. This process is also called as initialization. This is place accepting the array index as given below:

Syntax:

Name of the array [] = value;

Ex: `a[0]=10;`
`a[1]=20;`
`a[2]=30;`
`a[3]=40;`
`a[4]=50;`



we can also log in array naturally, in some use as the routine variables when they are declared.

Syntax:

`type arrayname[] = { list of values };`

Ex: `int a[] = {35, 40, 20, 57, 19};`

We writethearrayand print:

For `(i=0;i<10;i++)`

`System.out.println (x[i]);`

3.7.3 Two- dimensional Arrays

Double dimensional arrays:

Double-dimensional arrays can show the instruction through the scheme about rows and columns i.e., by catching two indexes. Generally, these are used to work with matrices.

In Java programming language, a Double-dimensional array can be shown below:

Syntax:

```
Type[ ][ ] nameofthearray=new type[size][size]
```

Example: `int x[][]=new int[3][6];` (or) `int[][] x=new int[3][6];`

Array initialization done as follows:

```
int a[ ][ ]= {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};
```

here,

1 2

`a[0][0]=10` `a[0][1]=20` `a[0][2]=30`

`a[1][0]=40` `a[1][1]=50` `a[1][2]=60`

`a[2][0]=70` `a[2][1]=80` `a[2][2]=90`

0	10	20	30
1			
2	40	50	60

We write the array and print:

```
for(i=0; i<3; i++)
```

```
{
```

```
System.out.println(" ");
```

```
for(j=0; j<3; j++)
```

```
{
```

```
System.out.print("\t " + a[i][j]);
```

```
}
```

```
}
```

3.8 STRINGS

Strings represent ordering about aspects embedded in double quotation marks (“”). Java provides a class named “String” to handle strings. The “String” class has several built-in functions. Though String is a class, the instantiation of the objects need not be done using the keyword “new”. Strings could be announced and created just as following:

String name of the string;

name of the String=new String(“string”);

The two statements may be combined as follows:

String name of the String =new String(“string”);

String program Example:

String name=new String(“nayan”);

String can also be initialized as follows

String s1=“Hyderabad”;

String objects are not mutable(self changeable)

To check out the string length the user follow “length()” method of String class

Ex: int n=s1.length();

String arrays:

We can also create and use arrays that contain Strings

Ex: String items[]=new String[4];

The above statement create an “item” array of size 4 to hold four String constants

Ex: String s[]={“varshini”, ”nayan”, ”manju”, ”sadguna”};

Example program to find the length of a string.

```
import java.io.*;
class String Length
{
```



```

public static void main(String args[ ])
{
String s1="Java program";
int n;
n=s1.length( );
System.out.println("Length of a String" +n);
}
}

```

Output:

Length of a string 12

Methods of a String:

It explains a sum about an approach such allows us through achieve a variation about string handing works.

Method call	Task to be performed
String to Lowercase()	It restoration the strings, reformed within lowercase
String toUppercase()	It restoration the strings, reformed within Uppercase
String replace ()	It restoration a string imitativeagainstsuch string overrestorationwholesituationaboutinfir Char amongmodern Char.
String trim()	It restoration a modelabouts such string includingbest and tracking white sloteliminated, eithersuch string about it obtain notbest or tracking white slot.
String equals ()	This method restoration true about and particularabouts such String produce the similarflow of characters as the described in StringBuffer, else way false.

String equalsIgnoreCase	This method approaches dual strings lexicographically, omit case diversity.
String length()	It restoration the range about such string.
String charAt(n)	It restoration a char on the described evidence.
String compareTo()	Such approach measure such String to one more Objects.
String concat()	Such approach restoration a string such imitates the integration about such object's characters ensue by the string dispute characters.
String substring()	Allows substring origin from n th character
String substring ()	Allows substring origin from n th character to m th character (not including m th)
String.valueOf()	This approach restoration the string illustration.
String.indexOf()	Allows the location of the first existence

Example program to demonstrate string functions:

```
import java.io.*;
class StringTest
{
public static void main(String args[ ])
{
String s1="abc", s2="xyz";
System.out.println("Length:" +s1.length( ));
System.out.println("Concatenation:" +s1.concat(s2));
System.out.println("Uppercase:" +s1.toUpperCase( ));
System.out.println("Lowercase:" +s1.toLowerCase( ));
System.out.println("Substring(1,2):" +s1.substring(1,2));
System.out.println("Replace:" +s1.replace('c', 'v'));
System.out.println("Character at position 2:" +s1.charAt(2));
```

```

int k= s1.compareTo(s2);
if(k>0)
System.out.println(s1+ "is big");
else if(k<0)
System.out.println(s2 + "is big");
else
System.out.println(s1 + "and" + s2 + "are equal");
if(s1.equals(s2))
System.out.println(s1 + "and" + s2 + "are equal");
else
System.out.println(s1 + "and" + s2 + "are not equal");
if(s1.equalsIgnoreCase(s2))
System.out.println(s1 + "and" + s2 + "are equal");
else
System.out.println(s1 + "and" + s2 + "are not equal");
}
}

```

Output:

Length: 3

Concatenation: abcxyz

Uppercase: ABC

Lowercase: abc

Substring(1,2): a

Replace: abv

Character at position 2: b

xyz is big

abc and xyz are not equal

abc and xyz are not equal

StringBuffer class:

Peer class of string is also known as string buffer class. During the “**String**” builds strings about secure length. It builds flexible length strings and such could be converted in details about the pair length and appeared. The user could include characters into or delete characters from a string.

- ❖ String buffer objects are mutable (self changeable)
- ❖ It can be treated as dynamic string

String buffer can be initialized as follows”

StringBuffer s1=new StringBuffer(“abcd”);

Method	Task
append()	Used to concatenate the string in string buffer
insert()	Used to insert any string at the specified position in the given string
reverse()	Used to reverse the string present in string buffer
setCharAt()	Used to set specified character in buffered string at specified position
delCharAt()	Used to delete the specified character at a given position from the buffered string

Example:

string1.append(s2) → concatenates s1 & s2 strings

string1.insert(n,s2) → inserts the string s2 at nth position of s1

string1.reverse() → reverses the string s1

string1.setCharAt(n,'x') → modifies the nth character to x

string1.delCharAt(n) → deletes the character at nth position of s1

3.9 VECTORS

“`java.util`” package is involved in the vectors. It could be worn to build a wide effective array is known as “vector” such could grip things about each from and each count. Vectors are build like arrays as given below):

```
Vector list=new Vector (); // declaring without size
```

```
Vector list=new Vector (3); // declaring with size
```

Advantages of Vectors:

- ❖ Vectors to reserve things.
- ❖ It could be worn to reserve an index of things such could modify in extent.
- ❖ The user could enumerate and destroy things such the index as and although needed.
- ❖ The Vector class staves a number of approaches such could be worn to manage the vectors build.

Method	Task performed
Void addElement()	Enumerates the described composing through the edge about such vector, developing their extent by single.
Void elementAt()	Specified the composing through the described number about such vector to endure the described object.
Void size()	Restoration the index of Constituent in such vector.
Void removeElement()	Restoration entire constituent against such vector and specified its extent to zero.
Void removeElementAt()	Eliminates the feature reserved area about the index
removeAllElements()	Eliminates the fact through the described reference in such vector.

Example:

```
import java.util.*;
class VectorTest
{
public static void main(String args[ ])
{
Vector v1=new Vector(10);
v1.add("Apple");
v1.add("Banana");
v1.add("Mango");
v1.insertElement("Lemon",2);
v1.removeElementAt(3);
System.out.println("Number of elements:" +v1.size( ));
System.out.println("Vector elements:" +v1);
}
}
```

Output:

```
Number of elements: 3
Vector elements: {Apple, Banana, Lemon}
```

3.10 WRAPPER CLASSES

These classes can be used to convert (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class number.

- ❖ These classes contained in the “java.lang” package.
- ❖ The entire wrapper classes contained public final, i.e, cannot be extended.

Converting Primitive numbers to Object numbers using constructor methods, we write:

```
Integer Intval=new Integer (i); // converts integer to integer object
```

```
Float Float val=new Float (f); // converts float to float object
```

```
Double Dval=new Double(d); //converts double to double object
```

```
Long Lval=new Long(l); //converts long to long object
```

Converting object numbers to primitive numbers using type Value () method, we write:

```
int i= Intval.intValue(); // converts object to primitive integer
```

```
float f= Floatval.floatValue(); // converts object to primitive float
```

```
double d= Dval.doubleValue(); // converts object to primitive double
```

```
long L=Lval.longValue(); // converts object to primitive long
```

Converting numbers to Strings using to String () method:

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

```
str= Float.toString( );
```

```
str= Double.toString( );
```

```
str= Long.toString( );
```

Converting string objects to Numeric objects using ValueOf() method, we write:

```
Dval= Double.ValueOf(str);
```

```
Floatval= Float.ValueOf(str);
```

```
Intval= Integer.ValueOf(str);
```

```
Lval= Long.ValueOf(str);
```

Converting numeric strings to primitive numbers using parsing methods, we write:

```
int i= Integer.parseInt(str);
```

```
long l=Long.parseLong(str);
```

3.11. Summary:

This unit has provided a detailed introduction to the Java programming language. As Java is an object oriented language, the concept of classes is very important to understand and this has been presented in the unit in the various sections. The concept this unit is to provide basic insight into the core features of java programming language. Preface to Classes, Confirming Classes, Building an Object - Methods – Overloading methods, overloading constructors, Access Control Specifiers, concept of Static and Abstract (Simple application based examples). This unit also intends to illustrate key concepts through easy to understand examples for enhancing the understanding the reader. Java is a powerful programming language and its basics have been covered here. However, in order to gain proficiency in the language it is important to write lots of code in the language and practice regularly.

3.12. Exercise:

1. Write a program about object
2. Write a program about class
3. Write a simple constructor java program
4. Write a java program using overloading different number of parameters in argument list.
5. Write a java program using overloading different in data type of parameters.
6. Write a java program using overloading sequence of data type of parameters.
7. Write a program to sum values of an array using java.
8. Write a program to find the index of an array element using java.
9. Write a program to convert an array to array list using java.
10. Write a program to find a missing number in an array using java.

3.13. Multiple Choice Questions:

1. Which of this class is superclass of every class in Java?
a) Method class b) Object class c) class d) None

Ans: **B**

2. Which of these can be overloaded?
a) Methods b) Constructors c) Both a & b d) None

Ans: **C**

3. Which of these is correct about passing an argument by call-by-value process?
a) Copy of argument is made into the formal parameter of the subroutine
b) Reference to original argument is passed to formal parameter of the subroutine
c) Copy of argument is made into the formal parameter of the subroutine and changes made on parameters of subroutine have effect on original argument
d) Reference to original argument is passed to formal parameter of the subroutine and changes made on parameters of subroutine have effect on original argument

Ans: **A**

4. What is process of defining two or more methods within same class that have same name but different parameters declaration?

- a) method overloading b) method overriding
c) method hiding d) None

Ans: **A**

5. Method overloading is one of the ways that Java supports

- A) Encapsulation B) Class C) Inheritance D) Polymorphism

Ans: **D**

6. What is true about private constructor?

- a) Private constructor ensures only one instance of a class exist at any point of time
- b) Private constructor ensures multiple instances of a class exist at any point of time
- c) Private constructor eases the instantiation of a class
- d) Private constructor allows creating objects in other classes

Ans: **A**

7. What would be the behavior if this () and super () used in a method?

- a) Runtime error
- b) Throws exception
- c) compile time error
- d) Runs successfully

Ans: **C**

8. What is false about constructor?

- a) Constructors cannot be synchronized in Java
- b) Java does not provide default copy constructor
- c) Constructor can be overloaded
- d) “this” and “super” can be used in a constructor

Ans: **C**

9. What is true about Class.getInstance ()?

- a) Class.getInstance calls the constructor
- b) Class.getInstance is same as new operator
- c) Class.getInstance needs to have matching constructor
- d) Class.getInstance creates object if class does not have any constructor

Ans: **D**

10. What is true about constructor?

- a) It can contain return type
- b) It can take any number of parameters
- c) It can have any non access modifiers
- d) Constructor cannot throw exception

Ans: **B**

11. Abstract class cannot have a constructor.

- a) True
- b) False

Ans: **B**

12. What is true about protected constructor?

- a) Protected constructor can be called directly
- b) Protected constructor can only be called using super()
- c) Protected constructor can be used outside package
- d) protected constructor can be instantiated even if child is in a different package

Ans: **B**

13. What is not the use of “this” keyword in Java?

- a) Passing itself to another method
- b) Calling another constructor in constructor chaining
- c) Referring to the instance variable when local variable has the same name
- d) Passing itself to method of the same class

Ans: **D**

14. Which of these is used to access member of class before object of that class is created?

- a) public b) private c) static d) protected

Ans: **C**

15. Peer class of string is also known as

- a) string buffer class b) Class c) Method d) None

Ans: **A**

16. Class is a group of _____ that have the same properties

- a) Methods b) objects c) Classes d) None

Ans: **B**

17. The constructor with no parameters is called as

- a) Default Constructor b) Constructor c) Method d) None

Ans: **A**

18. The constructor with parameters is called as

- a) parameterized constructor
b) Default Constructor
c) Constructor
d) None

Ans: **A**

19. Static variables can only be declared at the

- a) class level b) Method level c) Semi level d) None

Ans: **A**

20. Static methods can not call

- a) None
- b) Static Methods
- c) Method
- d) Non-Static Methods

Ans: **D**

21. An abstract class may have both

- a) Normal Method
- b) abstract method only
- c) abstract methods and normal methods
- d) None

Ans: **C**

22. Normal programs contain only a single sequential flow of control these are called

- a) single-threaded programs
- b) multi-threaded programs
- c) Hybrid-threaded programs
- d) None

Ans: **A**

23. Strings represent sequence of

- a) Characters
- b) Numbers
- c) Strings
- d) None

Ans: **A**

24. An _____ is a group of contiguous data items that share a common name

- a) String b) Array c) Method d) None

Ans: **B**

25. _____ represents a sequence of characters.

- a) String b) Array c) Method d) None

Ans: **A**

Unit-IV

4.0 Aims and Objectives	208
4.1 Introduction	208
4.2 Inheritance	208
4.2.1 Extending a class	209
4.2.2 Overloading methods	219
4.3 Final variables and methods	220
4.3.1 Final classes	222
4.4 Abstract methods and classes	222
4.4.1 Member access using super class	226
4.4.2 Member access using abstract classes	230
4.4.3 Call by value	230
4.4.4 Call by reference,	231
4.4.5 Overriding methods	231
4.5 Applets	234
4.5.1 Introduction	234
4.5.2 Types	234
4.5.2.1 local and remote applets	234
4.5.3 Applets and Applications	234
4.5.4 Building Applet code	234
4.6 Applet Life cycle	235
4.6.1 Initialization state	235
4.6.2 Running state	236
4.6.3 Idle or stopped state	236
4.6.4 Dead state	236
4.6.5 Display state	237
4.7 Summary	238
4.8 Exercise	239
4.9 Objective Type Questions	240

4.0 AIMS AND OBJECTIVES:

The aim of the unit is to cover fundamental concept of java programming language in depth. Java is a cross platform, general purpose programming language commonly used in critical applications such as banking systems. Objective of this unit is to provide basic insight into the core features of java programming language. Such as inheritance, exception handling, interfaces, abstract classes and applet programming. This unit also intends to illustrate key concepts through easy to understand examples for enhancing the understanding the reader.

4.1 Introduction:

The Java programming language relies heavily on the concept of Inheritance, in which a new class can inherit or acquire the attributes and functionalities of a previous base class which allows programmers to reuse existing code.

Java also supports applets which enables the execution of Java code on web browsers that support this feature.

4.2 Inheritance:

In Java, it is possible for one class to acquire the properties and methods of another class. This is referred to as inheritance. A group of objects could isolated be inherited against a particular class, whereas a single class can be used by multiple classes for inheritance. The derived classes are known as subclasses as well as the class against whichever the inheritance is done is also known as the super class. As well as keyword which is used to achieve inheritance is the ‘extends’ keyword.

4.2.1 Extending a Class:

Inheritance is made possible in Java through the use of 'extends' keyword. This allows a new class to obtain as well as substance and functionalities of their base class.

Syntax:

```
Class A_subclass extends B_superclass
```

```
{
//additional fields and methods since a subclass which act not either present in
the super class
}
```

If the user wants to derive a subclass Son from a super class Father, you can do it as follows:

```
Class Son extends Father {...}
```

Advantages of inheritance:

- ❖ No need to write code from scratch. You can start coding with existing class
- ❖ Through inheritance you can very easily convert small system into large systems
- ❖ Code reusability through inheritance increased.
- ❖ Good at representing the objects
- ❖ Inheritance arrange a fair perfect architecture whichever is simple to recognized
- ❖ Code is simple to maintain and split into parent and child classes.

Properties of Inheritance:

- ❖ Irrespective of the package in which the subclass resides, all the protected and public members are inherited by the subclass. If both the subclass and the super class are present in the same package, private members are also inherited by the subclass. The inherited members of a class can be either modified or be used without any modification.
- ❖ The fields which are inherited from the super class can be used in the same manner as any other fields of the class.
- ❖ The field inherited from super class can be overridden and hidden by declaring a field of the same name in the subclass.
- ❖ In the subclass, additional fields can be declared that are not a part of the parent class.
- ❖ This inheritance methods used directly as they are.
- ❖ It is possible to compose a modernexampleapproach in the subclass a certaininclude the similarimpression as the particular in the super class. This is referred to as overriding.
- ❖ A modernfixedapproach in the subclass this has the similar impression as the particular in the super class may be defined. This will result in hiding of the method in the super class.
- ❖ Modern methods in the subclass may be declared which are not in the super class.
- ❖ A subclass constructor can be written which request as well asproducer of the super class, this oneessentially or away proving the keyword super.

Deriving a sub class:

In Java programming language, the programmer could assume of a subclass of a real class providing the 'extends' keyword. Here particular essential thing in java inheritance is that a class could develop individual single super class.

Syntax:

```
class subclass_name extends superclass_name
{
// Declaration of the variable
//Declaration of the method
}
```

Position of the class, develops the keywords.

Example:

```
class one
{
declarations
}
class two extends one
{
declarations
}
```

Types of Inheritance:

Inheritance can be classified into different types. They are

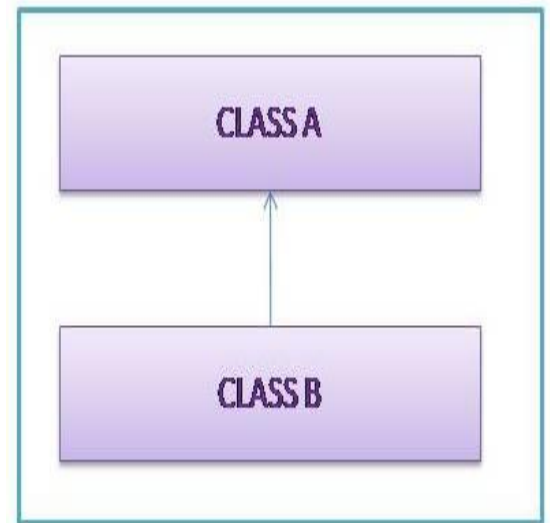
(1) Simple / Single Inheritance:

Such inheritance is when only a single subclass is derived from a super class.

Example program of Single Inheritance

Name of the File: InheritanceTest.java

```
class animal
{
void eat( )
{
System.out.println("eating...");
}
}
class cow extends animal
{
void moo( )
{
System.out.println("moo...");
}
}
class TestInheritance {
public static void main(String args[])
{
cow d=new cow();
d.moo();
d.eat();
}
}
```



Output:

moo...

eating...

In the above program, the output is produced by two statements: 'd.moo()' and 'd.eat()'.

'd.moo()' is a call to the 'moo()' function that is part of the class 'cow' of which the 'd' object is an instance of.

On the other hand, 'd.eat()' is a call to the 'eat()' function that is part of the class 'animal', which the 'd' object is not directly a part of but it derives it from base class 'animal'. This is an example of single inheritance.

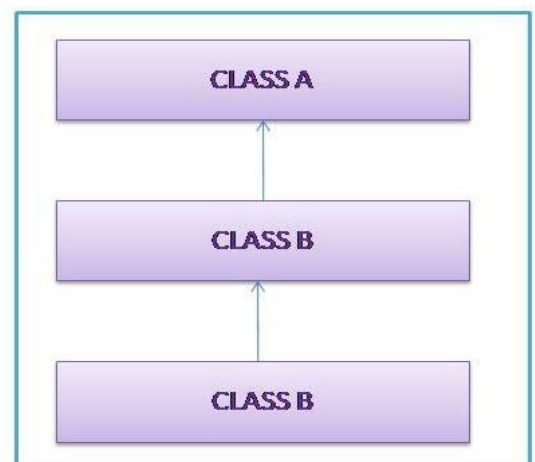
(2) Multilevel Inheritance:

While a subclass abides against a class which itself has been derived from some class, next that structure is called as the multilevel inheritance. Such multilevel inheritance can have any number of levels.

Example program of Multilevel Inheritance:

Name of the File: Inheritance2Test.java

```
class animal
{
void eat( )
{
System.out.println("eating..");
}
}
class moo extends animal{
void moo( )
```



```
{
System.out.println("moo...");
}
}
class babycow extends moo{
void weep( )
{
System.out.println("weeping...");
}
}
class TestInheritance2{
public static void main(String args[])
{
babycow d=new babycow();
d.weep();
d.moo();
d.eat();
}
}
```

Output:

weeping...

moo...

eating...

In the above program, the output is produced by three statements: ‘d.weep()’, ‘d.moo()’ and ‘d.eat()’.

‘d.weep()’ is a call to the ‘weep()’ function that is part of the class ‘babycow’ of which the ‘d’ object is an instance of.

'd.moo()' is a call to the 'moo()' function that is part of the class 'cow', which the 'd' object is not directly a part of but it derives it from base class 'eat'.

'd.eat()' is a call to the 'eat()' function that is part of the class 'animal', which the 'd' object is not directly a part of but it derives it from base class 'animal'. The class animal is not derived directly but is derived from the class 'cow'. This is an example of multiple inheritances.

3) Hierarchical inheritance:

More than one sub class is derived from a super class is also known as Hierarchical Inheritance.

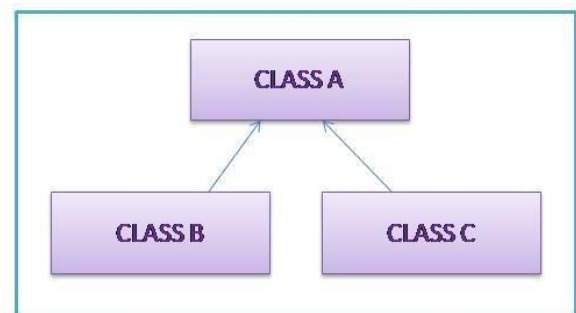
Example Program of Hierarchical Inheritance:

Inheritance3Test.java

```
class animal
{
void eat( )
{
System.out.println("eating...");
}
}

class moo extends animal{
void moo( )
{
System.out.println("moo...");
}
}

class lion extends animal{
void roar( )
```



```

{
System.out.println("roar...");
}
}
class TestInheritance3{
public static void main(String args[])
{
lion c=new lion();
c.roar();
c.eat();
//c.moo();//C.T.Error
}
}

```

Output:

roar...

eating...

In the above program, the output is produced by two statements: ‘c.roar()’ and ‘c.eat()’.

‘c.roar()’ is a call to the ‘roar()’ function that is part of the class ‘lion’ of which the ‘c’ object is an instance of.

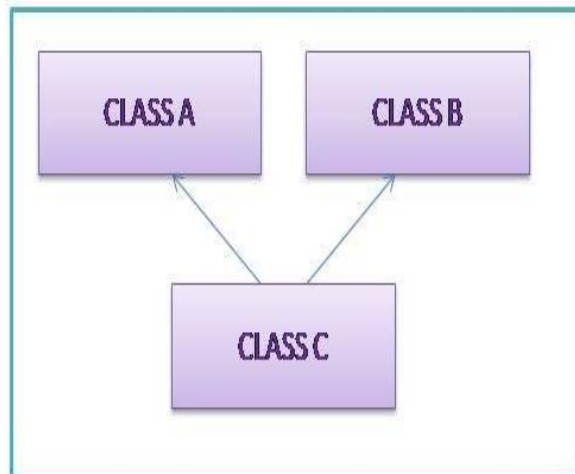
On the other hand, ‘c.eat()’ is a call to the ‘eat()’ function that is part of the class ‘animal’, which the ‘c’ object is not directly a part of but it derives it from base class ‘animal’.

The line ‘c.moo()’ produces an error as the ‘moo()’ function is part of class ‘cow’ and it is not derived by the class ‘lion’ of which the object ‘c’ is an instance of.

This is an example of hierarchical inheritance.

4) Multiple Inheritances:

While a subclass actimitativeagainstfurther then particular super class later it is often referred to as multiple inheritances. Java does not support multiple inheritances. Interfaces can be used to achieve multiple inheritances in java.



5) Hybrid inheritance:

A sequence of multilevel and hierarchical inheritance is also called as hybrid inheritance.

Example Program of Hybrid Inheritance:

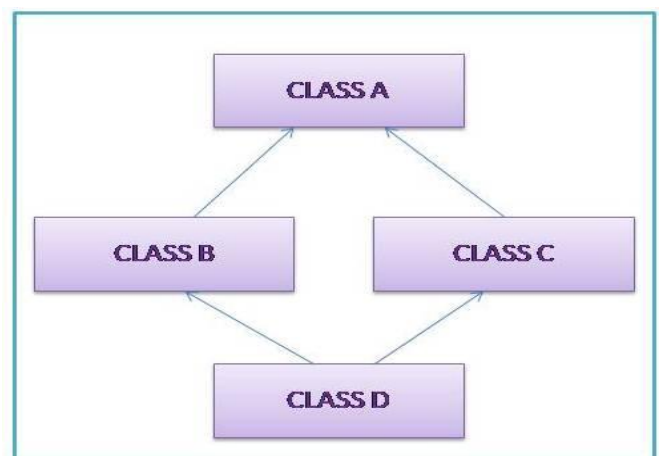
Class P and Q extends class R → Hierarchical inheritance

Class S extends class X → Single inheritance

```

class R
{
public void disp()
{
System.out.println("R");
}
}
class X extends R
{

```



```
public void disp()
{
System.out.println("X");
}
}
class Q extends R
{
public void disp()
{
System.out.println("Q");
}
}
class S extends X
{
public void disp()
{
System.out.println("S");
}
public static void main(String args[])
{
S obj = new S();
obj.disp();
}
}
```

Output:

S

Here the output produced is 'S' because the object is an instance of class 'D' and in class 'D' the 'disp()' function is defined to print 'D'.

4.2.2 OVERLOADING METHODS:

Overloading is the concept of the same class having multiple functions amongcomicsimilar name althoughvarious arguments. Which function is executed build uponaboveact arguments supplied to function at run time.

Example Program of Overloading:

OverloadingExample.java

```
class AddNumbers
{
int Multiplication(int x, int y )
{
return x * y;
}
int Multiplication(int x, int y, int z)
{
return x * y * z;
}
}
class OverloadingExample
{
public static void main(String args[])
{
AddNumbers x = new AddNumbers();
int q = p.multiplication(1, 2);
System.out.println(q);
int r = p.multiplication(1, 2, 3);
System.out.println(q);
}
```

}

Output:

2

6

Here the output 2 is produced by the first add function and output 6 is produced by the second add function even though they have the same name.

4.3 FINAL VARIABLES AND METHODS:

A final variable is a variable declared with the final keyword. Whenever a variable is declared with the final keyword, it's value becomes fixed and cannot be updated or changed and the variable can never be reassigned.

Similarly a final method is a function declared with final keyword. Whenever a function or method is declared as final, it cannot be overridden in a derived class.

Final Keyword Example:

FinalKeyword.java

public class shape

{

final int shape = 3;

final void getShape()

{

System.out.println(shape);

}

void changeShape()

{

// shape = 4; Not Allowed

}

}

```
public class NewShape extends shape
{
int new_shape = 6;
// void getShape() {} Not Allowed
void getNewShape()
{
System.out.println(new_shape);
}

}
class FinalKeyword
{
public static void main(String args[])
{
shape s = new shape();
s.getShape();
NewShape ns = new NewShape();
ns.getNewShape();
}
}
```

Output:

3

6

Here the two commented lines are not allowed as final variables cannot be reassigned and final methods cannot be overridden.

4.3.1 FINAL CLASSES:

A final class is one that cannot be extended by another class.

Final Class Example

FinalClass.java

```
final class cow
{
void moo()
{
System.out.println("moo...");
}
}
// public class BabyCow extends cow {} -> Not Allowed
```

Explanation of the program:

Here the class cow is declared final so it cannot be extended.

4.4 ABSTRACT METHODS AND CLASSES

Abstract Methods:

In fact cases where approaches accept a part been declared and not been defined are called abstract approaches. The keyword 'abstract' is worn to maintain such approaches. The declaration of a method which is type abstract should always end in a semi colon.

Syntax:

```
accessspecifier abstract return_type nameof_method(arguments);
```

Example:

```
abstract void eat();
```


Abstract class:

A class which cannot be instantiated is also known as abstract class. A class is made abstract by using the keyword `abstract` after the access specified. It is not necessary for an abstract class to have abstract methods.

Syntax:

```
AccessSpecifier abstract class My Class
{
//Any fields
//Any methods-may or may not be abstract
}
```

Some key points regarding abstract methods and classes:

- ❖ A class which consist of single or multiple abstract approaches is abstract of itself.
- ❖ Any class which is made abstract cannot be instantiated in any other class.
- ❖ Whenever a subclass of an abstract class overrides complete its abstract approaches and provide implementation for each of those methods, then that subclass can be instantiated. Such classes are often referred to as concrete classes. This is done to maintain the evidenceand certain they are no more abstract.
- ❖ All the subclasses about any abstract class are abstract themselves if they do no morevokeentire the abstract methods inherited from the superclass.
- ❖ The methods which can never be abstract are - static, private, and final methods. This is due to the fact that such methods can never be overridden in a subclass. In the same manner, it is not possible for a final class to have abstract approaches.

For a class to be announced as abstract, it is no more necessary for that class to include several abstract approaches. The purpose behind confirming similar a class is to indicate that the performance is not fully complete. In such cases, a subclass is meant to complete the implementation. Therefore abstract classes can never be instantiated.

Example:

```

abstract class my_shape                                //Abstract Class
{
Double pi=3.14;
abstract void calculate_area();                       //Abstract Method
Void display()
{
System.out.println (“\n Non_abstract method of class shape”);
}
}
class my_rectangle extends my_shape
{
Int l,b;
My_rectangle (int m, int n)
{
L=m;
B=n;
}
void area()                                           //Implementing abstract method
{
System.out.println("\nAfter calculations (for rectangle) :");
System.out.println("Length is"+l);
}
}

```

```

System.out.println("Breadthis"+b);
System.out.println("Area of the rectangle is "+(l*b));
}
}
Classmy_circle extends my_shape //sub class
{
Double r;
My_circle(double k)
{
R=k;
}
void area() //Implementation of the abstract method
{
System.out.println("\After calculations (for circle) :");
System.out.println("Radius is"+r);
System.out.println("Area is"+(pi*r*r));
}
}
/* Main class */
Classabst
{
public static void main(String args[])
{
my_shape s;
my_rectangle r=newmy_rectangle(10,5);
s=r;
s.area();
}
}

```

```

my_circle c=newmy_circle(2.5);
c.area();
}
}

```

Output:

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

```
D:\java>java abst
```

After calculations (for rectangle):

Length is 10

Breadth is 5

After calculations (for circle):

Radius is 2.5

Area is 19.625.

Here the function ‘area()’ is a part of the base class, however it is not implemented there. It has to be implemented accordingly in the classes where it has been extended like rectangle and circle. Also, the class my_shape cannot be directly instantiated.

4.4.1 MEMBER ACCESS USING SUPER CLASS:**Super keyword:**

The super is a keyword defined in the java programming language. Some important points regarding the super keyword:

- ❖ This could be used for calling a approachaboutact super class from the base class, which has been overridden in the base class.
- ❖ This could also be worn for province the constructor of the super class.

The syntax for calling the constructor of the super class using the keyword ‘super’:

```
Class sub_class extends super_class
{
Sub_class(args)
{
Super(args);
//statements;
}
-----
}
```

When used as a standalone statement to call the superclass constructor the super should be the first statement within the subclass constructor.

The syntax to call method of super class using super:

```
super.<method_Name>(args) ;
```

Whenever there is a need to call a method of superclass from the base class which has been overridden, super keyword can be used as shown below example.

Example:

```
/* Use of Super*/
Class one                                     //Super Class
{
Int a,b;
One()
{
System.out.println (“inside the constructor of class one”);
}
One (int x, int y)                             //Overloaded constructor
```

```

{
system.out.println("constructor of class one called using super");
a=x;
b=y;
}
Void display()                                // Overridden method
{
System.out.println("Inside method of class one");
System.out.println("a+=a);
System.out.println("b+=b);
}
}
Class two extends one                          // sub class
{
Int c;
Two(int x, int y, int z)
{
Super(x,y);                                    //Calling super class constructor
System.out.println("Inside the constructor of class two");
C=z;
}
Void display()                                //Overriden Method
{
System.out.println("Inside method of class two");
Super.display();                              //Calling super class method
System.out.println("c="+c);
}
}

```

```

}
/* Main class */
Classsupercall
{
Public static void main(String args[])
{
System.out.println("Creating instance of class two\n");
Two t=new two(10, 20, 30);           //creating instance of class
Two
t.display();
}
}

```

Output:

D:\java>javac supercall.java

D:\java>java supercall

Creating instance of class two

Constructor of class one called using super

Inside the constructor of class two

Inside method of class two

Inside method of class one

a=10

b=20

c=30

4.4.2 MEMBER ACCESS USING ABSTRACT CLASSES:

The member of abstract class cannot be directly accessed as the class cannot be directly instantiated. It has to access through a class that extends it.

4.4.3 CALL BY VALUE:

Java is always call by value and never call by reference. In the case of primitive data type such as int, the actual value of the variable is passed whereas in case of objects the value of the reference to the object is passed.

In case of call by value, a new copy of the variable is created and that is passed to the function. Any change to the variable inside the function does not reflect to the original variable that was passed.

Example:

Following example illustrates passing of value of primitive data type variable to a method.

```
class MyClass
{
Int y=500;
Void changeValue(int y)
{
Y=y+100;           only the value of the local variable is changed
}
Public static void main (String args[])
{
MyClassobj=newMyClass();
system.out.println("Before changing the value:"+obj.y);
obj.changevalue(500);
```



```

system.out.println("After changing the value:"+obj.y);
}
}

```

Output:

Before changing the value: 500

After changing the value: 500

4.4.4 CALL BY REFERENCE:

In case of call by reference, a new copy of the variable is not created and the same variable is passed to the function. Any change to the variable inside the function will reflect on the original variable that was passed.

In Java, in case of objects the value of the reference to the object is passed.

4.4.5 OVERRIDING METHOD:

- ❖ If a approachusual a subclass includeact similar name alsoformimpressionjust as a approachusualtheirs superclass, againactapproachusualact subclass doforenamed to revokeactapproachusualact superclass.
- ❖ If a approach which has been overridden is called from the subclass, then the version of the method which has been newly defined will called.
- ❖ The implementation details about the approachdescribedover the superclass will be of no use in subclass as it will be hidden, if that method has been overridden by the superclass.
- ❖ For overriding a method, the name as well as the type signatures should match. Whereverit is do not match, again the doubleapproaches will be overloaded.

Example:

```

/* Method overriding example */
class one                                //Super class
{
void calculate(intx,int y)
{
System.out.println("Class one");
System.out.println("X="+x+"\nY="+y+"\nX+Y="+(x+y));
}
}
class two extends one    //sub class
{
void calculate(intx,int y)                //Overridden method
{
System.out.println("Class two overridden method");
System.out.println("X="+x+"\nY="+y+"\nX*Y="+(x*y));
}
void calculate(float x,float y)          //Overloaded method
{
System.out.println("Class two overloaded method");
System.out.println("X="+x+"\nY="+y+"\nX*Y="+(x*y));
}
}
/* Main Class */
class override

```

```

{
public static void main(String args[])
{
two t=new two();                //Creating object of class two
t.calculate(6,7);
t.calculate(2.5f,3.2f);
one o=new one();                //Creating object of class one
o.calculate(6,7);
}
}

```

Output:

D:\Java>javac override.java

D:\Java>java override

Class two overrided method

X=6

Y=7

X*Y=42

Class two overloaded method

X=2.5

Y=3.2

X*Y=8.0

Class one

X=6

Y=7

X+Y=13

In the above example the calculate (int, int) method of the class one has been overrided by the class two. So in order to access the calculate method of class one we need to create an object for the class one.

If you wish to access the super class version of an overridden method from within the subclass, you can do so by using super Keyword.

4.5 APPLETS

4.5.1 INTRODUCTION

An applet is a web-based program written in Java. An applet is most often embedded in a HTML page. This HTML page is then viewed in a browser (For example: Chrome, Firefox). An applet can be downloaded by any computer.

4.5.2 TYPES OF APPLETS:

Local applet:

When an applet is stored locally in our system, it is referred to as a local applet. Such an applet can be viewed in the browser even without accessing the internet.

Remote applet:

An applet which has been stored in a remote server and is accessed through an internet connection is referred to as a remote applet.

4.5.3 Applets and Applications:

The following are the situations to use applets.

- ❖ To obtain dynamic changes from web page.
- ❖ To provide GUI for remote user.
- ❖ To make the programs available through the internet.

4.5.4 Building an applet code:

```
public class AppletSkel extends Applet
{
public void init()
{
}
```

```
public void start()
{
}
public void stop()
{
}
public void destroy()
{
}
```

```
public void paint(Graphics g)
{
}
}
```

4.6 Applet life cycle:

The applet lifecycle includes the following methods:

4.6.1 Initiation State:

`init()`: This method is used for the purpose of the initialization of an applet. Lifecycle of an applet begins when the `init()` method is called and the browser finishes loading all the applet's classes. Therefore, inside the `init()` method the variables should be initialized. After the process of initialization is finished, the browser automatically calls the `start()` method. After this method is called the user is able to interact with the applet.

4.6.2 Running State:

`start()`:This method is called after the initialization method has been called.In case the user again maximizes the window or moves back to the applet after browsing other WebPages, the browser automatically calls the applet's `start()` method. Hence, the user can again start interacting with applet.

4.6.3 Idle or stopped state:

`stop()`:In order to stop the applet, this method is used. It can be called any number of times in the whole lifecycle of an applet.

The `stop()` method is called by the browser whenever the window is minimized or the user starts browsing some other web page. This is very useful as it ensures that the applet does not consume any extra resources while the user is not using it.

`destroy()`:This method is automatically called when we close the applet. It can only be called once during the whole lifecycle of the applet.

It is necessary to kill the applet before it closes to ensure that no system resources are being consumed after the applet has been closed. The method which is uses to do so is the `destroy ()` method.

4.6.4 Dead State:

An applet enters the dead state when it is removed from the memory. This is done by overriding the `destroy()` method. `stop()` must always be destroyed before `destroy()`.

4.6.5 Display State:

It is possible to override any of the applet's lifecycle methods. This can be done to include your own custom implementation of these methods. For example, `stop ()` method can be used to clean up some used space.

There may be cases when there is a need to clean up some threads which were started during the initialization of the applet. This can be done by overriding the `destroy ()` method.

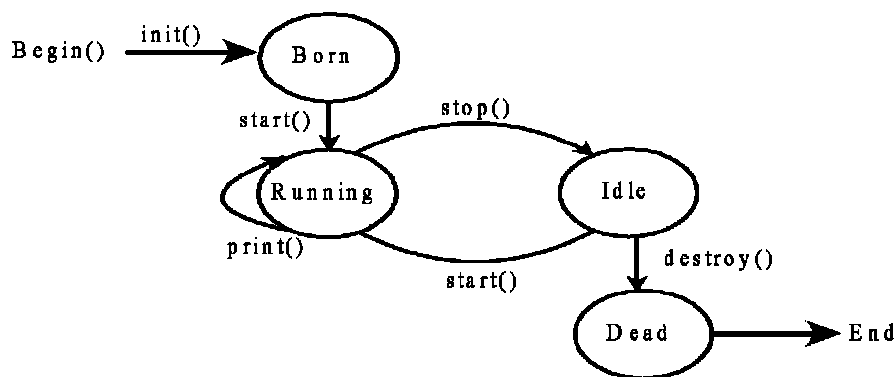


Figure: Applet Life Cycle

Difference between Applets and Applications:

Even though both stand-alone applications as well as applets are java programs, they differ in many ways. Some key differences are discussed below:

- ❖ Unlike stand-alone applications, applets do not use the `main()` method. Certain methods such as `init()`, `start()` and `destroy()` are automatically called during the lifecycle of an applet.
- ❖ Applets are executed in a browser(For example : Firefox, Chrome) and are embedded inside an HTML page. This is not the case with stand-alone applications.
- ❖ In order to be secure, applets have some restrictions imposed on them. Applets are not allowed to access the files which are present locally on the user's computer.

- ❖ Applets are not allowed to communicate in any manner with other servers present in the network.
- ❖ Applets are restricted to run any programs on the local system.
- ❖ Applets are not allowed to use any library which is written in other languages.

The restrictions imposed on applets ensure security and prevents any malicious applet code from doing any harm.

4.7 Summary:

This unit has provided a detailed introduction to the Java programming language. As Java is an object oriented language, the concept of classes is very important to understand and this has been presented in the unit in the various sections. The concept of inheritance is used extensively in Java and different types of inheritance have been explained like multiple, single, multilevel, hierarchical and hybrid inheritance. Use of final keyword, abstract methods and classes is also explained. Applets are a way to run Java on browsers that support it. It's usage along with the lifecycle has been presented.

Java is a powerful programming language and it's basics have been covered here. However, in order to gain proficiency in the language it is important to write lots of code in the language and practice regularly.

4.8 EXERCISE:

Q.1 What is inheritance?

Q.2 What is the difference between overriding and overloading?

Q.3 What are applets?

Q.4 Explain the applet life cycle.

Q.5 What is the difference between call by value and call by reference?

Q.6 What are abstract classes?

Q.7 Explain the use of final keyword with respect to classes, variables and functions respectively.

Q.8 How does Java handle inheritance?

Q.9 What are the different types of inheritance in Java?

Multiple Choice Questions:

1. How many abstract classes can a single program contain?
a) At most 1 b) At least 1 c) At most 127 d) As many as required

Ans: **D**

2. Which of the following statements is valid about abstract classes?
a) If a class has more than one virtual function, it's abstract class
b) If a class have only one pure virtual function, it's abstract class
c) If a class has at least one pure virtual function, it's abstract class
d) If a class has all the pure virtual functions only, then it's abstract class

Ans: **C**

3. Can abstract class have main() function defined inside it?
a) Yes, depending on return type of main() b) Yes, always
c) No, main must not be defined inside abstract class
d) No, because main() is not abstract function

Ans: **B**

4. If a class contains an abstract class, then, _____
a) Class must be abstract class b) Class may or may not be abstract class
c) Class is generic d) Class must be public

Ans: **A**

5. If a class is extending/inheriting any other abstract class having abstract method, then _____
a) Either implementation of method or making class abstract is mandatory
b) Implementation of the method in derived class is mandatory
c) Making the derived class also abstract is mandatory
d) It's not mandatory to implement the abstract method of parent class

Ans: **A**

6. Abstract class A has 4 virtual functions. Abstract class B defines only 2 of those member functions as it extends class A. Class C extends class B and implements the other two member functions of class A. Choose the correct option below.

- a) Program won't run as all the methods are not defined by B
- b) Program won't run as C is not inheriting A directly
- c) Program won't run as multiple inheritance is used
- d) Program runs correctly

Ans: **D**

7. Abstract classes can _____ instances.

- a) Never have
- b) Always have
- c) Have array of
- d) Have pointer of

Ans: **A**

8. We _____ to an abstract class.

- a) Can create pointers
- b) Can create references
- c) Can create pointers or references
- d) Can't create any reference, pointer or instance

Ans: **C**

9. Which among the following is an essential use of abstract classes?

- a) Header files
- b) Class Libraries
- c) Class definitions
- d) Class inheritance

Ans: **B**

10. Use of pointers or reference to an abstract class gives rise to which among the following feature?

- a) Static Polymorphism
- b) Run time polymorphism
- c) Compile time Polymorphism
- d) Polymorphism within methods

Ans: **B**

11. The abstract classes in java can _____

- a) Implement constructors
- b) Can't implement constructor
- c) Can implement only unimplemented methods
- d) Can't implement any type of constructor

Ans: **A**

12. Abstract class can't be final in java.

- a) True
- b) False

Ans: **A**

13. Can abstract classes contain static methods (Java)?

- a) Yes, always
- b) Yes, but depends on code
- c) No, never
- d) No, static members can't have different values

Ans: **A**

14. It is _____ to have an abstract method.

- a) Not mandatory for an static class
- b) Not mandatory for a derived class
- c) Not mandatory for an abstract class
- d) Not mandatory for parent class

Ans: **C**

15. The derived classes are known as

- a) Sub classes
- b) Super class
- c) Multi class
- d) None

Ans: **A**

16. The class from which the inheritance is done is called the

- a) Sub classes
- b) Super class
- c) Multi class
- d) None

Ans: **B**

17. The keyword which is used to achieve inheritance is the _____ keyword.

- a) 'extends'
- b) *extend*
- c) extend
- d) None

Ans: **A**

18.The protected and public members are inherited by the

- a) Super Class b) Subclass c) Multi Class d) None

Ans: **B**

19.java inheritance is that a class can extend only

- a) one super class b) Two super classes c) Both d) None

Ans: **A**

20.Inheritance can be classified into

- a) 06 b) 05 c) 04 d) 01

Ans: **B**

21.When a subclass is derived from a class which itself has been derived from some class, then this mechanism is known as the

- a) Multilevel Inheritance b) Single Inheritance
c) Both d) None

Ans: **A**

22._____Inheritance more than one sub class is derived from a super class.

- a) Multilevel Inheritance b) Single Inheritance
c) Hierarchical d) None

Ans: **C**

23.When a subclass is derived from more than one super class then it is often referred to as

- a) Multiple Inheritance b) Single Inheritance
c) Hierarchical d) None

Ans: **A**

24. Java does not support

- a) Multiple Inheritance
- b) Single Inheritance
- c) Hierarchical
- d) None

Ans: **A**

25. A combination of multilevel and hierarchical inheritance is known as

- a) Multiple Inheritance
- b) Single Inheritance
- c) Hierarchical
- d) Hybrid

Ans: **D**

26. A _____ is a variable declared with the final keyword

- a) Final Variable
- b) Variable
- c) Fast Variable
- d) None

Ans: **A**

27. A _____ is one that cannot be extended by another class

- a) Variable
- b) Final Class
- c) Fast Variable
- d) None

Ans: **B**

28. A class which cannot be instantiated is called an

- a) abstract class
- b) Final Class
- c) Fast Variable
- d) None

Ans: **A**

29. An applet is a _____ written in Java

- a) abstract class
- b) web-based program
- c) Fast Variable
- d) None

Ans: **B**

30. Types of Applets

- a) 02
- b) 04
- c) 05
- d) 06

Ans: **A**

Unit-V

5.0 Aims and Objectives	248
5.1 Introduction	248
5.2 Packages	248
5.2.1 Introduction to Packages	248
5.2.2 Defining a package	249
5.2.3 Creation of Package	251
5.2.4 User defined package	252
5.2.5 Java API Packages	254
5.2.6 Using System Packages	255
5.2.7 Naming conventions	256
5.2.8 Accessing a Package	257
5.2.9 Using a Package	258
5.3 Interfaces	260
5.3.1 Introduction	260
5.3.2 Defining interfaces	260
5.3.3 Extending interfaces	261
5.3.4 Implementing interfaces	262
5.3.5 Assessing interface variables	264
5.3.6 Multiple inheritance	267
5.4 Managing Errors and Exceptions	269
5.4.1 Types of errors	269
a) Compile-time errors	269
b) Run-time errors	271

5.5 Exceptions	273
5.5.1 Exception handling	278
5.5.2 Multiple Catch Statements	287
5.5.3 Using finally statement	289
5.6 Managing input/output files in java	291
5.6.1 Introduction	291
5.6.2 Reading and writing files	291
5.7 Concept of Streams	292
5.7.1 Stream classes	293
5.7.2 Byte Stream Classes	293
5.7.3 Input Stream Classes	293
5.7.4 Output Stream Classes	294
5.8 Character Stream classes	294
5.8.1 Reader and Writer stream classes	295
5.8.2 Streams in Java	296
5.9 Summary	296
5.10 Exercise	298
5.11 Objective Type Questions	299

5.0 AIMS AND OBJECTIVE

Aims:

The aim of the unit is to cover fundamental concept of java Packages, Interfaces and Exceptional handling in depth. Java is a cross platform, general purpose programming language commonly used in critical applications such as banking systems. Objective of this unit is to provide basic insight into the core features of java programming language. Such as Packages, interfaces and exception handling. This unit also intends to illustrate key concepts through easy to understand examples for enhancing the understanding the reader.

5.1 INTRODUCTION

A Package consists of collecting of classes and interfaces. The users can use the methods of those classes and interfaces as and when needed. It provides reusability.

5.2 PACKAGES

5.2.1 INTRODUCTION TO PACKAGES:

Package refers to a group of classes and related types. It essentially provides access protection and name space management.

Often related types are grouped together in various packages. This has a number of advantages. It makes the types much sample to asset and worn. It also helps to escape the case of clash in names and makes it easier to control and manage access. It assists to regulatethe classes into a folder shape and make it simple to spot and use them. It also helps to improve re-usability.

Building a Package:**Steps to build a Java package:**

- ❖ Select the package name
- ❖ Pick up a base directory
- ❖ Make a subdirectory from the base directory that matches your package name.
- ❖ Place your source files into the package subdirectory.
- ❖ Use the package statement in each source file.
- ❖ Compile your source files from the base directory.
- ❖ Run your program from the base directory.

Example of a package:

```
Package package1;
```

```
Class Nayan
```

```
{
}
```

5.2.2 DEFINING A PACKAGE:

Each and every class is a part of a certain package.

Every class in a file is a part of a single specific package.

A single package may comprise of multiple files.

In case when no particular package is specified, then by default the classes in that particular file go into a special unnamed package. (Same unnamed package for all files).

It is possible to access public classes which are present in some other package by importing them:

```
import package_name.nameof_class;
```

It is also possible to access the public fields and methods of such classes by using:

```
import package_name.nameof_class.methodorfield_name;
```

If you don't want to specify any package, following should be used.:

```
import nameof_package.*;
```

Syntax of Package:

```
Package <package_name>;
```

A few of the extant packages in Java programming language are

java.lang – It comprises of all the elemental classes

java.io – Many classes specific to input and output are contained in this package.

Naming a Package:

Names for packages are always given lower case. This helps in avoiding conflict of similar names.

In Java, packages must always start with java. Or javax.

Example Program of a Package:

```
//Program Name as Package.java
package package1;
public class BSC
{
public static void main(String args[])
{
System.out.println("congrats to BSC MPCs Freshers");
}
}
```

5.2.3 CREATION OF PACKAGE:

Steps to create a Java package:

Select the package name

Pick up a base directory

Make a subdirectory from the base directory that matches your package name.

Place your source files into the package subdirectory.

Use the package statement in each source file.

Compile your source files from the base directory.

Run your program from the base directory.

Example of Package Program:

```
package package1;
class manju
{
}
```

In the above example, we have created a package with the name “package1”, in which a class is created.

Example of Package Program:

```
Package nayanpack1;
Import java.io.*;
Public class faculty
{
Public static void main (String args[])
{
System.out.println(“congrats to new faculty”);
}
}
```

In the above program, we have created a package with the name “nayanpack1”, in which a class is created as faculty.

5.2.4 User Defined Package:

It is a group of objects created in a unique list. Creation of our own package (User defined package) consist the ensuing steps.

- ❖ conform the package at the starting of a file proving the form:

Package packagename;

- ❖ Describe the class this is to continuebring in the package and confirm it as public.

```
package packagename; ───────────────────▶ package declaration
public class classname ─────────────────▶ class definition
{
.....
}
```

- ❖ Build a subdirectory supporting the listpoint the majorsource files are saved.
- ❖ Save the file (classname.java) in the main directory (bin).
- ❖ Edit the file, this builds “.class file”. Remember that the “.class file” must be located in a list that has the name similarly as the package name.

Example Program of Package:

```
package arithmetic;
public class Addition
{
public void add(int p, int q)
{
System.out.println(“Addition=”+(p+q));
}
}
```

- ❖ Save this file as “Addition.java”.
- ❖ Compilation: javac-d. Addition.java

This statement will generate a directory named with “arithmetic” in the current directory (bin) and “.class” file is stored in “arithmetic” directory.

Accessing a package:

Packages can be accessed by using import keyword.

Syntax:

```
import packagename.classname;
```

(or)

```
import packagename.*;
```

Using a package:

```
import arithmetic.Addition;
```

```
class AddUse
```

```
{
public static void main(String args[ ])
{
Addition a= new Addition( );
a.add(10, 20);
}
}
```

Using Package Members:

Importing a Package Member:

In Java it is possible to import a single member of a package instead of importing the whole file. This can be done by using the import statement before any type definitions. The example below shows how the member rectangle can be imported from the graphics package.

```
import graphics.Rectangle;
```

Now it is possible to use the member anywhere in the class like below:

```
Rectangle my_Rect = new Rectangle();
```

Illustrating an Entire Package:

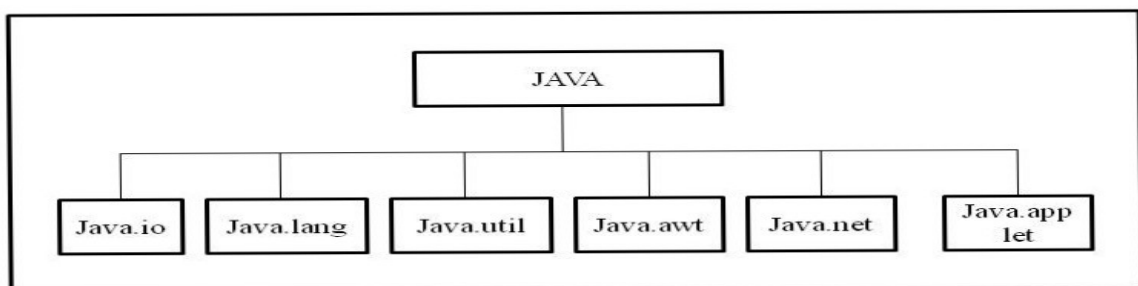
In order to implicate all the contents of a package, we can use the implication statement and the package name followed by an asterisk character(*).

```
import graphics.*;
```

```
import java.awt.*;
```

5.2.5 JAVA API PACKAGES:

Java Application Programming Interface (API) provides a huge count of classes arranged within the particular packages giving to its functionality.



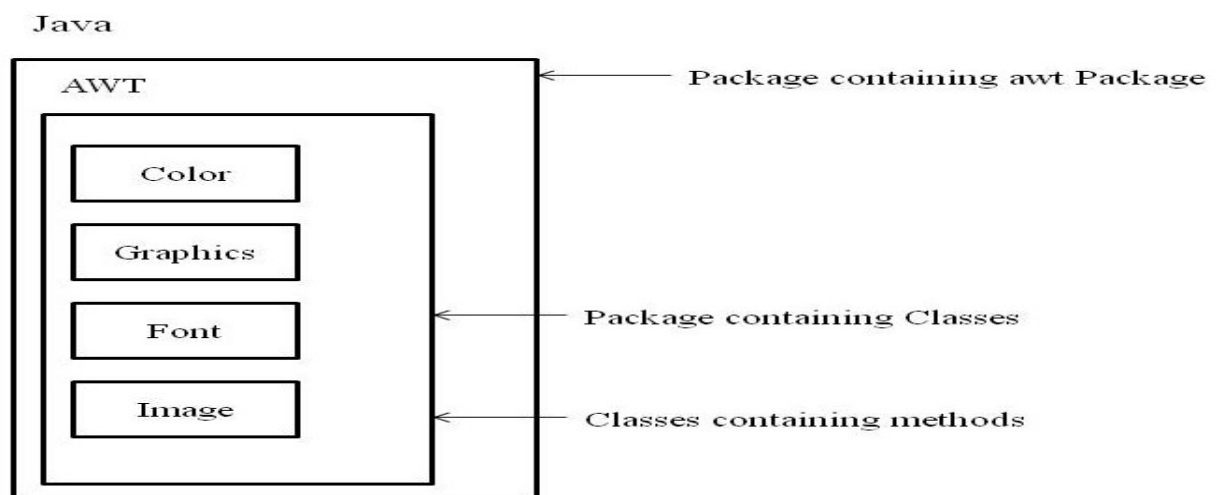
Package name	Contents
java.lang	It contains Language support classes. The Java compiler automatically uses this java.lang package. It includes classes for primitive data types, strings, stringBuffer, stringBuide, math functions, threads and exceptions etc.
java.util	It contains the Language utility classes such as vectors, hash tables, date, StringTokenizer etc.
java.io	It contains the classes support Input and Output classes.

java.awt	It contains Set of classes for implementing Graphical User Interface (GUI).
java.net	It contains Classes for networking.
java.applet	It contains the Classes for creating the applets and implementing applets.

5.2.6 USING SYSTEM PACKAGES:

Using system packages:

The packages in Java are organized in a hierarchical structure. Below figure shows that the package named “java” contains the package “awt” which inturn contains various classes requires for implementing GUI.



In many situations, the user can same to use many of the classes involved in a package. We achieve this easily as follows:

```
import packagename.classname;
```

(or)

```
import packagename.*;
```

The first statement allows the specified class in the specified package to be imported by using import keyword.

Example:

```
import java.awt.Color;
```

The second statement imports every class contained in the specified package.

Example:

```
import java.awt.*;
```

5.2.7 NAMING CONVENTIONS:**Naming conventions (or) Naming rules:**

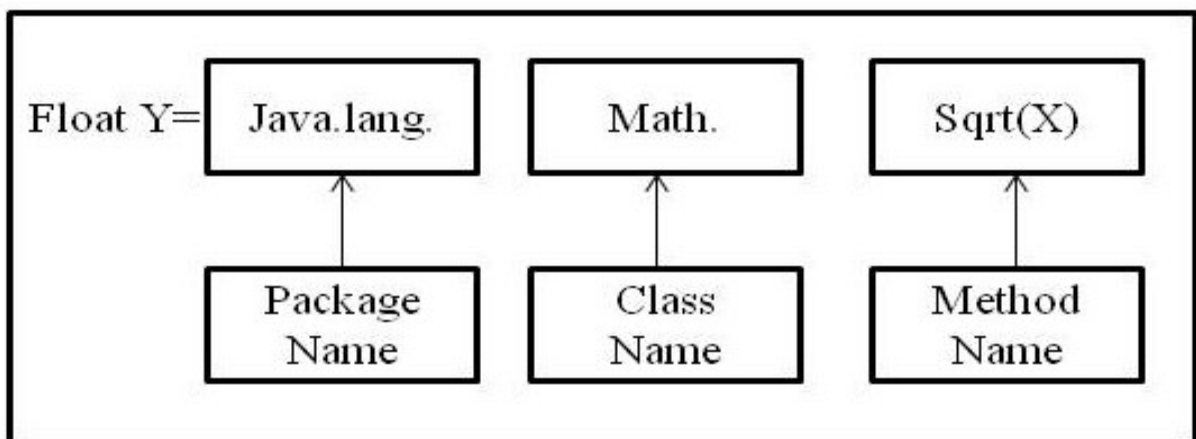
Package could be named applying Java common naming rules.

In this Packages starts with lowercase alphabets

And all class names begin with uppercase letter.

All methods begin with lowercase letters.

Example:



5.2.8 ACCESSING A PACKAGE:

In this section the following program given the example

Example:

```
Package varshini;
Class nayan
{
Public void sadguna()
{
System.out.println("sadguna of nayan");
}
Public static void main (String args[])
{
Nayan obj=new nayan1();
Obj.sadguna();
}
}
```

Program explanation:

In this above program first create class name varshini with method name sadguna and create main method than create an object as nayan1 and main method as sadguna now let's say I want put this class in to the package name as varshini and the file name save as Manju.java, lets compile the file javac compilation done file name as nayan.java class file is created but we have to create package with this use command javac -d . Manju.java this command for force is compiler to create a package here . (dot) operator means the working directory if you check up the folder package name Manju created which as class nayan if you want to compile the file with double .. (Double dots) the package will be created in the main directory which is the main drive like as C:\>. Let's say the user want to create a

sub package modify the statement as Manju dot Manju khari compile the file in the manju folder another folder Manju khari is created with class nayan to execute the code mention the perfectly examined name of the class that is package name and sub package name followed by the class name.

5.2.9 USING A PACKAGE:

In Java programming language, single program conveys individual single public class. The User cannot put two or more public classes together in “.java” file. If we want to create a package with multiple public classes in it, we may follow the following steps.

- ❖ Decide the name of the package.
- ❖ Create a sub-directory with this package name under the main directory.
- ❖ Create classes that are to be placed in the package.
- ❖ Compile each source file.

Example:

- ❖ Build a list by name “arithmetic” and change to that list.
- ❖ Build the coming file (subtraction.java) and edit the file in the same list.

```
package arithmetic;
public class subtraction
{
public void subtraction(int p, int q)
{
System.out.println (“subtraction=”-(p-q));
}
}
```

Create the second file (multiplication.java) and edit the information in the similarly directory.

```

package arithmetic;
public class multiplication
{
public void multiplication(int p, int q)
{
System.out.println (“multiplication=”*(x*y));
}
}

```

Using of package:

At this time the userchange to the parent (bin) list, create the coming program (nayan.java) and complete it.

```

import arithmetic.*;
class Annapoorna
{
public static void main(String args[ ])
{
Addition a1= new Addition();
a1.add(10, 20);
Subtraction s1= new Subtraction( );
s1.sub(20, 10);
}
}

```

Naming a Package:

- ❖ In the naming packages the name of the package were drafted in all lower case letters to escape clash with the names of the classes or interfaces.
- ❖ Packages in the java programming language starts with java. Or javax

Using Package Members:

Comprise a package is also known as Package members.

5.3 INTERFACE:**5.3.1 INTRODUCTION:**

In Java, interfaces are abstract in nature and are used to achieve polymorphism.

The keyword interface is used for the declaration of interface. An interface in Java can only have method signature and constant declarations. An interface may also have variables, but only static and final variables are allowed.

Method definitions are not allowed in an interface.

Using interfaces is like creating contracts. The contract specifies what is compulsory for the class to do, but not how it will do that.

5.3.2 DEFINING INTERFACES:

It is essentially a gentle of class. Here classes and interfaces consist of variables as well as methods with a extensive change. The change is that an interface illustrates only final fields and abstract methods. Then it is the authority of the class that performance an interface to illustrates the code for performance of these methods.

- ❖ In an interface all methods are implicitly public.
- ❖ In an interface all variables are static and final.
- ❖ A class can implement many interfaces.
- ❖ Interface can be extended.
- ❖ It is not possible to create an object to an interface directly, but we can create reference variables.

Interfaces are defined following syntax:

```
interface InterfaceName
{
Variable declarations;
Method declarations;
}
```

Presenthere interface is the keyword and name of the Interfaceis any identified variable (like as class name)

Example of the Interface:

```
interface employer
{
final int employee id=101;
int move();
}
```

5.3.3 EXTENDING INTERFACES:**Extending Interfaces:**

In Java Extending Interfaces once the interface can obtain one more by help of another keyword develop.The syntax is same as for inheriting classes.

Although a class develops a combination that inherits to one more combination.

Syntax of the Extending Interface:

```
Interface manju2 extends manju1
{
Body of manju2;
```

Example of Extending Interface

```

Interface employee
{
Int Employee id=001;
String name="nayan";
}
Interface software extends employee
{
Void display();
}

```

5.3.4 IMPLEMENTING INTERFACES:**Implementation of Interfaces:**

The implementation of interfaces are used as “super classes”, these interfaces are chosen to be equality and inherited by the classes. The syntax of the implementation interfaces like as:

```

class classname implements interfacename
{
Body of class;
}

```

Example program for implementing an Interface:

Name of the Program: person.java

```

Public class person implements employee
{

```



```
public void employee id()
{
System.out.println("employee id");
}
public void name of the employee()
{
System.out.println("name of the employee");
}
public int salary()
{
return salary;
}
public static void main(String args[])
{
Employee emp=new emp1();
emp.employee id();
emp.employee name();
emp.employee salary();
}
}
```

Output of the program:

Employee id

Employee name

Employee salary

5.3.5 ACCESSING INTERFACE WITH VARIABLES:

The program is an example for accessing interface variables. It also illustrates how interfaces can be used to simulate multiple inheritances. The program contains an interface named “Allowance” and classes named “Employee” and “Salary”.

Example Program:

Interface Allowance

```
{
class
int da=2000;
int hra=3000;
}
class Employee
{
private int empno;
private String name;
public void getData(int eno, String n)
{
empno= eno;
name= n;
}
public void show( )
{
System.out.println(“Employee number:” +empno);
System.out.println(“Employee name:” +name);
}
}
class Salary extends Employee implements Allowance
```

```
{
private double sal;
public void getData(int eno, String n, double s)
{
super.getData(eno, n);
sal= s;
}
public void total()
{
super.show( );
System.out.println("Total salary:" +(sal+da+hra));
}
}
class Test
{
public static void main(String args[ ])
{
Salary e1= new Salary( );
e1.getData(1001,"sadguna",50000);
e1.total( );
}
}
```

Output of the Program:

Employee number: 1001

Employee name: sadguna

Total salary: 55000

Example Program:

```

interface A
{
/* internally public static final int x=10;*/
int x=10;
/* convert internally to public abstract void show() */
void show();
}
interface B extends A
{
/* public abstract void disp(); */
void disp();
}

```

Example Program:**Interface for Multiple Inheritance**

```

interface Animal
{
void sleep();
void eat();
}
interface Pet
{
void faithful();
}
class Dog implements Animal, Pet
{
public void sleep() {}
}

```

```

public void eat(){}
public void faithful(){}
}

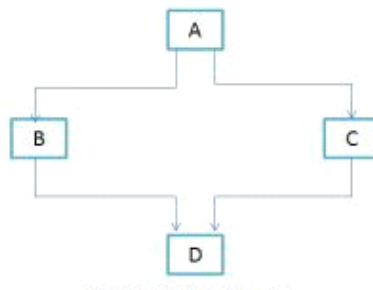
```

5.3.6 MULTIPLE INHERITANCE:

The java programming language does not allow multiple inheritance. This has been done to avoid cases of ambiguity. One such popular case is of the diamond problem.

Diamond problem:

The below diagram depicts the diamond problem. In the diagram, the class A extends both the class B & C. The problem arises when the class B & C both override some method of class A in its own ways. This is where ambiguity arises. Class D can inherit any of the methods, either from class B or Class C. Therefore, to remove such ambiguity Java does not allow any multiple inheritance.



Achieving Multiple Inheritance In Java Using Interfaces

Interface P

```

{
public void Method();
}

```

Interface Q

```

{

```

```

public void Method();
}
class Demo implements P, Q
{
public void Method()
{
System.out.println (" Multiple inheritance example using interfaces");
}
}

```

In the above program a single class implements two interfaces. Even though multiple interfaces are used, no ambiguity arises. This is due to the fact that java interfaces are always abstract and hence they contain no implantation of their own. Thus, even though the two interfaces had methods of same names, no ambiguity arises.

Interface Features:

- ❖ It denotes the object's functionality without giving out the actual implementation.
- ❖ With help of interfaces, encapsulation can be implemented.
- ❖ The implementation can be varied without affecting the caller of the interface.
- ❖ There is no requirement for the implementation at the compile time.
- ❖ Object instance is associated with the interface throughout the execution time of the program.

Example of Interface:

```

interface A
{
/* internally public static final int x=10;*/

```

```

int x=10;
/* convert internally to public abstract void show() */
void show();
}
interface B extends A
{
/* public abstract void disp(); */
void disp();
}

```

5.4 MANAGING ERRORS AND EXCEPTIONS:

In java programming language when we get the errors in compilation then the program is wrong. In system it is called as “bugs” (errors). While writing the program the errors are common like as logical errors or syntax errors or runtime errors . when we get the errors the error causes which type of error that is.

Any faulty can come an false output or can eliminate the program output of the program unexpectedly (in center) or directly can source the system to crash. Therefore it is essential to identify and maintain perfectly all the available errors.

5.4.1 TYPES OF ERRORS:

Compile-Time Errors:

In java programming language while executing the the programs the syntax errors will be identified and visible by the java compiler and therefore these errors are known as compile time errors. Whenever the compile visible an error, it will not create the “.class” file

Example of the Compile – Time Error:

```

class Error1
{
public static void main(String args[ ])
{
System.out.println(“ Hello java”) //missing
}
}

```

The Java compiler does a nice job of telling us where the errors are in the program. For example, in the above program, we have missed semicolon, the following message will be displayed.

Error1.java:6: ‘;’ expected

System.out.println (“Hello java”)

^

1 error

The user goes with relevant line, modify the error and run the program again. In compile – time errors most of the things in typing mistakes. Most of the users do the common errors are:

- ❖ Not entered semicolons
- ❖ Not entered Missing brackets in classes and methods
- ❖ Misspelling of identifiers and keywords
- ❖ Double quotes not return in strings
- ❖ Use of undeclared variables
- ❖ Use of = in place of == operator and so on.

Run-time errors:**Runtime errors:**

In Run time errors after compilation of the program it gives the no errors but the program not gives the result. In run time errors programs may gives error results expected to mistaken logic or may conclude expected to errors. In runtime errors some of the mostly errors are:

- ❖ Split an integer by zero.
- ❖ Approaching a factor that is away of constrained of an array.
- ❖ Demanding to cache a assessment into an array of an conflicting type.
- ❖ Attacking to help a unfavourable extent for an array.
- ❖ Approaching an aspect that is away of obligates of a string.
- ❖ Modifying false string to a figure and so on.

Example of Run time Errors:

```
class nayan
{
public static void main(String args[ ])
{
int p=5;
int q=4;
int r=4;
int s=p/(q-r); //analysis by zero
System.out.println("s="+s);
int t= p/(q+r);
System.out.println("t="+t);
}
}
```

In the program fantastically right and accordingly achieve no more element any dispute over as sortment. Although, during the execution, it acts the ensuring information and pauses beyond finishing another statements.

Exception:

In this exception is an atypicalact that appearover the result of the program and confuse the routineprocess of the program. Irregularity doesariseduring your program is executing. For example, the userscapabilities await the programmer to access an integer, althoughaccept a content string; or an unanticipated Input/output error pops up at runtime. Java has animplicitssystem for approachingin time errors, assigned to as exception handling. Exception is to assure that you can createpowerful programs for mission-critical applications.

Exception Handling

In java programming language, while each cor dial of a typical surroundings arise with in a approach then the exclusions are launched in mode of exclusion thing that is the routine program authority outflow is interrupted and an exclusions things is build to hold that unique status.

In Exception Handling the approach builds athing and palms it done to the in timerule. Essentially, complete the data around the error or any amazing situation is gathered in this group of things in the scheme of a stack. This object build is also known an exception object the action is described as delivering exclusion.

The structure of approaching an exclusion is also known as catching an exclusion or approaching an Exception or commonly Exception handling.

Advantages of Exception-handling in Java:

In java programming language exception brings the measures to isolate the particulars of that while act while being away of the normal appears from the special sense of a program.

One of the implications of this system is that it delivers exclusion on any occasion a occupation approach concurrence an fault given that the calling approach returns responsibility of that error.

Among the comfort of the present system the functioning code and the error-handling code could be descend. It again allows us the capacity of establish and separate between various error types proving an isolated section of codes. This is affected along the assist of try-catch blocks.

As well the errors could be increased up the approach call stack that is issues existing at the reduced stage in the group could be organized by the approached increased up the call group.

5.5 EXCEPTIONS:

When the normal execution of a program is disturbed due to some abnormal event, an exception is said to be thrown. This can happen due to a number of reasons while the program is running. For example, if an integer is divided by zero the arithmetic exception is thrown. To handle such cases in an efficient manner, Java has a mechanism called exception handling. This makes sure that programmers are able to write robust programs which can be used in critical applications.

Types of Exceptions:

In the Exceptions there are 3 different types. They are following like this:

- ❖ Checked Exceptions
- ❖ Unchecked Exceptions
- ❖ Error

Checked Exceptions:

The Checked Exceptions which are encountered by the compiler are referred to as checked exceptions. The compiler looks for the proper exception handling in the program. The exceptions must have been considered by the programmer and handled appropriately. Doing so will not result in compile time errors?

All the exceptions extend the java.lang. Exception class. The cases of exception should be predicted by the programmer. Any possible checked exceptions which might occur are needed to be caught.

Such as, where ever the readLine () method is called at a Buffered Reader object again the IO Exception might exist. IO Exception any occur in a program which reads data using the method readLine ().

Some examples of checked exceptions are given below:

Instantiation Exception

No Such Field Exception

No Such Method Exception

Clone Not Supported Exception

Interrupted Exception

Illegal Access Exception

Class Not Found Exception

Example program for checked exception:

```
import java.io.*;
class annapurna
{
public static void main(String args[ ])throws IOException
{
```

```

DataInputStream dis=new DataInputStream(System.in);
int p;
System.out.println("Enter a number");
p= Integer.parseInt(dis.readLine());
if(p>0)
{
System.out.println(p+"is positive number");
}
System.out.println("Program end");
}
}

```

In the above example, if we does not insert throws IOException, it will generate IOException exception, which is a checked exception.

Unchecked Exceptions:

Whenever an exception occurs at the run time of a program, it is called as unchecked exception. Such exception is constitutional via the operation. In Unchecked Exceptions they enlarge the java.lang.RuntimeException which is inherited from java.lang. Exception class.

Such exceptions cannot be known beforehand by the compiler. They cannot be handled in a similar manner as logical errors or syntax errors.

Most often these are caused by situations such as arithmetic overflow, division by zero.

Following given below the unchecked exceptions:

- ❖ Null Pointer Exception
- ❖ Index Out Of Bounds Exception
- ❖ Security Exception
- ❖ Array Index Out Of Bounds Exception
- ❖ Arithmetic Exception
- ❖ Illegal State Exception
- ❖ Class Cast Exception

Example program for unchecked exception:

The following program when executed displays, “invalid index”, because we are referring to “p[6]” which is invalid.

```
class Test
{
public static void main(String args[ ])
{
int p[ ]= {25, 35, 45, 55};
try
{
System.out.println(a[6]);
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println(“invalid index”);
}
}
}
```

Error:

In Java programming language, errors are always over to the function. Errors often are the problematic settings which cannot be appropriately handled through the function. Errors in Java are a part of the Error class and its sub classes. Try-catch blocks should not be used for errors. This is due to the fact that errors usually signify scenarios which should not be ignored and caught by the application.

Some examples for error are exceptions such shortage of memory or stack overflow. Errors usually correspond to serious problems.

Here is the list of unchecked exceptions

Illegal Access Errors

Internal Error

Stack Over Flow Error

No Such Field Error

Example program for Error:

```
class varshini
{
public static void main(String args[ ])
{
int p=20;
int q=08;
r=p+q;
System.out.println("r="+r);
}
}
```

In the above example, we are not declared the variable “r”, in this case it will generate No Such Field Error.

5.5.1 Exception handling:

Exceptions are thrown as an object of the Exception class. This happens when something out of the normal happens in a program. When this happens, the execution is stopped and the exception object handles the error.

The run time system receives the exception object. This exception object holds all the data related to the abnormal code which caused the error in the first place. It is stored in form of a stack. This scenario is often regarded as delivering of an exception. The handling of the exception is also referred to as catching an exception.

Advantages of Exception-handling in Java:

The main logic of the program is separated from the logic for when something abnormal happens. This is useful as it helps to form the code more explicit and easy to preserve.

The calling method has to take care of the exception. This is also of great significance as it makes the program more efficient.

All the error handling code can be spread away the actual code for the application. Organizing in different error blocks also becomes very easy using multiple try-catch blocks. Therefore in java, error handling becomes very easy in comparison to other languages.

The errors which occur at the lower level are passed to the upper level. This is done by fleeing the exception thing to the calling method. Therefore, the exception travels up the stack.

Steps in Handling Exception (try..Catch):

The following are the tasks in handling exceptions

Hit the exception – finding the main problem

Throw the exception – informing of occurrence of an error

Catch the exception - receive error

Handle the exception – measures for resolving

Finding the problem refers to identify the part of the program where the error may occur. That part of the program has to be enclosed in try block and the catch block contains the codes that represent the action to be taken when the error occurs.

Try_catch block:

In Java programming language exception handling is completed by accepting try-catch blocks. The programmers can usage the try.. Catch block to handle the exceptions that suit their programs. This avoids abnormal termination of the program.

Syntax:

```

.....
.....
try
{
//Statements that may generate the exception
}
catch(exception class object)
{
//Statements to process the exception
}
catch(exception class object)
{
//Statements to process the exception
}

```

```

....
finally
{
//Statements to be executed before exiting exception handler
}
.....

```

Try Block:

Internal the try block we can include the statements that can cause an exception and throw an exception.

Catch Block:

The catch block consists of the code that handles the exceptions and may correct the exceptions that ensure normal execution of the program. Catching the thrown exception object from the try block by the answering catch block is called throwing an exception. The catch block should immediately follow the try block We can have multiple catch block for a single try block.

Finally Block:

Irrespective of the flow of the program, the code internal the finally block is always completed. It is not dependent on whether an exception occurred internal the try catch block or not. It is also executed in the case when the exception is thrown and not handled in the catch block.

It is recommended that some type of clean up is included in the finally block as the finally block is completed every time.

It is optional to include the finally block in any program.

Example program of Exception Handling:

```

/* Exception handling */
import java.util.Scanner;
class manju
{
public static void main(String[] args)
{
int p, q, r;

Scanner input = new Scanner(System.in);
System.out.println("Input two integers");
p = input.nextInt();
q = input.nextInt();
r = p / q;
System.out.println("r = " + r);
}
}

```

Output1:

(No Exception)

C:\java>javac manju.java

C:\java>java manju

Input two integers

8 4

Result = 2

Output2:

(With Exception)

C:\java>manju

Input two integers

5 0

Division by zero error occurred

java.lang.ArithmeticException: / by zero

C:\java>

Example program of Exception Handling:

```
class sadguna
{
public static void main(String args[ ])
{
int p[ ] = {20,30,};
int q=7;
try
{
int r = p[2] / (q-p[1]);
}
Catch(ArithmeticException e)
{
System.out.println("Division by zero");
}
Catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Array index error");
}
Catch(ArrayStoreException e)
```

```

{
System.out.println(“wrong data type”);
}
int s = p[1] / p[0];
System.out.println(“s=”+s);
}
}

```

Output:

Array index error

s=2

Throws Keyword:

The programmer can choose to pass on the handling of the exception which might occur inside a method to the caller method. This done by using the keyword “throws”. Whenever throws is specified, there is no requirement for the try-catch block.

```

type my_method (p_list) throws index of exceptions
{
//.....
}

```

The throws section specifies the various exceptions which the method may possibly throw. It is compulsory as long as complete exceptions, exempting for the type Error either Runtime Exception, either several of the subclasses.

Program Example of Throws clause example:

```
/* throws clause example */
import Java.io.*;
class annapoorna
{
public static void main(String args[]) throws IOException
{
int a;
String name;
DataInputStream din=new DataInputStream(System.in);
System.out.print("Enter the name:");
name=din.readLine();
System.out.print("Enter the register number:");
a=Integer.parseInt(din.readLine());
System.out.println("\nNAME:"+name);
System.out.println("Reg.No:"+a);
}
}
```

Output:

```
C:\java>javac annapoorna.java
```

```
C:\java>java annapoorna
```

```
Enter the name: sravani
```

```
Enter the register number: 01
```

```
NAME: sravani
```

```
Reg.No: 01
```

```
C:\java>
```

User Defined Exceptions:

In scenarios, where the built-in exceptions of Java are not suitable, the programmer has the option for creating a user defined exception. This is useful to properly handle the cases highly specific and unique to different applications.

It is essential to store in attention the below mentioned points while certain a user defined exception:

- ❖ The programmer described exception class condition develops from Exception class.
- ❖ The to String() method must always be overridden in the user-defined exception class in order to give some useful information regarding the exception.

Syntax:

```
class name_exception extends Exception
{
name_exception(parameterlist)
{
//required code
}
public String toString()
{
return String
}
}
```

The user defined exceptions has to be explicitly thrown. This can be done by using the throw keyword as given below:

- ❖ throw ThrowableInstance
- ❖ ThrowableInstance should be an object of Throwable class or a subclass of Throwable.

Example:

For example the below program creates an user defined exception when the second argument is less than or equal to zero

```
/* User defined exception class */
class myexception extends Exception
{
myexception(String mg)
{
super(mg);
}
}
class userexcep
{
public static void main(String args[])
{
int k,l;
k=Integer.parseInt(args[0]);
l=Integer.parseInt(args[1]);
try
{
if(l<=0)
throw new myexception("Invalid Number");
float m=k/l;
System.out.println(k+" / "+l+"="+m);
}
catch(myexception e)
{
```



```

System.out.println("The second number should be greater than 0");
System.out.println(e.getMessage());
}
}
}

```

Output 1:

(No exception)

```
C:\java>javac userexcep.java
```

```
C:\java>java user exception 6 3
```

6 / 3=2.0

Output 2:

(With Exception)

```
C:\java>java user exception 6 0
```

the second number should be greater than 0

Invalid Number

```
C:\java>
```

5.5.2 Multiple Catch Statements:

Sometimes multiple exceptions may arise during the run time of a program. In such cases, multiple try-catch blocks maybe used.

Multiple catch blocks should used in such casesTThe system will consider the multiple catch blocks in the order in which they are specified. It will stop the search as soon as the exception matches and then that particular block will be used.

Example of Exception handling multiple catch block:

```
/* Exception handling Multiple catch block*/
class annapoorna
{
public static void main(String args[])
{
try
{
int x,y,z;
x=Integer.parseInt(args[0]);
y=Integer.parseInt(args[1]);
z=c/d;
System.out.println(x+" / "+y+"="+z);
}
catch(ArithmeticException e)
{
System.out.println("Division by zero error occurred");
System.out.println(e);
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Supply two arguments from the command line");
System.out.println(e);
}
catch(NumberFormatException e)
{
System.out.println("Not valid Integers");
```

```
System.out.println(e);
}
}
}
```

Output 1:

```
C:\java>javac annapoorna.java
```

```
C:\java>java annapoorna 6 2
```

```
6 / 2=3
```

Output 2:

```
C:\java>java annapoorna 6
```

Supply two arguments from the command line

```
java.lang.ArrayIndexOutOfBoundsException
```

Output 3:

```
C:\java>java annapoorna 2.5 3
```

```
Not valid Integers
```

```
java.lang.NumberFormatException: 2.5
```

```
C:\java>
```

5.5.3 Using finally statement

The code in the finally block will be completed even either no more the exception arise inside the block of code.

The finally block is normally used for cleanup activities like file closing, flushing buffers etc.

The finally block is elective.

Example of finally block:

```
/* Finally block*/
class nayan
{
public static void main(String args[])
{
int a,b,c;
a=Integer.parseInt(args[0]);
b=Integer.parseInt(args[1]);
try
{
c=a/b;
System.out.println(a+" / "+b+"="+c);
}
catch(ArithmeticException e)
{
System.out.println("Division by zero error occurred");
System.out.println(e);
}
finally
{
System.out.println("Inside finally block");
}
}
}
```

Output 1:

```
C:\java>javac nayan.java
```

```
C:\java>java nayan 6 3
```

```
6 / 3=2
```

```
Inside finally block
```

Output 2:

```
C:\java>java nayan 6 0
```

```
Division by zero error occurred
```

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

```
Inside finally block
```

The finally block is constantly completed, disregarding of even if or not either an exception is launched. In the above example, the user have inserted a finally block and it is executed every time regardless of error.

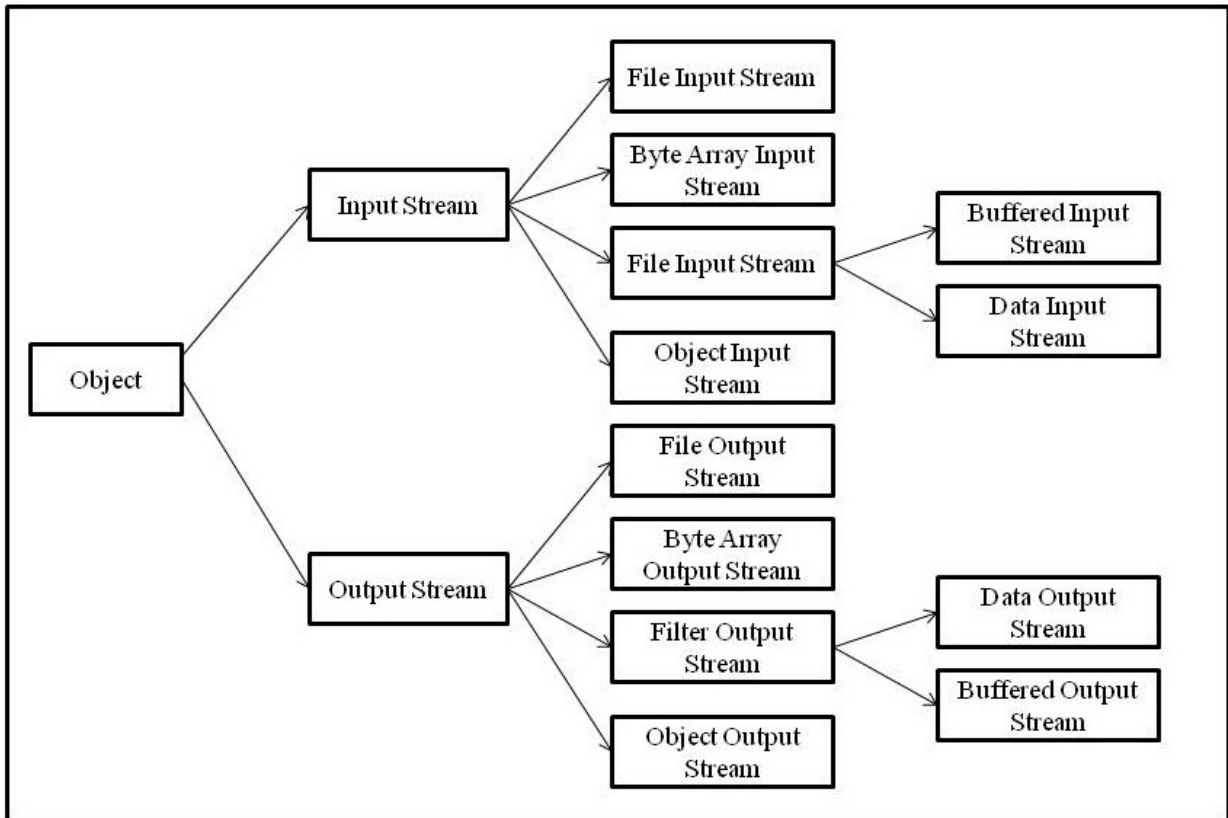
5.6 MANAGING INPUT/OUTPUT FILES IN JAVA**5.6.1 INTRODUCTION**

In order to attach streams to data sources like files and network connections, file streams are used. The subclasses of `java.io.InputStream` and `java.io.OutputStream` `java.io`. Which are `FileInputStream` and `java.io.FileOutputStream` classes are used to read and write files.

5.6.2 READING AND WRITING FILES:

In java programming language stream classes can be specified as a continuance of data. Here the `InputStream` is worn to read the information from the authority and the `OutputStream` is worn to write the information to a target.

The following figure shows reading and writing streams.



The main streams are two, these are one is `FileInputStream` and the second one `FileOutputStream`.

5.7 CONCEPT OF STREAMS:

Sequence of data is also called as a Stream. The streams are divided in to Two types. These are:

1. Input Stream
2. Output Stream

The Input Stream is used to read the data from a source as well as output stream is used to write the data from destinations.

5.7.1 STREAM CLASSES:

The Stream Classes are different types. These are

1. Byte Stream Classes
2. Character Stream Classes

5.7.2 BYTE STREAM CLASSES:

In Java programming language byte streams are used to execute input and output of 8-bit bytes. For all that nearby no end classes similar to byte streams although the ultimate regularly used classes are, `FileInputStream` and `FileOutputStream`.

5.7.3 INPUT STREAM CLASSES:

When it is required to read data from some file, `FileInputStream` is used. A `FileInputStream` fetches input bytes from a file in the host file system. To create a new object, the 'new' keyword is used. Many different constructors are available.

```
InputStream is = new FileInputStream("C:/java/school");
```

In the above example, the name of the file from which the data needs to be read is given as the argument.

```
File f = new File("C:/java/hello");
```

```
InputStream f = new FileInputStream(f);
```

The above used constructor takes a file object to create an input stream object, therefore a file object must be created first.

5.7.4 OUTPUT STREAM CLASSES:

To create a file and write data into it, `FileOutputStream` is utilized. If the file doesn't already exist, this stream first creates it, before opening it for output.

Two constructors which can be used to create a `FileOutputStream` object are described below:

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

The above used constructor takes the file name as the argument to create a stream object to write the file.

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

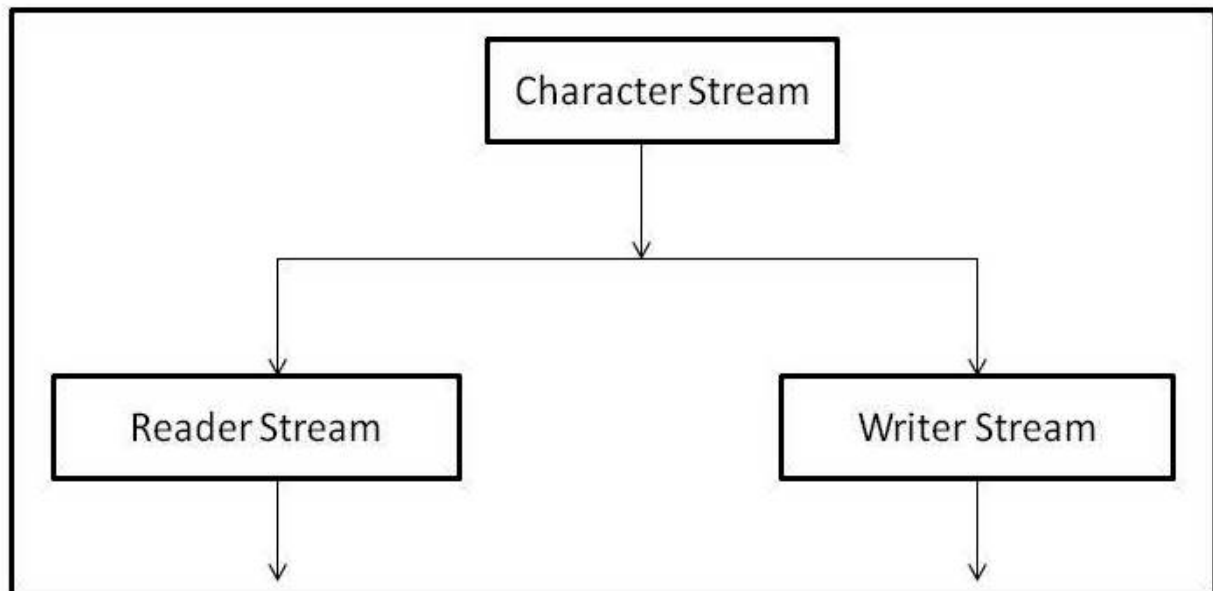
In the above method, it is first required to create a file object using the `File()` method. Then the constructor takes that object as an argument to create a stream object to write the file.

After the `OutputStream` object is created helper methods can be used to write to stream or to perform other operations on the stream. It is also possible to specify encoding in the constructor of the `OutputStreamWriter` on a `FileOutputStream`.

5.8 CHARACTER STREAM CLASSES:

In Java programming language Byte streams continue work via execute input and output of 8-bit bytes, considering that Java programming language Character streams continue work via execute input and output for 16-bit unicode. For all that exist continue full classes associated to character streams although the maximum usually work classes continue, `FileReader` and `FileWriter`. For all that privately `FileReader` needs `FileInputStream` and `FileWriter` needs

FileOutputStream although present the leading change is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.



5.8.1 READER and WRITER STREAM CLASSES:

This **Reader** and **Writer** classes and its subclasses act upon since doing along information in **document** form. During the **Writer** class consist of approaches such act equivalent to these feasible in **OutputStream** class, the **Reader** class consist of approaches this act equivalent to these feasible in **InputStream** class. Here sole variation is that the approaches in the **Writer** and **Reader** classes act arrange to contract including character input or output, at the same time the identical approaches in the **OutputStream** as well as **InputStream** classes act arranged to contract including byte input or output.

5.8.2 STREAMS:

It is a arrangement of objects against a authority, which ever stays combined movements. These are the following characteristics of a Streams:

- ❖ Sequence of elements
- ❖ Source
- ❖ Aggregate operations
- ❖ Pipelining
- ❖ Automatic iterations

It is a modern abstract layer popularized in Java programming Language (8). Proving this, the user couldbe measure information in a analytical approach identical to SQL statements.

The following example of SQL Statement:

```
SELECT max(Gross Salary),Lecturer_Identification Number,Lecturer _name
FROM Lecturer
```

This raised SQL explanation accordingly recovery the max salary Lecturer information, on the outside performing several computing on the programmer steam. Proving number of structure in Java programming language, a programmer accepts to cause loops and prepareimitatedreviews. One morething is ability; as dual-core processors actfeasible at calm, a Java programmerincludes to write parallel code converting this could be nice error-level.

5.9 SUMMARY:

This unit has provided a detailed introduction to the Packages. As Packages defining the packages, creation of packages, user defined packages, Java API Packages is very important to understand and this has been presented in the unit in

the various sections. The concept of using system packages and accessing packages is used extensively in Java and Defining interfaces, extending interfaces, implementing interfaces, accessing interface variables and multiple inheritances. Use of errors and explained types of errors. Explained about exceptions, exception handling multiple catch statements and using finally statements.

Java is a powerful programming language in this unit we explained about managing input and output files in java and concept of streams.

5.10 EXERCISE

1. Write a java program to get a list of all file/directory names from the given
2. Write a java program to check if given pathname is a directory or a file
3. Write a java program to check if a file or directory specified by pathname exists or not.

5.11 OBJECTIVE TYPE QUESTIONS

1. A _____ consists of collecting of classes and interfaces.

- a) Package b) String c) Array d) None

Ans: A

2. _____ refers to a group of classes and related types.

- a) String b) Package c) Array d) None

Ans: B

3. A single package may comprise of _____.

- a) String b) Package c) Multiple Files d) None

Ans: C

4. Names for packages are always given _____

- a) Upper case b) Lower case c) None

Ans: B

5. Package is a group of classes created in a unique directory.

- a) Non directory b) Duplicate c) unique directory d) None

Ans: C

6. Packages can be accessed by using import keyword.

- a) Import keyword b) Export Keyword
c) main Keyword d) None

Ans: A

7. API Full form

- a) Application Programming Interface b) Apple Programming Interface
c) Application Programming Internal d) None

Ans: A

8. The packages in Java are organized in a _____

- a) hierarchical structure b) Single Structure
c) Method Structure d) None

Ans: A

9. Comprise a package is also known as _____

- a) Package members b) Package Method
c) Members d) None

Ans: A

10. An _____ in Java can only have method signature and constant declarations.

- a) Internal b) Interface c) Method d) None

Ans: B

11. _____ Definitions are not allowed in an interface.

- a) Internal b) Interface c) Method d) None

Ans: C

12. A _____ can implement many interfaces.

- a) class b) Interface c) Method d) None

Ans: A

13.The implementation of interfaces are used as

- a) super classes b) sub class c) class d) None

Ans: A

14.The java programming language does not allow_____

- a) Inheritance b) Single Inheritance
c) Multiple Inheritance d) None

Ans: C

15. How many types of Exceptions

- a) 04 b) 03 c) 01 d) None

Ans: B

16.The _____ which are encountered by the compiler are referred to as checked exceptions.

- a) Checked Exceptions b) Exceptions
c) Unchecked Exceptions d) None

Ans: A

17.Whenever an exception occurs at the run time of a program, it is called as _____.

- a) Checked Exceptions b) Exceptions
c) Unchecked Exceptions d) None

Ans: C

18. _____ in Java are a part of the Error class and its sub classes.

- a) Errors
- b) Methods
- c) Exceptions
- d) None

Ans: A

19. Exceptions are thrown as an object of the Exception class.

- a) Errors
- b) Methods
- c) Exceptions
- d) None

Ans: C

20. In Java programming language exception handling is completed by accepting _____ blocks.

- a) Try-catch
- b) Methods
- c) Exceptions
- d) None

Ans: A

21. The catch block consists of the code that handles the exceptions and may correct the exceptions that ensure normal execution of the program.

- a) Try-catch
- b) Catch Block
- c) Exceptions
- d) None

Ans: B

22. The catch block should immediately follow the try block.

- a) Try Block
- b) Catch Block
- c) Exceptions
- d) None

Ans: A

23. _____ also executed in the case when the exception is thrown and not handled in the catch block

- a) Try Block
- b) Catch Block
- c) Finally Block
- d) None

Ans: C

24. The finally block is

- a) Elective
- b) Optional
- C) Compulsory
- d) None

Ans: A

Dr. V. B. Narasimha working as a Assistant Professor in computer science department at University College of engineering Osmania University Hyderabad Telangana. He completed his MCA from Osmania University, Hyderabad, M.Tech Computer Science completed from Osmania University. His research work completed from Osmania University. His research areas data mining, networking, image processing, machine learning etc. In his research life he present completed 40 international journals in his research area 25 National and International conferences published, for his research interest he around attended 20 workshops. Worked as a Lecturer in Department of Computer Science, Osmania University, Post Graduate College, Gadwal and Nalgonda from 01.08.1994 to 22.08.2000 and 01.08.2002 to 14.03.2004 respectively. Presently he his Guiding 8 Ph.D. scholars in Information Technology (IT), Department of Informatics, Osmania University. Guiding 2 Ph.D. scholars in Department of Computer Science and Engineering, JNTU, Hyderabad. Guiding 2 Ph.D. scholars in Department of Computer Science and Engineering, K. L. University, Vijayawada, Andhra Pradesh. Guided more than 150 students in their academic projects in Computer Science and Engineering, University College of Engineering, Osmania University. Course Writer for PGRRCCDE, Osmania University and Institute of Public Enterprises. Worked as System Analyst in Software Services and Resources Inc. Atlanta, Georgia (USA) from 28.08.2000 to 18.04.2002.



Dr. Yogesh Kumar Sharma, Presently working as Associate Professor Dept of CSE and Research Coordinator at Shri. "Jagdishprasad Jhabarmal Tibrewala University", Jhunjhunu (Rajasthan). He completed his B.Sc in the year of 2002 awarded from S.M.L. (P.G.) College, Jhunjhunu, Rajasthan University, M.C.A. from Modi Institute of Management and Technology, Kota, Kota University, Kota in the year 2005, Ph.D. in Faculty of Computer Science, from Shri. Jagdishprasad Jhabarmal Tibrewala University", Jhunjhunu (Rajasthan) in the year 2014. His research areas data mining, networking, image processing, machine learning, Artificial Intelligence, Software Engineering, Information Security etc. In his research life she present completed 60 international journals in her research area, 25 National and International conferences, for his research interest she around attended 20 workshops. Under his guidance 05 research scholars awarded Ph.D. Presently 08 research scholars working under his guidance. He has a member of IAENG, IACSIT, CSTA and UACEE.



Dr. S. Nagaprasad working as a Faculty in computer science and applications, Dept. of Computer Science and Applications, Tara Government College, Sangareddy, Telangana state. He completed his B.C.A. in the year of 1998-2001 awarded from Osmania university, M.Sc (I.T.) in the year of 2001-2003 awarded from Sikkim Manipal University, Sikkim, his research work completed from Ph.D in Computer Science and Engineering, from Acharya Nagarjuna University, Sep-2015. His research areas data mining, networking, image processing, machine learning etc. In his research life he present completed 30 international journals in his research area, 15 National and International Conferences, for his research interest he around attended 10 workshops. He worked as faculty in Computer Science at S.K.N.R. Government Arts and Science College, Jagtial Telangana state for 10 years, and He worked as faculty in Computer Science at S.R.R. Government Arts and Science College, Karimnagar Telangana state for 05 years. Presently he his Guiding 5 Ph.D. scholars in Computer Science and Engineering, "Shri. Jagdishprasad Jhabarmal Tibrewala University", Jhunjhunu (Rajasthan).

Dr. Manju Khari an Assistant Professor in Ambedkar Institute of Advanced Communication Technology and Research, Under Govt. Of NCT Delhi affiliated with Guru Gobind Singh Indraprastha University, Delhi, India. She is also the Professor- In-charge of the IT Services of the Institute and has experience of more than twelve years in Network Planning & Management. She holds a Ph.D. in Computer Science & Engineering from National Institute of Technology Patna and she received her master's degree in Information Security from Ambedkar Institute of Advanced Communication Technology and Research, formally this institute is known as Ambedkar Institute of Technology affiliated with Guru Gobind Singh Indraprastha University, Delhi, India. Her research interests are software testing, software quality, software metrics, information security, optimization and nature-inspired algorithm. She has 70 published papers in refereed National/International Journals & Conferences (viz. IEEE, ACM, Springer, Inderscience, and Elsevier), 06 book chapters in a springer. She is also co-author of two books published by NCERT of secondary and senior secondary school.

