VidyaJyothi Institute of Technology

(Approved by AICTE, New Delhi, Accredited by NAAC, Permanently Affiliated to JNTUH, Hyderabad)

An Autonomous Institution

Aziznagar Gate, ChilkurBalaji Road,
Hyderabad – 500075, Telangana, India

www.vjit.ac.in

Department of Information Technology

Course file

Regulations : R19

Batch : 2019 - 2023

Academic year : 2020 - 2021

Program : B.Tech

Course name : JAVA PROGRMMING

Year/ Sem : II/II

Course Code : A3G511

Pre-Requisites : PPS - I & II

Course Coordinator : D. Sravanthi.

Index

INDEX

Syllabus

JAVA PROGRAMMING

II Year B.Tech. IT - II Sem

L	T	P	C
3	0	0	0

Course Outcomes:

At the end of the course student would be able to

1. Understand OOP concepts to apply basic Java constructs.
2. Analyze different forms of inheritance and usage of Exception Handling
3. Understand the different kinds of file I/O,Multithreading in complex Java programs, and usage of Container classes
4. Contrast different GUI layouts and design GUI applications
5. Construct a full-fledged Java GUI application, and Applet with database connectivity

UNIT - I

Java Basics:

History of Java, Java buzzwords, data types, variables, scope and life time of variables, arrays, operators, expressions, control statements, type conversion and casting, simple java program

Fundamentals of Object Oriented Programming:

Object-Oriented Paradigm, Basic Concepts of Object Oriented Programming, Applications of OOP. Concepts of classes, objects, constructors, methods, access control, this keyword, garbage collection, overloading methods and constructors, parameter passing, recursion, static keyword, nested and inner classes, Strings, Object class.

UNIT - II

Inheritance & Polymorphism:

Introduction, Forms of Inheritance - specialization, specification, construction, extension, limitation, combination, Member access rules, super keyword, polymorphism-method overriding, abstract classes, final keyword.

Interfaces and Packages:

Introduction to Interfaces, differences between abstract classes and interfaces, multiple inheritance through interfaces, Creating and accessing a package, Understanding CLASSPATH, importing packages.

Exception handling:

Concepts of exception handling, exception hierarchy, built in exceptions, usage of try, catch, finally, throw, and throws, creating own exception sub classes.

UNIT - III

Files:

Introduction to I/O Streams: Byte Streams, Character Streams. File I/O.

Multi threading: Differences between multi threading and multitasking, thread life cycle,

creating threads, thread priorities, synchronizing threads, inter thread communication. Java.util package- Collection Interfaces: List, Map, Set. The Collection classes: LinkedList, HashMap, TreeSet, StringTokenizer, Date, Random, Scanner.

UNIT - IV

AWT:

Class hierarchy, Component, Container, Panel, Window, Frame, Graphics.

AWT controls:

Labels, Button, Scrollbar, Text Components, Checkbox, CheckboxGroup, Choice, List, Panes – ScrollPane, Dialog and Menu Bar.

Event Handling:

Events, Event sources, Event classes, Event Listeners, Delegation event model, handling mouse and keyboard events, Adapterclasses.

UNIT - V

Layout Manager:

Border, Grid, Flow, Card and Gridbag.

Applets:

Concepts of Applets, life cycle of an applet, creating applets, passing parameters toapplets.

JDBC Connectivity:

JDBC Type 1 to 4 Drivers, connection establishment, Query Execution.

Text Books:

1. Java- the complete reference, Seventh edition, Herbert Schildt, Tata McGraw Hill.
2. Database Programming with JDBC & JAVA, Second Edition, George Reese, O'Reilly Media.

Reference Books:

1. Programming in JAVA, Second Edition, OXFORD Higher Education.
2. Thinking in Java Fourth Edition, Bruce Eckel
3. Introduction to Java programming, Y. Daniel Liang, Pearson Education.
4. Understanding OOP with Java, updated edition, T. Budd, Pearson Education.

Program Educational Objectives & Program Outcomes

Vidya Jyothi Institute of Technology
(Affiliated to JNTUH)
AzizmagarGate,C.B.Post,Hyderabad-500 075
DEPARTMENT OF INFORMATION TECHNOLOGY

Program Educational Objectives

(PEOs)

PEO1: Core Capabilities / Competence: Impart profound knowledge in humanities and basic sciences along with core engineering concepts for practical understanding and project development.

PEO2: Career Advancement: Enrich analytical and industry based technical skills through ICT for accomplishing research, higher education and entrepreneurship

PEO3: Life-Long Learning: Infuse life-long learning, professional ethics, adaptation to innovation and effective communication skills with a sense of social awareness.

Vidya Jyothi Institute of Technology

DEPARTMENT OF INFORMATION TECHNOLOGY

Program Outcomes

PO1	**Engineering Knowledge**: Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem Analysis**: Identify, Formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
PO3	**Design/Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural, societal and environmental considerations.
PO4	**Conduct Investigations of Complex problems**: Use research–based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
PO5	**Modern Tool Usage**: Create, Select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The Engineer and Society**: Apply Reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues, and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and Sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of and need for sustainable development.
PO8	**Ethics**: Apply Ethical Principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and Teamwork**: Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large such as, being able to comprehend and with write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project Management and Finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team to manage projects and in multi disciplinary environments.
PO12	**Life-Long Learning**: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

Vidya Jyothi Institute of Technology
(Affiliated to JNTUH)
AziznagarGate,C.B.Post,Hyderabad-500 075
DEPARTMENT OF INFORMATION TECHNOLOGY

Program Specific Outcomes

(PSOs)

PSO1: Enhanced ability in applying mathematical abstractions and algorithmic design along with programming tools to solve complexities involved in efficient programming.

PSO2: Developed effective software skills and documentation ability for graduates to become employable/ higher studies/ Entrepreneur/ Researcher.

Course Outcomes (Cos)

Course Outcomes:

1. Understand OOP concepts to apply basic Java constructs.
2. Analyze different forms of inheritance and usage of Exception Handling
3. Understand the different kinds of file I/O,Multithreading in complex Java programs, and usage of Container classes
4. Contrast different GUI layouts and design GUI applications
5. Construct a full-fledged Java GUI application, and Applet with database connectivity

Mapping of Course Outcomes, POs and PSOs

Articulation matrix of Course outcomes with Pos

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	3	3	3	2	1	1	3	1	1	2
CO2	3	3	3	3	3	2	1	1	3	1	1	2
CO3	3	3	3	3	3	2	1	1	3	2	1	2
CO4	3	3	3	3	3	2	1	1	3	2	1	2
CO5	3	3	3	3	3	2	1	1	3	2	1	2

Articulation matrix of Course outcomes with PSOs

	PSO1	PSO2
CO1	3	3
CO2	3	3
CO3	3	3
CO4	3	3
CO5	3	3

Articulation matrix of Course outcomes with Pos

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	3	3	3	2	1	1	3	1	1	2
CO2	3	3	3	3	3	2	1	1	3	1	1	2
CO3	3	3	3	3	3	2	1	1	3	2	1	2
CO4	3	3	3	3	3	2	1	1	3	2	1	2
CO5	3	3	3	3	3	2	1	1	3	2	1	2

Articulation matrix of Course outcomes with PSOs

	PSO1	PSO2
CO1	3	3
CO2	3	3
CO3	3	3
CO4	3	3
CO5	3	3

Lecture Plan /Teaching Plan

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

DEPARTMENT OF INFORMATION TECHNOLOGY

Name : D. Sravanthi **Class:** IIB.Tech, IISEM

Subject : Java Programming

Lecture	Topics to be Covered	Suggested Books	Expected Date	Actual Date
	UNIT -- I **Java Basics**			
L1	History of Java, Java buzzwords	T1(Chapter 1)	30/03/2021	30/03/2021
L2	Data types, variables, scope and life time of variables	T1(Chapter 3)	31/03/2021	06/04/2021
L3	Arrays, operators, expressions, control statements	T1(Chapter 3, 5)	06/04/2021	7/4/21
L4	Type conversion and casting, simple java program	T1(Chapter 2, 3)	07/04/2021	9/4/21
L5	Object-Oriented Paradigm	T1(Chapter 2)	09/04/2021	12/4/21
L6	Basic Concepts of Object Oriented Programming- Objects and Classes	T1(Chapter 1)	12/04/2021	16/4/21
L7	Applications of OOPs	T1(Chapter 1)	16/04/2021	19/4/21
L8	Concepts of classes, objects, constructors, methods	T1(Chapter 6)	19/04/2021	20/4/21
L9	Access control, this keyword, garbage collection	T1(Chapter 6)	20/04/2021	23/4/21
L10	Overloading methods and constructors, parameter passing,	T1(Chapter 6)	23/04/2021	26/4/21
L11	Recursion, nested and inner classes, Strings	T1(Chapter 6)	26/04/2021	24/4/21
L12	Object Class	T1(Chapter 8)	27/04/2021	28/4/21
L13	Tutorial Class		28/04/2021	30/4/21
	UNIT -- II **Inheritance& Polymorphism**			
L14	Introduction, Forms of inheritance-specialization, specification, construction, extension	T1(Chapter 8)	30/4/21	30/4/21
L15	Limitation, combination, Member access rules, super keyword	T1(Chapter 8)	1/5/21	1/5/21
L16	polymorphism- method overriding, abstract classes, Final keyword	T1(Chapter 8)	3/5/21	3/5/21
	Interfaces and Packages		4/5/21	4/5/21
L17	Introduction to Interfaces	T1(Chapter 9)	7/5/21	7/5/21
L18	Differences between abstract classes and interfaces	T1(Chapter 9)	8/5/21	8/5/21
L19	Multiple inheritance through interfaces	T1(Chapter 9)	15/5/21	15/5/21

L20	Creating and Accessing a Package	T1(Chapter 9)	16/5/21	6/5/21
L21	Understanding CLASSPATH, importing packages	T1(Chapter 9)	18/5/21	14/5/21
	Exception handling			
L22	Concepts of exception handling, exception hierarchy,	T1(Chapter 10)	17/5/21	18/5/21
L23	Built in exceptions, usage of try, catch, finally, throw and throws	T1(Chapter 10)	18/5/21	19/5/21
L24	Creating own exception sub classes	T1(Chapter 10)	20/5/21	20/5/21
L25	Tutorial Class		31/5/21	21/5/21
	UNIT -- III **Files**			
L25	Introduction to I/O Streams: Byte Streams	T1(Chapter 19)	2/6/21	31/5/21
L26	Character Streams, File I/O	T1(Chapter 19)	4/6/21	1/6/21
	Multi threading			
L27	Differences between multi threading and multitasking	T1(Chapter 11)	7/6/21	2/6/21
L28	Thread life cycle	T1(Chapter 11)	16/6/21	4/6/21
L29	Creating threads, thread priorities	T1(Chapter 11)	19/6/21	7/6/21
L30	Synchronizing threads	T1(Chapter 11)	22/6/21	19/6/21
L31	Inter thread communication	T1(Chapter 11)	23/6/21	21/6/21
	java.util Package			
L32	The Collection Interface: List, Map, Set	T1(Chapter 17)	1/7/21	22/6/21
L33	The Collection class: LinkedList Class, HashMap Class, TreeSet Class	T1(Chapter 17)	6/7/21	23/6/21
L34	StringTokenizer, Date, Random, Scanner.	T1(Chapter 18)	9/7/21	24/6/21
	UNIT -- IV **AWT**			
L35	Class hierarchy	T1(Chapter 23)	14/7/21	1/7/21
L36	Component, Container, Panel	T1(Chapter 23)	20/7/21	6/7/21
L37	Window	T1(Chapter 23)	23/7/21	9/7/21
L38	Frame	T1(Chapter 23)	28/7/21	14/7/21
L39	Graphics	T1(Chapter 23)	30/7/21	20/7/21
	AWT controls			
L40	Labels, Button, Scrollbar, Text components	T1(Chapter 24)	3/8/21	25/7/21
L41	CheckBox, CheckBoxGroup, Choice, List	T1(Chapter 24)	4/8/21	28/7/21
L42	Pane- ScrollPane, Dialog, MenuBar	T1(Chapter 24)	5/8/21	30/7/21
L43	Programs on AWT controls	T1(Chapter 24)	6/8/21	3/8/21

	Programs on AWT controls	T1(Chapter 24)	07/8/21	4/8/21
	Event Handling			
L45	Events, Event sources, Event classes	T1(Chapter 22)	08/8/21	5/8/21
L46	Event Listeners, Delegation event model	T1(Chapter 22)	09/8/21	6/8/21
L47	Handling mouse and keyboard events, Adapter classes,	T1(Chapter 22)	10/8/21	06/8/21
L48	Tutorial Class	T1(Chapter 22)	11/8/21	7/8/21
	UNIT -- V			
	Layout Managers			
L49	Layout manager types – Border, Grid, Flow, Card and GridBag Layouts	T1(Chapter 23)	12/8/21	08/8/21
	Applets			
L50	Concepts of Applets	T1(Chapter 21)	12/8/21	09/8/21
L51	Differences between applets and applications	T1(Chapter 21)	13/8/21	10/8/21
L52	Life cycle of an applet, Create Applets	T1(Chapter 21)	14/8/21	4/8/21
L53	Passing parameters to applets	T1(Chapter 21)	14/8/21	12/8/21
	JDBC Connectivity			
L54	JDBC Type 1 to 4 Drivers	T2(Chapter 3)	14/8/21	13/8/21
L55	Connection establishment, Query Execution, JDBC Programs	T2(Chapter 4)	14/8/21	14/8/21

Unit wise Date of Completion and Remarks

	26/4/24
Remarks:	– completed
Unit – II	
Date:	21/5/24
Remarks:	– completed
Unit – III	
Date:	24/6/24
Remarks:	– completed
Unit – IV	
Date:	23/7/24
Remarks:	– completed
Unit – V	
Date:	1/8/24
Remarks:	– completed

Unit wise Assignment Questions

Unit Wise Assignments (With different Levels of thinking – Blooms Taxonomy and Course Outcomes)

Unit – I	
1	Discuss the various characteristics of object oriented programming concepts. (Level-2, CO-1)
2	Explain about different loop control statements and conditional statements with an examples (Level-2, CO-1)
3	Discuss about the features of constructors and constructor overloading and with an example (Level-2, CO-1)
Unit – II	
1	Differentiate Character Streams and Byte Streams (Level-4, CO-2)
2	List different types of inheritances in java? Explain each of them in detail with an example program. (Level-1, CO-2)
Unit – III	
1	Explain with an example how thread class methods can be used to control the behavior of a thread? (Level-2, CO-3)
2	Explain inter thread communication in detail. (Level-2, CO-3)
Unit – IV	
1	Write a program to handle mouse events and mouse motion events. (Level-1, CO-4)
2	What are different Layouts? Explain with example (Level-1, CO-4)
Unit – V	
1	Write a program to develop calculator (Level-1, CO-5)
2	How to establish a connection in JDBC. Explain. (Level-2, CO-5)

Unit-V	
1	What is JDBC? Explain different types of JDBC drivers. (Level-2) (CO-5)
2	Write a Java program to display the message on the applet wherever mouse click occurs. (Level-4) (CO-5)
3	Write short note on following components. (a) Label (b) TextField (c) TextArea (d) List (e) Choice (f) Button (g) Checkbox (h) MenuBar (Level-1) (CO-5)

Access Modifier

20715A1201
Krishna Prasad

* Protected Access Modifier -

when methods and data members are declared "protected" we can access them within the same package as as from "subclasses."

```
* class Animal {          // Protected method
    Protected void display() {
        system.out.println("I am an animal");
    }
}
class Dog extends Animal {
    Public static void main(string [] args) {
        // create an object of Dog class
        Dog dog = new Dog();

        dog.display();
    }
}
```

output:

I am an animal

* we can't declare classes / Interfaces protected in Java

* Public Access Modifier –

when methods, variables, classes, and so on are declared "Public", then we can access them from anywhere.

* It has no scope restriction.

```
* Public class Animal {        // Public class
    Public int legcount;        // Public variable
    Public void display() {     // Public method
      system.out.println("I am an animal.");
    system.out.println("I have "+leg count+"legs.");
    }
}

Public class main {             // Main.Java
  Public static void main(string[] args) {
    Animal animal = new Animal(); // accessing Public class
    animal.legcount = 4;        // accessing Public variable
    animal.display();           // accessing Public method.
  }
}
```

Output:

I am an animal
I have 4 legs.

* Public can be accessed anywhere.

* Default access modifier—

```
Package P1;

class Kr
 {
   Void display()
   {
     System.out.println("Hello world!");
   }
 }

Package P2;
import P1.*;

class Kr New
{
  Public static Void main (string args[])
  {

      Kr obj = new kr();
        obj.displqy();
  }
}
```

output:

compile time error

1BM21AI245
Preethi

Assignment

```
// a java programm used to analyse student details using Abstract window
toolkit(AWT) (GUI)
import java.awt.*;
class Assignment
{
        Assignment()
        {
        Frame f = new Frame("MY ASSIGNMENT");
        Label l1 = new Label("studentID: ");
        Label l2 = new Label("password: ");
        TextField t1 = new TextField(26);
        TextField t2 = new TextField(26);
        //t2.setEchoChar("*");
        Button b = new Button("Login");
        f.add(l1);
        f.add(t1);
        f.add(l2);
        f.add(t2);
        f.add(b);


        Label l3 = new Label("student_branch: ");
        CheckboxGroup cg = new CheckboxGroup();
        Checkbox c1 = new Checkbox("IT",cg,true);
        Checkbox c2 = new Checkbox("CSE",cg,false);
        Checkbox c3 = new Checkbox("ECE",cg,false);
        Checkbox c4 = new Checkbox("EEE",cg,false);
        Checkbox c5 = new Checkbox("MECH",cg,false);
        Checkbox c6 = new Checkbox("AI",cg,false);
        Checkbox c7 = new Checkbox("CIVIL",cg,false);
        c1.setBounds(400,100,50,50);
        c2.setBounds(500,150,100,100);
        c3.setBounds(600,200,150,150);
        c4.setBounds(700,250,200,200);
        c5.setBounds(800,300,250,250);
        c6.setBounds(900,350,300,300);
        c7.setBounds(1000,400,350,350);
        f.add(l3);
        f.add(c1);
        f.add(c2);
        f.add(c3);
        f.add(c4);
        f.add(c5);
        f.add(c6);
        f.add(c7);

        Label l7 = new Label("Gender: ");
        CheckboxGroup cg1 = new CheckboxGroup();
        Checkbox cb1 = new Checkbox("male",cg1,true);
        Checkbox cb2 = new Checkbox("female",cg1,false);
        cb1.setBounds(900,450,50,50);
        cb2.setBounds(1000,500,50,50);
```

```
        f.add(l7);
        f.add(cb1);
        f.add(cb2);


        Label l4 = new Label("course enroll by student: ");
        List a1 = new List(7);
        a1.setBounds(600,550,50,50);
        a1.add("C");
        a1.add("C++");
        a1.add("Python");
        a1.add("Java");
        a1.add("Machine learning");
    a1.add("Data Science");
    a1.add("Web devlopment");
    f.add(l4);
    f.add(a1);

    Label l5 = new Label("student Feedback: ");
    TextArea ta = new TextArea(20,10);
    ta.setBounds(600,600,50,50);
    f.add(l5);
    f.add(ta);

        f.setSize(400,400);
        f.setVisible(true);
        f.setBackground(Color.gray);
        f.setLayout(new FlowLayout());


        }
        public static void main(String[] args)
        {
                Assignment as = new Assignment();

        }
}
```

Unit wise Question Bank

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

Department of Information Technology

II B.Tech, II Sem (R19)

Java Programming - Question Bank

Faculty : P. Lakshmi Sony **Phone** no: 9908545005

UNIT – I

Short Answer Questions

1. Explain briefly about history of java.
2. Summarize standard Data types in JAVA.
3. What is a constant? Explain different types of constants.
4. What is a Variable? Describe scope and life time of variables.
5. Explain ragged arrays in JAVA
6. What is a byte code in JAVA?
7. Define type conversion.
8. What is type casting? Explain.
9. List OOP concepts
10. Distinguish between procedural language and OOP language
11. Define Encapsulation.
12. Define Inheritance.
13. Define Polymorphism.
14. List advantages and disadvantages of OOPs.
15. List the applications of OOPs.
16. Discuss Inner classes with necessary syntax
17. Define constructor. List the types of constructors
18. Discuss recursion with example.
19. List out the access control modifiers and explain in brief.
20. Define garbage collection with necessary syntax
21. Discuss static keyword for variables
22. Why we use finalize() method ?
23. Explain How to pass parameters with an example
24. What is Object class ? List the methods

Long Answer Questions

1. Discuss the various characteristics of object oriented programming concepts.
2. Explain the features (buzzwords) of Java.
3. Explain the importance of "this" keyword with an example.
4. What is an Array? Explain Declaration & Initialization of ID and 2D array with an example

26. Differentiate throw and throws with necessary syntax
27. Discuss the purpose of finally block.

Long Answer Questions

1. List different types of inheritances in java. Explain each of them in detail with an example program.
2. Explain the Uses of "Super" keyword with example code.
3. Define dynamic binding. Explain with an example.
4. Explain method overriding. Demonstrate with an example program.
5. Discuss in detail about creating and importing user defined packages with an example. Also explain the advantages of packages.
6. What are various Member access rules ? Explain with an example.
7. Define interface. Justify how interfaces support multiple inheritance.
8. Compare and contrastoverriding and overloading with an example program
9. Explain the following
   a) finalkeywordb)finalize() method c)finally block
10. What is "final" keyword? Explain final with inheritance
11. Explain exception hierarchy in detail.
12. Explain exception handling mechanism with all the key words. Give example program
13. Demonstrate creating user defined Exception with a program.

UNIT – III

Short Answer Questions

1. Define stream.List the standard IO streams
2. Differentiate InputStrean and OutputStream
3. Differentiate ByteStream and Character Stream
4. Write a program to demonstrate console IO operations.
5. Write the syntax and an example for different forms of read( ) operation
6. Write the syntax and an example for different forms of write( ) operation
7. List the mandatory methods to implement threads
8. Differentiate process and a thread.
9. List and explain thread states.
10. What are the different ways to create a thread?
11. List the methods for inter-thread communication
12. What are the pre defined thread priorities?
13. How threads are synchronized?
14. Explain about the alive() and join() methods.
15. List the Thread class methods.
16. Differentiate Thread class and Runnable interface for creating threads.
17. Define Collection Class and its framework.

14. Differentiate MenuBar and MenuItem write necessary syntax.
15. Define ScollPane. Write the syntax to create ScrollPane
16. Write the syntax to create Dialog box
17. Explain about Events, Event sources, Event classes
18. Differentiate Event Listeners and Event classes
19. List the methods of MouseListener interface
20. List the Methods of KeyListener interface
21. Explain ActionEvent Class
22. Define Adapter class with necessary syntax and an example

Long Answer Questions

1. Explain AWT class hierarchy
2. Write a program to create a Window using AWT
3. List the methods with necessary syntax to Draw and Fill Rectangle and an Arc using Graphics object
4. Explain Delegation Event Model in java with an example
5. Write a program to demonstrate Mouse Events
6. Write a program to demonstrate Key Events
7. Explain ActionEvent class and ActionListener interface with an example program
8. Write a program to demonstrate MouseAdapter class.
9. Write a program to create TextArea with scrollbars
10. Write a program to display an Application Form using required AWT controls.

UNIT – V

Short Answer Questions

1. Define Layout manager with its uses.
2. List the various layouts in java
3. What is a Flow Layout? Write the necessary syntax
4. List the alignment constants of FlowLayout
5. Explain Grid Layout with its syntax.
6. Differentiate border layout and grid layout
7. List the constants of BorderLayout
8. Differentiate Grid and Gridbag Layout
9. List Gridbag Layout constraints
10. What is the purpose of card Layout? Write the necessary syntax for creating card layout.
11. Define Applet. List the life cycle methods of an Applet
12. What is the purpose of Appletviewer tool?
13. Explain the different approaches of running an applet
14. Why we use <param >tag for an applet
15. Explain paint() method of an applet

Mid Question Papers

Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

II Year B.Tech II Semester 1st Mid Exam

Branch:	Duration: 90Min
Sub: Java Programming	Marks: 20
Date: 18-02-2020	Session:FN

Course Outcomes:

1.Understand theObjectOrientedProgramming concepts.

2.Applythe concepts of package and interfaces.

3.Apply the concepts of Exceptions and mult ithreading.

4.Analyze GUI applications and AWT using Frames.

5.Design the programs using Applet and JDBC Concepts.

Bloom's Level:

Remember	I
Understand	II
Apply	III
Analyze	IV
Evaluate	V
Create	VI

PART-A (3Q×2M =6Marks)		Course Outcomes		Bloom's Level	Marks
ANSWER ALL THE QUESTIONS		CO	PO		
1.i)	Define polymorphism . List the types	1	1	1	2
	[OR]				
ii)	Declare and Intialize ID and 2DArrays	1	1	3	2
2.i)	Differentiate throw and throws clause	2	2	3	2
	[OR]				
ii)	Define inheritance. Give syntax and example	2	1	2	2
3.i)	List the thread states	3	1	2	2
	[OR]				
ii)	Differentiate process and thread	3			2

PART-B (5+5+4= 14 Marks)		Course Outcomes		Bloom's Level	Marks
ANSWER ALL THE QUESTIONS		CO	PO		
4.i.a)	Explain the Buzzwords of java		1	2	3
b)	Differentiate break and continue statements with an example		2	2	2
	[OR]				
ii.a)	Define costructor. Demonstarte constructor overloading		2	3	2.5
b)	Explain Type casting and conversion with necessary syntax		1	2	2.5
5. i.a)	Define interface. Multiple inheritance is supported through interfaces, justify.		4	4	3
b)	Demonstrate multilevel inheritance with a program		3	3	2
	[OR]				
ii.	Explain Exception Handling Mechanism		1	2	5
6.i)	Demonstrate creating a thread by Extending Thread class		3	3	4
	[OR]				
ii)	Explain Thread Priorities, write a program to set and get priority of a thread		3	3	4

VJIT(A)

Controller of Exminations

DIRECTOR

Java Programming

Scheme of Evaluation

II B.Tech II Semester mid I Exam 18-02-2020

Branch : IT

Subject : Java Programming

Max Marks : 20 Marks

Instruction : D. Sravanthi

Part-A

1 i) Define Polymorphism. List the types 2M

definition - 1M

List - 1M

2 ii) Declare and intialize 1D and 2D arrays - 2M

declaration of 1D & 2D arrays 1M

intialize 1D & 2D - 1M

2. i) differentiate throw and throws clause - 2M

differentiate throw & throws - 2M

ii) define inheritance. Give syntax and example - 2M

definition inheritance - 1M

Syntax with an example - 1M

3. i) list the thread states - 2M

ii) differentiate process and thread - 2M

differentiate the process — 1M
threads — 1M

Part - B

4. i a) Explain the Buzzword of java — 3M

Buzzword of java — 3M

b) differentiate break and continue statements with an example — 2M

break and continue syntax — 1M

Example program — 1M

4. ii a) define Constructor. demonstrate Constructor overloading → 2.5M

definition of Constructor — 1M

Constructor overloading — 1.5M

b) Explain type casting and conversion with necessary syntax — 2.5M

define type casting — 1M

type conversion example and syntax — 1.5M

5 i.a) define interface. multiple ~~inheritance~~ in heritance is supported through interface justify — 3M

define interface — 1M

multiple inheritance with an example — 2M

5 i.b) Demonstrate multilevel inheritance with a program — 2M

multilevel inheritance definition — 1M

Program — 1M

5. ii) Explain exception handling mechanism — 5M

6 i) Demonstrate creating a thread by extending thread class - 4M

→ definition of threads — 1M

thread class with an example program - 3M

6ii) Explain thread priorities, write a program to set and get priority of a thread — 4M

thread priorities — 1M

example program set and get priority — 3M.

Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

I,II,III & IV Year B.Tech II Semester 1st Mid Exam

Branch: II- IT **Duration: 90Min**

Sub: Java Programming **Marks: 20**

Date: Session:

Course Outcomes:

1. Understand OOP concepts to apply basic Java constructs.
2. Analyze different forms of inheritance and usage of Exception Handling
3. Understand the different kinds of file I/O,Multithreading in complex Java programs, and usage of Container classes
4. Contrast different GUI layouts and design GUI applications
5. Construct a full-fledged Java GUI application, and Applet with database connectivity

Remember	I
Understand	II
Apply	III
Analyze	IV
Evaluate	V
Create	VI

PART-A (3Q×2M =6Marks)		Course Outcomes		Bloom's Level	Marks
ANSWER ALL THE QUESTIONS		**CO**	**PO**		
1.i)	List the features of OOP concepts.	1	1, 2, 3, 4, 5, 11, 12	1	2
	[OR]				
ii)	What is method overloading? Explain with an example.	1	1, 2, 3, 4, 5, 11, 12	1	2
2.i)	Describe the use of "Super" keyword.	2	1, 2, 3, 4, 5, 11, 12	2	2
	[OR]				
ii)	Define the various steps for creating and importing packages.	2	1, 2, 3, 4, 5, 11, 12	1	2
3.i)	Differentiate Character Streams and Byte Streams.	3	1, 2, 3, 4, 5, 11, 12	2	2
	[OR]				
ii)	Define Buffered Streams and its classification.	3	1, 2, 3, 4, 5, 11, 12	1	2
PART-B (5+5+4= 14 Marks)		**Course Outcomes**		**Bloom's Level**	**Marks**
ANSWER ALL THE QUESTIONS		**CO**	**PO**		
4.i.a)	Explain the console input and output with an example.	1	1, 2, 3, 4, 5, 11, 12	3	2
b)	Explain about nested classes in detail.	1	1, 2, 3, 4, 5, 11, 12	3	3
	[OR]				
ii	Explain the importance of i)this ii) static keywords in JAVA with example.	1	1, 2, 3, 4, 5, 11, 12	3	5
5. i.a)	List different types of inheritance in java? Explain each concept with suitable program.	2	1, 2, 3, 4, 5, 11, 12	3	3
b)	Define package. Write a java program to use packages.	2	1, 2, 3, 4, 5, 11, 12	3	2
	[OR]				
ii.a)	List out the keywords used for exception handling. Write any java program to illustrate with user defined exception	2	1, 2, 3, 4, 5, 11, 12	3	5
6.i)	Illustrate the use of Byte Streams with examples	3	1, 2, 3, 4, 5, 11, 12	3	4
	[OR]				
ii)	Illustrate the use of InputStreamReader and OutputSteamWriter with suitable program.	3	1, 2, 3, 4, 5, 11, 12	3	4

VJIT(A)

Dean Exminations **DIRECTOR**

Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

II Year B.Tech II Semester 1st Mid Exam

Branch:	Duration: 90Min
Sub: Java Programming	Marks: 20
Date: 18-02-2020	Session:FN

Course Outcomes:

1.Understand theObjectOrientedProgramming concepts.
2.Applythe concepts of package and interfaces.
3.Apply the concepts of Exceptions and mult ithreading.
4.Analyze GUI applications and AWT using Frames.
5.Design the programs using Applet and JDBC Concepts.

Bloom's Level:

Remember	I
Understand	II
Apply	III
Analyze	IV
Evaluate	V
Create	VI

PART-A (3Q×2M =6Marks)		Course Outcomes		Bloom's Level	Marks
		CO	PO		
ANSWER ALL THE QUESTIONS					
1.i)	Define constructor.list the types	1	1	2	2
	[OR]				
ii)	Discuss operator precedence and associativity with an expression	1	2	3	2
2.i)	Differentiate try and catch block	2	2	2	2
	[OR]				
ii)	Explain the purpose of final keyword	2	1	2	2
3.i)	List the Thread Priority methods with syntax	3	1	2	2
	[OR]				
ii)	Differentiate process and thread	3	1		2

PART-B (5+5+4= 14 Marks)		Course Outcomes		Bloom's Level	Marks
		CO	PO		
ANSWER ALL THE QUESTIONS					
4.i.a)	Explain object oriented programming concepts	1	1	2	3
b)	Explain the concept of garbage collection	1	1	2	2
	[OR]				
ii.a)	Explain Type casting and conversion with necessary syntax	1	1	3	3
b)	Explain inner classes with necessary syntax	1	1	3	2
5. i.a)	Define package.Explain creating, importing and implementing packages with necessary code.	2	3	3	3
b)	Differentiate abstract class and an interface	2	2	2	2
	[OR]				
ii.a)	Demonstrate how to create user defined exceptions with a program	**2**	**3**	**3**	**3**
b)	Differentiate method Overloading and Overriding	2	2	4	2
6.i)	Explain the life cycle of a thread with a neat diagram	3	1	2	4
	[OR]				
ii)	Write a program to demonstrate multiple child threads	3	3	3	4

VJIT(A)

Controller of Exminations

DIRECTOR

End Exam Papers

Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)

(Aziz Nagar, C.B.Post, Hyderabad -500075) **Sub Code: A14511** **R15**

II B.Tech II Semester Supplementary Examination, October/November-2020

Subjects' Name: Java Programming **BRANCH: CSE&IT**

Time: 2 Hours **Max Marks:75**

Note: 1. Thisquestion paper contains EIGHT questions and the students are asked to answer any FIVE questions.

2. Each question carries 15 marks.

Bloom's Level:

Remember	L1	Analyze	L4
Understand	L2	Evaluate	L5
Apply	L3	Create	L6

	ANSWER ANY FIVE QUESTIONS (5Qx15M=75M)	Bloom's Level	Marks
1	(a)Explain about the Data abstraction and encapsulation with examples. (b)List the primitive data types available in Java and explain?	L2,L2	15M
2	(a) Write short note on Message Communication. (b) How does the parameter passing working in Java? write a java program to illustrate it?	L2,L3	15M
3	(a) Write a program to illustrate the uses of final keyword in inheritance? (b) Define package in Java and explain the categories of packages with suitable examples for each type.	L2,L2	15M
4	(a) With a neat diagram, describe types of inheritance in Java? (b) What is a stream? Explain Byte Stream and Character Stream?	L3,L2	15M
5	(a) Discuss exception handling mechanism in Java? (b) Write a Java program to demonstrate thread synchronization?	L2,L2	15M
6	(a) Write a Java program to create user defined exception. Create an exception called "Invalid Marks Exception". Create a method to read student marks within 50. Throw the exception on reading input marks greater than 50? (b) What are differences between implementing Runnable and extending Thread?	L3,L3	15M
7	(a) Explain about the event listeners? (b) What is an Layout manager? Explain different types of Layout managers.	L2,L2	15M
8	(a) Write a simple banner applet to demonstrate repaint( ) method? (b) How many packages are available in JDBC API?Explain it?	L2,L2	15M

Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

R18

Subject Code:A24509

B.Tech. II Year II Semester Regular Examination, OCTOBER/NOVEMBER-2020

SUBJECT : Java Programming **BRANCH :** CSE&IT

Time: 2 Hours **Max. Marks:75**

Note: This question paper contains EIGHT questions and answer any FIVE questions. Each question carries 15 marks.

Bloom's Level:

Remember	L1
Understand	L2
Apply	L3
Analyze	L4
Evaluate	L5
Create	L6

	ANSWER ANY FIVE QUESTIONS 5QX15M = 75 M	Bloom's Level	Marks
1.a)	Describe the different types of datatypes used in Java.	L2	10M
b)	Differentiate between class and object.	L2	5M
2.a)	Explain the different types of constructors with an example.	L3	9M
b)	Write a java program to find the factorial of a given number.	L2	6M
3.a)	Explain the various types of inheritance with suitable code segments.	L2	10M
b)	write the differences between abstract classes and interfaces.	L2	5M
4.a)	How to define a package? How to access, import a package.?	L3	5M
b)	Explain the usage of try, catch, finally with suitable example.	L3	10M
5.a)	Describe about Byte Streams.	L2	8M
b)	Differentiate between multi-threading and multitasking.	L2	7M
6.a)	Write about StringTokenizer.	L2	7M
b)	Explain the Date Class methods .	L3	8M
7.a)	Discuss about AWT Container class .	L2	7M
b)	What is the role of event listeners in event handling? List the Java event listeners.	L2	8M
8.a)	Write the differences between applets and applications.	L3	7M
b)	Discuss briefly about Flow layout.	L2	8M

Vidya Jyothi Institute of Technology (Autonomous)

(Accredited by NAAC & NBA, Approved By A.I.C.T.E., New Delhi, Permanently Affiliated to JNTU, Hyderabad)
(Aziz Nagar, C.B.Post, Hyderabad -500075)

R15

Subject Code:A14511

II B.Tech. II Semester Examinations - MAY 2019

SUBJECT:JAVA PROGRAMMING **BRANCH :**CSE&IT

Time: 3 Hours **Max. Marks:75**

Note: This question paper contains two Parts A and B.
Part A is compulsory which carries 25 Marks. Answer all the questions
Part B consists of 5 questions. Answer all the questions.

Bloom's Levels:

Remember	L1	Analyze	L4
Understand	L2	Evaluate	L5
Apply	L3	Create	L6

		B.L	25M
	PART - A		
	ANSWER ALL THE QUESTIONS		
1	Define Encapsulation?	L2	2M
2	What is enum data type?	L1	3M
3	What is Method Overriding?	L3	2M
4	Explain in brief about interfaces?	L2	3M
5	List the keywords used to handle the exceptions?	L5	2M
6	What is Thread Priority?	L5	3M
7	Differentiate between Multi threading and Multi tasking?	L4	2M
8	What is AWT?Draw neatly the hierarchy of AWT Classes?	L6	3M
9	How Parameters are passed to Applets?	L3	2M
10	Explain the steps in JDBC Process?	L6	3M
	PART - B	**B.L**	**50M**
	ANSWER ALL THE QUESTIONS		
11.i.a)	Explain in detail about different concepts used in Object oriented programming.	L1	5
b)	Write a java program to explain scope and lifetime of a varible	L3	5
	[OR]		
ii.a)	How objects are constructed?Explain constructor overloading with an example.	L3	5
b)	Write short notes on access specifier's in java	L4	5
12.i.a)	What is inheritance?Explain the forms of inheritance with an example.	L2	5
b)	Define abstract classes.Write a java code to explain abstract classes.	L2	5
	[OR]		
ii.a)	With the help of a java code explain how to create and access a package.	L6	5
b)	Explain the classes and interfaces of Java I/O Pacakge.	L6	5
13.i.a)	Discuss the concept of exception handling with an application of your choice.Write necessary code snippets.	L5	5
b)	Discuss about the benefits of Exception handling.	L4	5
	[OR]		
ii.a)	With a neat diagram explain the thread life cycle in detail.	L6	5
b)	What is multithreading?What are the methods available in java for inter-thread communication?Discuss with an example.	L6	5
14.i.a)	What is the role of event listeners in event handling?List the java event	L3	5
b)	Write a java program to handle the mouse events.	L3	5
	[OR]		
ii.a)	With an example explain flow and card layout.	L4	5
b)	What is frame?Explain with an example.	L6	5
15.i.a)	Write a code snippet to Text Area and Menu bar using AWT.	L6	5
b)	Differentiate between Applet and Application?	L4	5
	[OR]		
ii.a)	How parameters can be passed to applets .Explain with suitable example?	L5	5
b)	What is JDBC?Briefly explain various JDBC Drivers with syntax?	L3	5

Content Beyond Syllabus

Topics beyond Syllabus

S. No	Name of the Topic
1	Swings • Swing is a GUI widget toolkit for Java
2	Java Server Technology • Introduction to servlets and its life cycle

Unit Wise PPTs and Lecture Notes

UNIT-I

OOP Concepts

Object Oriented Programming is a paradigm that provides many concepts such as **inheritance**, **data binding**, **polymorphism** etc.

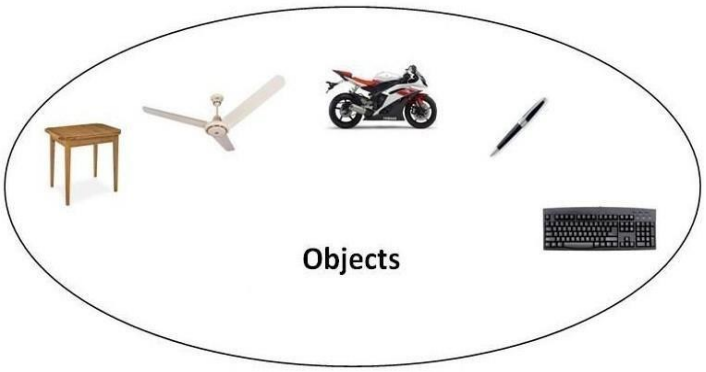
Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object is known as truly object-oriented programming language.

Smalltalk is considered as the first truly object-oriented programming language.

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation



Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Benefits of Inheritance

- One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses. Where equivalent code exists in two related classes, the hierarchy can usually be refactored to move the common code up to a mutual superclass. This also tends to result in a better organization of code and smaller, simpler compilation units.
  - Inheritance can also make application code more flexible to change because classes that inherit from a common superclass can be used interchangeably. If the return type of a method is superclass
  - **Reusability** - facility to use public methods of base class without rewriting the same.
  - **Extensibility** - extending the base class logic as per business logic of the derived class.

- **Data hiding -** base class can decide to keep some data private so that it cannot be altered by the derived class

Procedural and object oriented programming paradigms

Features	Procedural Oriented Programming (POP)	Object Oriented Programming (OOPS)
Divided into	In POP, program is divided into smaller parts called as functions.	in OOPs , the program is divided into parts known as **objects**.
Importance	In POP, importance is not given to **data** but to functions as well as **sequence** of actions to be done.	In OOPs, Importance is given to the data rather than procedures or functions because it works as a **real world**.
Approach	POP follows **Top Down approach**.	OOPs follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOPs has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOPs, objects can move and communicate with each other through member functions.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOPs, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOPs provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOPs, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	C, VB, FORTRAN, Pascal.	C++, JAVA, VB.NET, C#.NET.

Java Programming- History of Java

The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Originally designed for small, embedded systems in electronic appliances like set- top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling and file extension was .gt.

4) After that, it was called Oak and was developed as a part of the Green project.

Java Version History

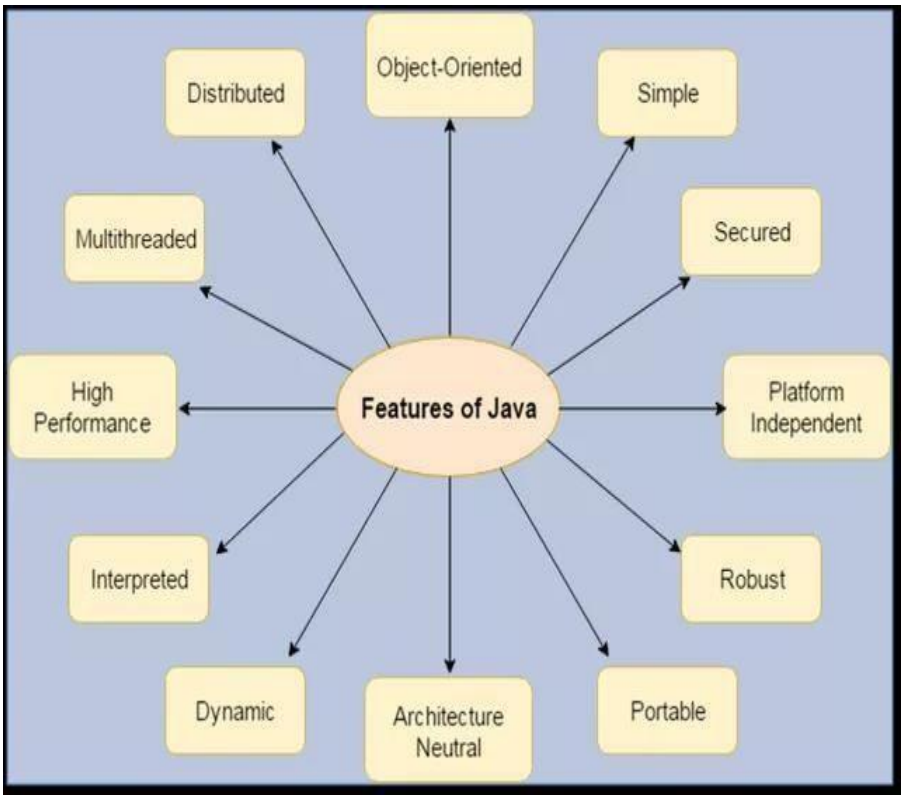
There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995) 2. JDK 1.0 (23rd Jan, 1996) 3. JDK 1.1 (19th Feb, 1997) 4. J2SE 1.2 (8th Dec, 1998) 5. J2SE 1.3 (8th May, 2000) 6. J2SE 1.4 (6th Feb, 2002) 7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)

Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed



Java Comments

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

Types of Java Comments

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

Java Single Line Comment

The single line comment is used to comment only one line.

Syntax:

```
1. //This is single line comment
```

Example:

```
public class CommentExample1 {
public static void main(String[] args) {
   int i=10;//Here, i is a variable
   System.out.println(i);
}
}
```

Output:

```
10
```

Java Multi Line Comment

The multi line comment is used to comment multiple lines of code.

Syntax:

```
 /*
 This
 is
 multi line
 comment
 */
```

Example:

```
public class CommentExample2 {
public static void main(String[] args) {
/* Let's declare and
 print variable in java. */
   int i=10;
   System.out.println(i);
} }
```

Output:

```
10
```

Java Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

Syntax:

```
/**
Thi
s is
documentatio
n comment
*/
```

Example:

```
/** The Calculator class provides methods to get addition and subtraction of given 2 numbers.*/
public class Calculator {
/** The add() method returns addition of given numbers.*/
public static int add(int a, int b){return a+b;}
/** The sub() method returns subtraction of given numbers.*/
public static int sub(int a, int b){return a-b;}
}
```

Compile it by javac tool:

```
javac Calculator.java
```

Create Documentation API by javadoc tool:

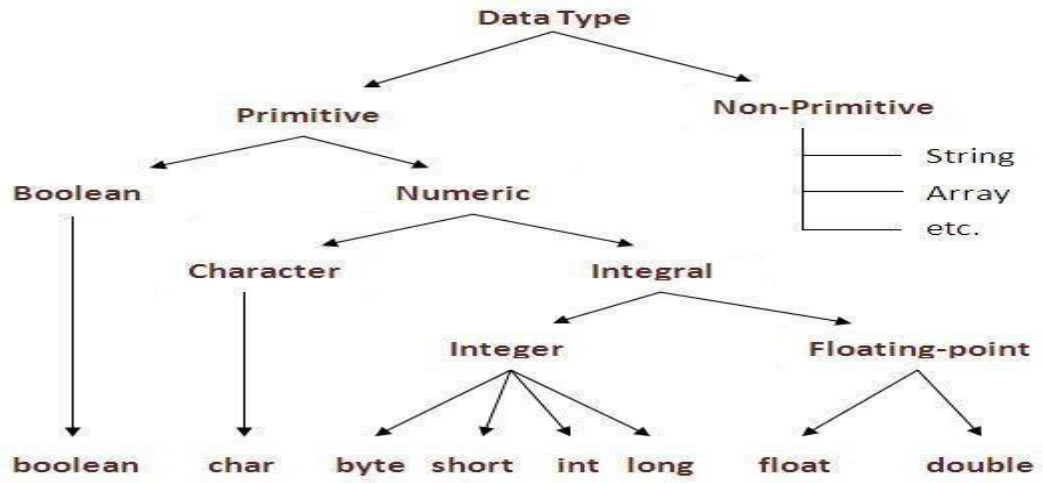
```
javadoc Calculator.java
```

Now, there will be HTML files created for your Calculator class in the current directory. Open the HTML files and see the explanation of Calculator class provided through documentation comment.

Data Types

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types



Data Type	Default Value	Default size
boolean	False	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Java Variable Example: Add Two Numbers

```
class Simple{
public static void main(String[] args){
int   a=10;
int   b=10;
```

```
int c=a+b;
System.out.println(c);
}}
```

Output:20

Variables and Data Types in Java

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

Types of Variable

There are three types of variables in java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable which is declared inside the method is called local variable.

2) Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable . It is not declared as static.

3) Static variable

A variable that is declared as static is called static variable. It cannot be local.

We will have detailed learning of these variables in next chapters.

Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Constants in Java

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 c
```

Scope and Life Time of Variables

The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before it is destroyed. Destroying variables refers to deallocating the memory that was allotted to the variables when declaring it. We have written a few classes till now. You might have observed that not all variables are the same. The ones declared in the body of a method were different from those that were declared in the class itself. There are three types of variables: instance variables, formal parameters or local variables and local variables.

Instance variables

Instance variables are those that are defined within a class itself and not in any method or constructor of the class. They are known as instance variables because every instance of the class (object) contains a copy of these variables. The scope of instance variables is determined by the access specifier that is applied to these variables. We have already seen about it earlier. The lifetime of these variables is the same as the lifetime of the object to which it belongs. Object once created do not exist for ever. They are destroyed by the garbage collector of Java when there are no more reference to that object. We shall see about Java's automatic garbage collector later on.

Argument variables

These are the variables that are defined in the header oaf constructor or a method. The scope of these variables is the method or constructor in which they are defined. The lifetime is limited to the time for which the method keeps executing. Once the method finishes execution, these variables are destroyed.

Local variables

A local variable is the one that is declared within a method or a constructor (not in the header). The scope and lifetime are limited to the method itself.

One important distinction between these three types of variables is that access specifiers can be applied to instance variables only and not to argument or local variables.

In addition to the local variables defined in a method, we also have variables that are defined in bocks life an if block and an else block. The scope and is the same as that of the block itself.

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Operators Hierarchy

Operator Precedence

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Expressions
Expressions are essential building blocks of any Java program, usually created to produce a new value, although sometimes an expression simply assigns a value to a variable. Expressions are built using values, variables, operators and method calls.

Types of Expressions

While an expression frequently produces a result, it doesn't always. There are three types of expressions in Java:

- Those that produce a value, i.e. the result of (1 + 1)
- Those that assign a variable, for example (v = 10)
- Those that have no result but might have a "side effect" because an expression can include a wide range of elements such as method invocations or increment operators that modify the state (i.e. memory) of a program.

Java Type casting and Type conversion

Widening or Automatic Type Conversion
Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

Narrowing or Explicit Conversion
If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explicit Conversion

Java Enum

Enum in java is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.

Java Enums can be thought of as classes that have fixed set of constants.

Simple example of java enum

```
class EnumExample1{
public enum Season { WINTER, SPRING, SUMMER, FALL }

public static void main(String[]
args) { for (Season s :
Season.values())
System.out.println(s);
}}
```

Output:

```
WINTER
SPRING
SUMMER
FALL
```

Control Flow Statements

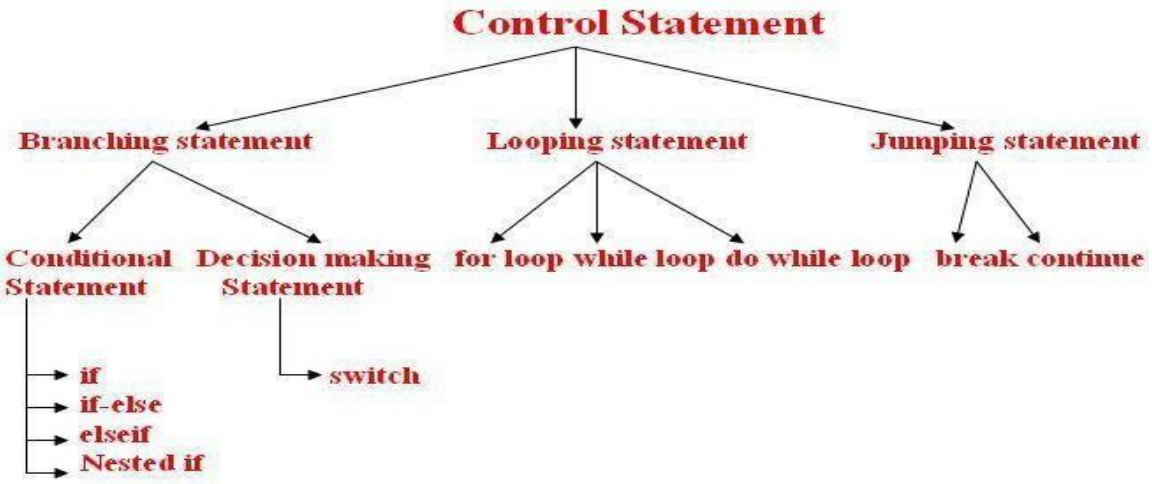
The control flow statements in Java allow you to run or skip blocks of code when special conditions are met.

The "if" Statement

The "if" statement in Java works exactly like in most programming languages. With the help of "if" you can choose to execute a specific block of code when a predefined condition is met. The structure of the "if" statement in Java looks like this:

```
if (condition) {
// execute this code
}
```

The condition is Boolean. Boolean means it may be true or false. For example you may put a mathematical equation as condition. Look at this full example:



Creating a Stand-Alone Java Application

1. Write a main method that runs your program. You can write this method anywhere. In this example, I'll write my main method in a class called Main that has no other methods. **For example:**

```
2.  public class Main
3.  {
4.    public static void main(String[] args)
5.    {
6.      Game.play();
7.    } }
```

8. Make sure your code is compiled, and that you have tested it thoroughly.
9. If you're using Windows, you will need to set your path to include Java, if you haven't done so already. This is a delicate operation. Open Explorer, and look inside C:\ProgramFiles\Java, and you should see some version of the JDK. Open this folder, and then open the bin folder. Select the complete path from the top of the Explorer window, and press Ctrl-C to copy it.

Next, find the "My Computer" icon (on your Start menu or desktop), right-click it, and select properties. Click on the Advanced tab, and then click on the Environment variables button. Look at the variables listed for all users, and click on the Path variable. Do not delete the contents of this variable! Instead, edit the contents by moving the cursor to the right end, entering a semicolon (;), and pressing Ctrl-V to paste the path you copied earlier. Then go ahead and save your changes. (If you have any Cmd windows open, you will need to close them.)

10. If you're using Windows, go to the Start menu and type "cmd" to run a program that brings up a command prompt window. If you're using a Mac or Linux machine, run the Terminal program to bring up a command prompt.
11. In Windows, type dir at the command prompt to list the contents of the current directory. On a Mac or Linux machine, type ls to do this.

12. Now we want to change to the directory/folder that contains your compiled code. Look at the listing of sub-directories within this directory, and identify which one contains your code. Type cd followed by the name of that directory, to change to that directory. For example, to change to a directory called Desktop, you would type:

cd Desktop

To change to the parent directory, type:

cd ..

Every time you change to a new directory, list the contents of that directory to see where to go next. Continue listing and changing directories until you reach the directory that contains your .class files.

13. If you compiled your program using Java 1.6, but plan to run it on a Mac, you'll need to recompile your code from the command line, by typing:

javac -target 1.5 *.java

14. Now we'll create a single JAR file containing all of the files needed to run your program.

Arrays

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar; // preferred way.
or
dataType arrayRefVar[]; // works but not preferred way.
```

Note: The style **dataType[] arrayRefVar** is preferred. The style **dataType arrayRefVar[]** comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

Example:

The following code snippets are examples of this syntax:

```
double[] myList;        // preferred way.
or
double myList[];        // works but not preferred way.
```

Creating Arrays:

You can create an array by using the new operator with the following syntax:

```
arrayRefVar = new dataType[arraySize];
```

The above statement does two things:

- It creates an array using new dataType[arraySize];
- It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

```
dataType[] arrayRefVar = new dataType[arraySize];
```

Alternatively you can create arrays as follows:

```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

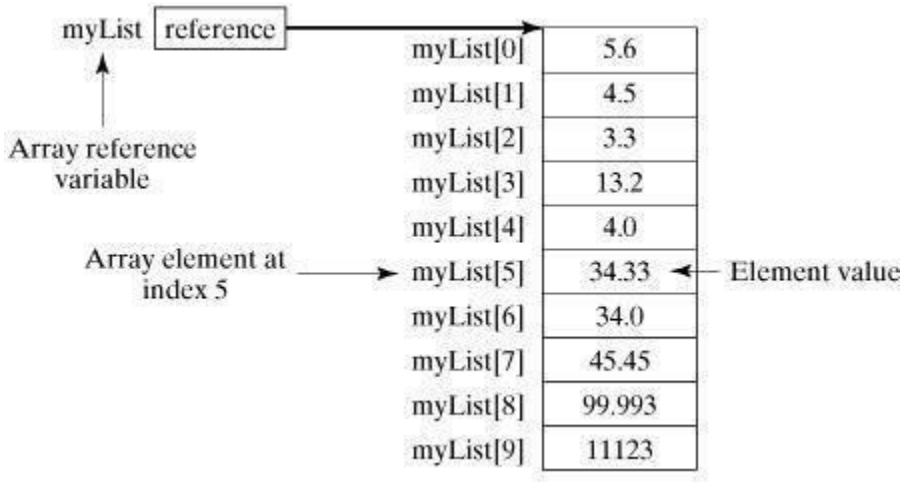
The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

```
double[] myList = new double[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Processing Arrays:

When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known.

Example:

Here is a complete example of showing how to create, initialize and process arrays:

```
public class TestArray
{
  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};
            // Print all the array elements
    for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
    }
    // Summing all elements
    double total = 0;
    for (int i = 0; i < myList.length; i++) {
      total += myList[i];
    }
    System.out.println("Total is " + total);
    // Finding the largest element
    double max = myList[0];
    for (int i = 1; i < myList.length; i++) {
      if (myList[i] > max) max = myList[i];
    }
    System.out.println("Max is " + max);
  }
}
```

This would produce the following result:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

```
public class TestArray {
public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};
 // Print all the array elements
    for (double element: myList) {
       System.out.println(element);
}}}
```

Java Console Class

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to the user.

The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.

Let's see a simple example to read text from console.

```
1.      String text=System.console().readLine();
2.      System.out.println("Text is: "+text);
```

Java Console Example

```
import java.io.Console;
class ReadStringTest{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter your name: ");
String n=c.readLine();
System.out.println("Welcome "+n);    }    }
```

Output

```
Enter your name: Nakul Jain
Welcome Nakul Jain
```

Constructors

Constructor in java is a *special type of method* that is used to initialize the object.

Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

1. <class_name>(){}

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1{
Bike1(){System.out.println("Bike is created");}
public static void main(String args[]){
Bike1 b=new Bike1();
} }
```

```
Output: Bike is created
```

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4{
 int id;
 String name;

 Student4(int i,String n){
 id = i;
 name = n;
 }
 void display(){System.out.println(id+" "+name);}

 public static void main(String args[]){
 Student4 s1 = new Student4(111,"Karan");
 Student4 s2 = new Student4(222,"Aryan");
 s1.display();
 s2.display();
} }
```

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
class Student5{
int id;
String name;
int age;
Student5(int i,String n){
id = i;
name = n;
}
Student5(int i,String n,int a){
id = i;
name = n;
age=a;
}
void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){
Student5 s1 = new Student5(111,"Karan");
Student5 s2 = new Student5(222,"Aryan",25);
s1.display();
```

```
s2.display();
} }
```

Output:

```
111 Karan 0
222 Aryan 25
```

Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.

```
class Student6{
int id;
String name;
Student6(int i,String n){
id = i;
name = n;
}

Student6(Student6 s){
id = s.id;
name =s.name;
}
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
Student6 s1 = new Student6(111,"Karan");
Student6 s2 = new Student6(s1);
s1.display();
s2.display();
} }
```

Output:

```
111 Karan
111 Karan
```

Java -Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.**println()** method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

Creating Method

Considering the following example to explain the syntax of a method −

Syntax

```
public static int methodName(int a, int b) {
   // body
}
```

Here,

- **public static** − modifier
- **int** − return type
- **methodName** − name of the method
- **a, b** − formal parameters
- **int a, int b** − list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes −

- **modifier** − It defines the access type of the method and it is optional to use.
- **returnType** − Method may return a value.
- **nameOfMethod** − This is the method name. The method signature consists of the method

name and the parameter list.

- **Parameter List** − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **method body** − The method body defines what the method does with the statements.

Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

```
class Operation{
 int data=50;
 void change(int data){
 data=data+100;//changes will be in the local variable only
 }
 public static void main(String args[]){
   Operation op=new Operation();
   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);
 }
}
```

```
Output:before change 50
       after change 50
```

In Java, parameters are always passed by value. For example, following program prints i = 10, j = 20.

```
// Test.java
class Test {
   // swap() doesn't swap i and j
   public static void swap(Integer i, Integer j) {
      Integer temp = new Integer(i);
      i = j;
      j = temp;
   }
   public static void main(String[] args) {
      Integer i = new Integer(10);
      Integer j = new Integer(20);
      swap(i, j);
      System.out.println("i = " + i + ", j = " + j);
```

```
  }
}
```

Static Fields and Methods

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

Java static variable

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable

```
1. class Student{
2.     int rollno;
3.     String name;
4.     String
college="ITS"; 5. }
```

Example of static variable

```
//Program of static variable
class Student8{
  int rollno;
```

```
  String name;
  static String college
  ="ITS"; Student8(int
  r,String n){ rollno = r;
  name = n;
  }
void display (){System.out.println(rollno+" "+name+" "+college);}
public static void main(String args[]){
Student8 s1 = new Student8(111,"Karan");
Student8 s2 = new Student8(222,"Aryan");

s1.display();
s2.display();
} }
```

Output:111 Karan ITS

222 Aryan ITS

Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- static method can access static data member and can change the value of it.

Example of static method

```
//Program of changing the common property of all objects(static field).

class Student9{
    int rollno;
    String name;
    static String college = "ITS";
    static void change(){
    college = "BBDIT";
    }
    Student9(int r, String n){
    rollno = r;
    name = n;
```

```
}
 void display (){System.out.println(rollno+" "+name+" "+college);}
public static void main(String args[]){
Student9.change();
Student9 s1 = new Student9 (111,"Karan");
Student9 s2 = new Student9 (222,"Aryan");
Student9 s3 = new Student9 (333,"Sonoo");
s1.display();
s2.display();
s3.display();
} }
```

```
Output:111 Karan BBDIT
    222 Aryan BBDIT
     333 Sonoo BBDIT
```

Java static block

- Is used to initialize the static data member.
- It is executed before main method at the time of class loading.

Example of static block

```
class A2{
 static{System.out.println("static block is invoked");}
 public static void main(String args[]){
 System.out.println("Hello main");
 } }
```

Output: static block is invoked
Hello main

Access Control

Access Modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

private access modifier

The private access modifier is accessible only within class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");} }
public class Simple{
 public static void main(String args[]){
  A obj=new A();
  System.out.println(obj.data);//Compile Time Error
  obj.msg();//Compile Time Error
  } }
```

2) default access modifier

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only within package.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;
```

```
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error }  }
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) protected access modifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");} }
//save by B.java
package mypack;
import pack.*;
class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  } }
```

Output:Hello

4) public access modifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
//save by A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");} }
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  } }
```

Output:Hello

Understanding all java access modifiers

Let's understand the access modifiers by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

this keyword in java

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.

4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:
111 ankit 5000
112 sumit 6000

Difference between constructor and method in java

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be

	same as class name.

There are many differences between constructors and methods. They are given belo

Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
class Student5{
int id; String
name; int
age;
Student5(int i,String n){
id = i;
name = n;
}
Student5(int i,String n,int a){
id = i;
name = n;
age=a;
}
void display(){System.out.println(id+" "+name+" "+age);}

public static void main(String args[]){
Student5 s1 = new Student5(111,"Karan");
Student5 s2 = new Student5(222,"Aryan",25);
s1.display();
s2.display();
}
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

Method Overloading in java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

Output:

```
22
33
```

Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

Java Recursion Example 1: Factorial Number

```
public class RecursionExample3 {
  static int factorial(int n){
   if (n == 1)
     return 1;
     else
    return(n * factorial(n-1));
    } }
public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
} }
```

Output:

```
Factorial of 5 is: 120
```

Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}
```

Simple Example of garbage collection in java

```
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 } }
```

```
object is garbage collected
object is garbage collected
```

Java String

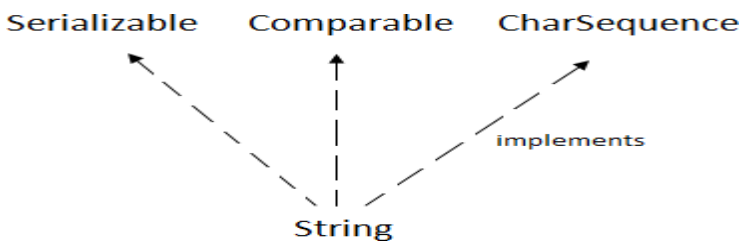
string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);
```

ssame as:

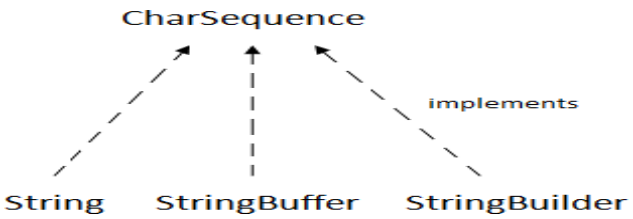
```
1. String s="javatpoint";
```

2. **Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.
3. The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.



CharSequence Interface

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.



The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

There are two ways to create String object:

1. By string literal
2. By new keyword

String Literal

Java String literal is created by using double quotes. For Example:

```
1. String s="welcome";
```

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

```
1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance
```

By new keyword

```
1. String s=new String("Welcome");//creates two objects and one reference variable
```

In such case, JVM will create a new string object in normal (non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap (non pool).

Java String Example

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

```
java
```

strings
example

Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

```
class Testimmutablestring{
  public static void main(String args[]){
    String s="Sachin";
    s.concat(" Tendulkar");//concat() method appends the string at the end
    System.out.println(s);//will print Sachin because strings are immutable objects
  } }
```

Output:Sachin

```
class Testimmutablestring1{
  public static void main(String args[]){
    String s="Sachin";
    s=s.concat(" Tendulkar");
    System.out.println(s);
  } } Output:Sachin Tendulkar
```

UNIT-II

Inheritance in Java

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object. Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

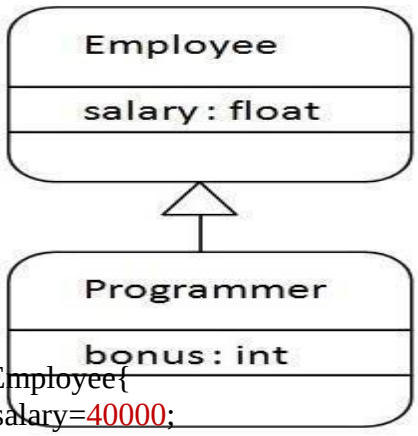
Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Syntax of Java Inheritance

```
1. class Subclass-name extends Superclass-name
2. {
3.     //methods and fields
4. }
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

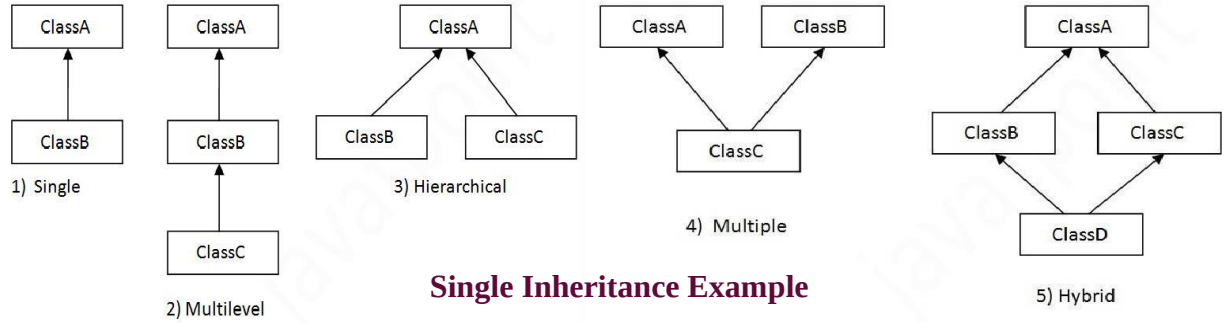



```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
} }
```

Programmer salary is:40000.0

Bonus of programmer is:10000

Types of inheritance in java



Single Inheritance Example

File: TestInheritance.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:
barking...
eating...

Multilevel Inheritance Example

File: TestInheritance2.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
```

```
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example

File: TestInheritance3.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

```
meowing...
eating...
```

Member access and Inheritance

A subclass includes all of the members of its super class but it cannot access those members of the super class that have been declared as private. Attempt to access a private variable would cause compilation error as it causes access violation. The variables declared as private, is only accessible by other members of its own class. Subclass have no access to it.

super keyword in java

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

super is used to refer immediate parent class instance variable.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
```

```
d.printColor();
}}
```

Output:

```
black
white
```

Final Keyword in Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

Object class in Java

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object. For example:

```
1. Object obj=getObject();//we don't know what object will be returned from this method
```

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

Example of method overriding

```
Class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
class Bike2 extends Vehicle{
void run(){System.out.println("Bike is running safely");}
public static void main(String args[]){
Bike2 obj = new Bike2();
obj.run();
}
```

Output:Bike is running safely

```
1. class Bank{
int getRateOfInterest(){return 0;}
}
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}
 class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
} }
```

```
Output:
SBI Rate of Interest: 8
```

```
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Abstract class in Java

A class that is declared with abstract keyword is known as abstract class in java. It can have abstract and non-abstract methods (method with body). It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
1. abstract class A{}
```

abstract method

```
1. abstract void printStatus();//no body and abstract
```

Example of abstract class that has abstract method

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely..");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
1. }
```

```
running safely..
```

Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

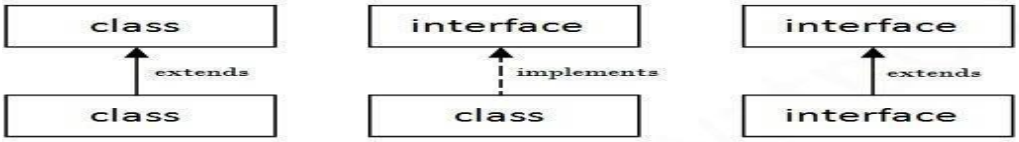
It cannot be instantiated just like abstract class.

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Internal addition by compiler

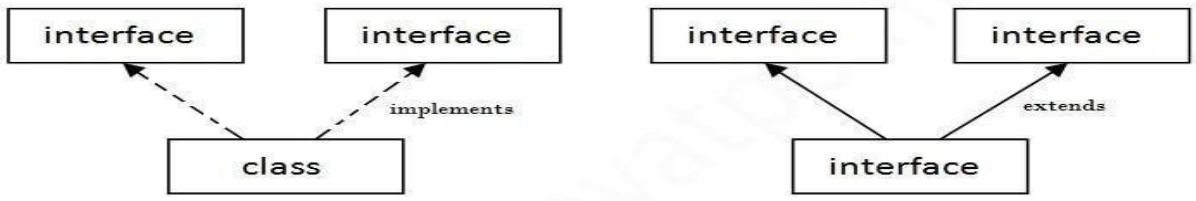
Understanding relationship between classes and interfaces




```
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

Output:drawing circle

Multiple inheritance in Java by interface



Multiple Inheritance in Java

```
interface Printable{
```

```
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}
public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 } }
```

Output:Hello
Welcome

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

6) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Java Inner Classes

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Syntax of Inner class

```
1. class Java_Outer_class{
2.  //code
3.  class Java_Inner_class{
4.   //code
5.  } }
```

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.

2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.

3) **Code Optimization**: It requires less code to write.

Difference between nested class and inner class in Java

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

- Non-static nested class (inner class)
    1. Member inner class
    2. Anonymous inner class
    3. Local inner class
- Static nested class

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

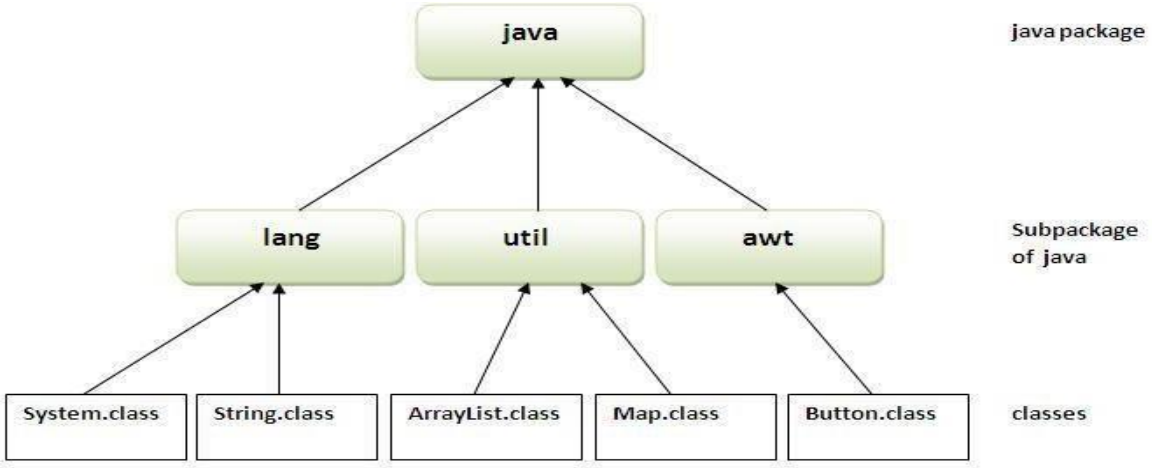
Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

```
package mypack;
public class Simple{
 public static void main(String args[]){
   System.out.println("Welcome to package");
  } }
```



How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

javac -d directory javafilename

How to run java package program

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Using fully qualified name

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");} }
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

Output:Hello

Exception Handling

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

What is exception

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling.

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

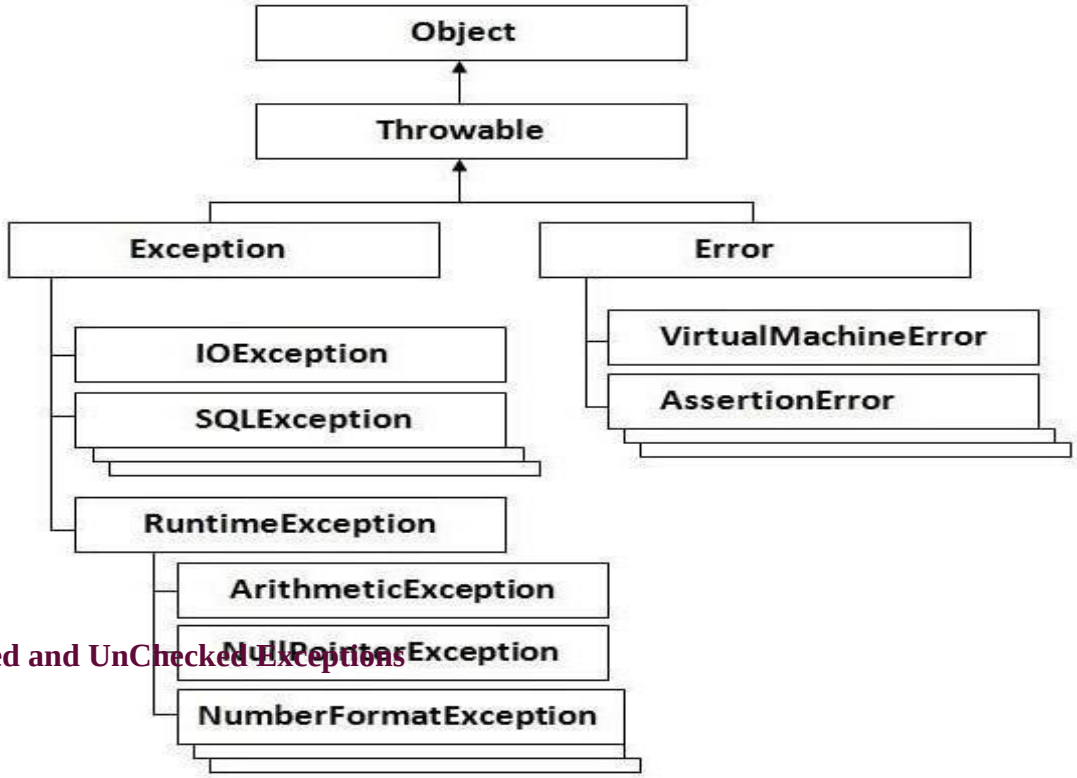
Difference between checked and unchecked exceptions

1) Checked Exception: The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception: The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error: Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionErroretc.

Hierarchy of Java Exception classes



Checked and UnChecked Exceptions

Checked Exceptions	Unchecked Exceptions
• Exception which are checked at Compile time called Checked Exception • If a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword	• Exceptions whose handling is NOT verified during Compile time. • These exceptions are handled at run-time i.e., by JVM after they occurred by using the try and catch block
• Examples: ○ IOException ○ SQLException ○ DataAccessException ○ ClassNotFoundException ○ InvocationTargetException ○ MalformedURLException	• Examples ○ NullPointerException ○ ArrayIndexOutOfBound ○ IllegalArgumentException ○ IllegalStateException

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Syntax of java try-catch

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
try{
//code that may throw exception
}finally{}
```

Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1{
  public static void main(String args[]){
    int data=50/0;//may throw exception
    System.out.println("rest of the code...");
} }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
```

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.

Solution by exception handling

Let's see the solution of above problem by java try-catch block.

```
public class Testtrycatch2{
```

```
 public static void main(String args[]){
  try{
    int data=50/0;
  }catch(ArithmeticException e){System.out.println(e);}
  System.out.println("rest of the code...");
} }
```

1. Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

Java Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

Let's see a simple example of java multi-catch block.

```
1.  public class TestMultipleCatchBlock{
2.    public static void main(String args[]){
3.     try{
4.      int a[]=new int[5];
5.      a[5]=30/0;
6.     }
7.    catch(ArithmeticException e){System.out.println("task1 is completed");}
8.   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");
9. }
10.    catch(Exception e){System.out.println("common task completed");
11. }
12.    System.out.println("rest of the code...");
13. } }
```

```
Output:task1 completed
      rest of the code...
```

Java nested try example

Let's see a simple example of java nested try block.

```
class Excep6{
 public static void main(String args[]){
  try{
    try{
     System.out.println("going to divide");
     int b =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
```

```
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}
    System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}
  System.out.println("normal flow..");
 }
1. }
```

Java finally block

Java finally block is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

Usage of Java finally

Case 1

Let's see the java finally example where **exception doesn't occur**.

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

```
Output:5
      finally block is always executed
      rest of the code...
```

Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
1. throw exception;
```

Java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
1. public class TestThrow1{
      static void validate(int age){
      if(age<18)
      throw new ArithmeticException("not valid");
      else
      System.out.println("welcome to vote");
      }
      public static void main(String args[]){
      validate(13);
      System.out.println("rest of the code...");
      }  }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:not valid
```

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

```
1.  return_type method_name() throws exception_class_name{
2.  //method code
3. }
4.
```

Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;
class Testthrows1{
  void m()throws IOException{
    throw new IOException("device error");//checked exception
```

```
    }
    void n()throws IOException{
      m();
    }
    void p(){
     try{
      n();
     }catch(Exception e){System.out.println("exception handled");}
    }
    public static void main(String args[]){
     Testthrows1 obj=new Testthrows1();
     obj.p();
     System.out.println("normal flow..."); } }
```

Output:

```
exception handled
normal flow...
```

Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Let's see a simple example of java custom exception.

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 } }
class TestCustomException1{
  static void validate(int age)throws InvalidAgeException{
    if(age<18)
     throw new InvalidAgeException("not valid");
    else
     System.out.println("welcome to vote");
  }
  public static void main(String args[]){
     try{
     validate(13);
     }catch(Exception m){System.out.println("Exception occured: "+m);}

     System.out.println("rest of the code...");
 } }
```

Output:Exception occured: InvalidAgeException:not valid rest of the code...

UNIT-III

Java - Files and I/O

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.

Stream

A stream can be defined as a sequence of data. There are two kinds of Streams −

- **InPutStream** − The InputStream is used to read data from a source.
- **OutPutStream** − The OutputStream is used for writing data to a destination.

Java provides strong but flexible support for I/O related to files and networks but this tutorial covers very basic functionality related to streams and I/O. We will see the most commonly used examples one by one −

Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**. Following is an example which makes use of these two classes to copy an input file into an output file −

Example

```
import java.io.*;
public class CopyFile {
   public static void main(String args[]) throws IOException {
      FileInputStream in = null;
      FileOutputStream out = null;
      try {
         in = new FileInputStream("input.txt");
         out = new FileOutputStream("output.txt");
         int c;
         while ((c = in.read()) != -1) {
```

```
        out.write(c);
      }
    }finally {
      if (in != null) {
        in.close();
      }
      if (out != null) {
        out.close();
      }}}
```

Now let's have a file **input.txt** with the following content −

```
This is test for copy file.
```

As a next step, compile the above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put the above code in CopyFile.java file and do the following −

```
$javac CopyFile.java
$java CopyFile
```

Character Streams

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write the above example, which makes the use of these two classes to copy an input file (having unicode characters) into an output file −

Example

```
import java.io.*;

public class CopyFile {

   public static void main(String args[]) throws IOException {
```

```
FileReader in = null;
FileWriter out = null;
try {
   in = new FileReader("input.txt");
   out = new FileWriter("output.txt");
   int c;
   while ((c = in.read()) != -1) {
      out.write(c);}
}finally {
   if (in != null) {
      in.close();}
   if (out != null) {
      out.close();
   }} }}
```

Now let's have a file **input.txt** with the following content −

```
This is test for copy file.
```

As a next step, compile the above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put the above code in CopyFile.java file and do the following −

```
$javac CopyFile.java
$java CopyFile
```

Standard Streams

All the programming languages provide support for standard I/O where the user's program can take input from a keyboard and then produce an output on the computer screen. Java provides the following three standard streams −

- **Standard Input** − This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as**System.in**.

- **Standard Output** − This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as **System.out**.
- **Standard Error** − This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as **System.err**.

Following is a simple program, which creates **InputStreamReader** to read standard input stream until the user types a "

Example

```
import java.io.*;

public class ReadConsole {
   public static void main(String args[]) throws IOException {
      InputStreamReader cin = null;
      try {
         cin = new InputStreamReader(System.in);
         System.out.println("Enter characters, 'q' to quit.");
         char c;
         do {
            c = (char) cin.read();
            System.out.print(c);
         } while(c != 'q');
      }finally {
if (cin != null) {
            cin.close();
         } } }}
```

This program continues to read and output the same character until we press 'q' −

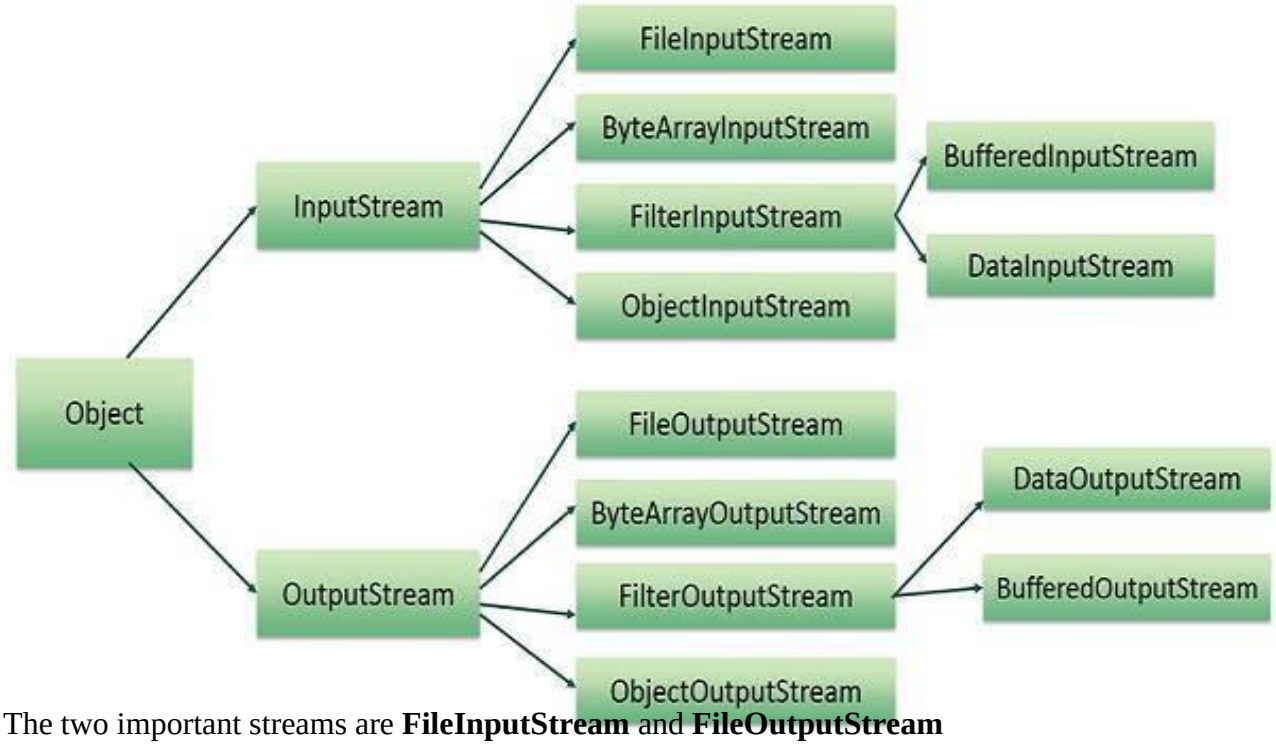
```
$javac ReadConsole.java
$java ReadConsole
```

```
Enter characters, 'q' to quit.
1
1
e
e
q
q
```

Reading and Writing Files

As described earlier, a stream can be defined as a sequence of data. The **InputStream** is used to read data from a source and the **OutputStream** is used for writing data to a destination.

Here is a hierarchy of classes to deal with Input and Output streams.



The two important streams are **FileInputStream** and **FileOutputStream**

FileInputStream

This stream is used for reading data from the files. Objects can be created using the keyword **new** and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file −

```
InputStream f = new FileInputStream("C:/java/hello");
```

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows −

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Once you have *InputStream* object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

- ByteArrayInputStream
- DataInputStream

FileOutputStream

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object.

Following constructor takes a file name as a string to create an input stream object to write the file −

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows −

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Once you have *OutputStream* object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

- ByteArrayOutputStream
- DataOutputStream

Example

Following is the example to demonstrate InputStream and OutputStream −

```
import java.io.*;

public class fileStreamTest {

  public static void main(String args[]) {

    try {
```

```
        byte bWrite [] = {11,21,3,40,5};
        OutputStream os = new FileOutputStream("test.txt");
        for(int x = 0; x < bWrite.length ; x++) {
           os.write( bWrite[x] ); // writes the bytes}
        os.close();
        InputStream is = new FileInputStream("test.txt");
        int size = is.available();
        for(int i = 0; i < size; i++) {
           System.out.print((char)is.read() + " "); }
        is.close();
     } catch (IOException e) {
        System.out.print("Exception");
     }       }}
```

Java.io.RandomAccessFile Class

The **Java.io.RandomAccessFile** class file behaves like a large array of bytes stored in the file system.Instances of this class support both reading and writing to a random access file.

Class declaration

Following is the declaration for **Java.io.RandomAccessFile** class −

```
public class RandomAccessFile
  extends Object
    implements DataOutput, DataInput, Closeable
```

Class constructors

S.N.	Constructor & Description
1	**RandomAccessFile(File file, String mode)** This creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.

2	**RandomAccessFile(File file, String mode)** This creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Methodsinherited

This class inherits methods from the following classes −

- Java.io.Object

Java.io.File Class in Java

The File class is Java's representation of a file or directory path name. Because file and directory names have different formats on different platforms, a simple string is not adequate to name them. The File class contains several methods for working with the path name, deleting and renaming files, creating new directories, listing the contents of a directory, and determining several common attributes of files and directories.

- It is an abstract representation of file and directory pathnames.
- A pathname, whether abstract or in string form can be either absolute or relative. The parent of an abstract pathname may be obtained by invoking the getParent() method of this class.
- First of all, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.
- Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change.

How to create a File Object?

A File object is created by passing in a String that represents the name of a file, or a String or another File object. For example,

```
File a = new File("/usr/local/bin/geeks");
```

defines an abstract file name for the geeks file in directory /usr/local/bin. This is an absolute abstract file name.

Program to check if a file or directory physically exist or not.

```
// In this program, we accepts a file or directory name from
// command line arguments. Then the program will check if
// that file or directory physically exist or not and
// it displays the property of that file or directory.
*import java.io.File;

// Displaying file property
class fileProperty
{
   public static void main(String[] args) {
```

```
        //accept file name or directory name through command line args
        String fname =args[0];

        //pass the filename or directory name to File object
        File f = new File(fname);

        //apply File class methods on File object
        System.out.println("File name :"+f.getName());
        System.out.println("Path: "+f.getPath());
        System.out.println("Absolute path:" +f.getAbsolutePath());
        System.out.println("Parent:"+f.getParent());
        System.out.println("Exists :"+f.exists());
        if(f.exists())
        {
           System.out.println("Is writeable:"+f.canWrite());
           System.out.println("Is readable"+f.canRead());
           System.out.println("Is a directory:"+f.isDirectory());
           System.out.println("File Size in bytes "+f.length());
        }
     }
}
```

Output:

```
File name :file.txt

Path: file.txt

Absolute path:C:\Users\akki\IdeaProjects\codewriting\src\file.txt

Parent:null

Exists :true

Is writeable:true

Is readabletrue

Is a directory:false

File Size in bytes 20
```

Connceting to DB

Multithreading in java is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.

2) You **can perform many operations together so it saves time**.

3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

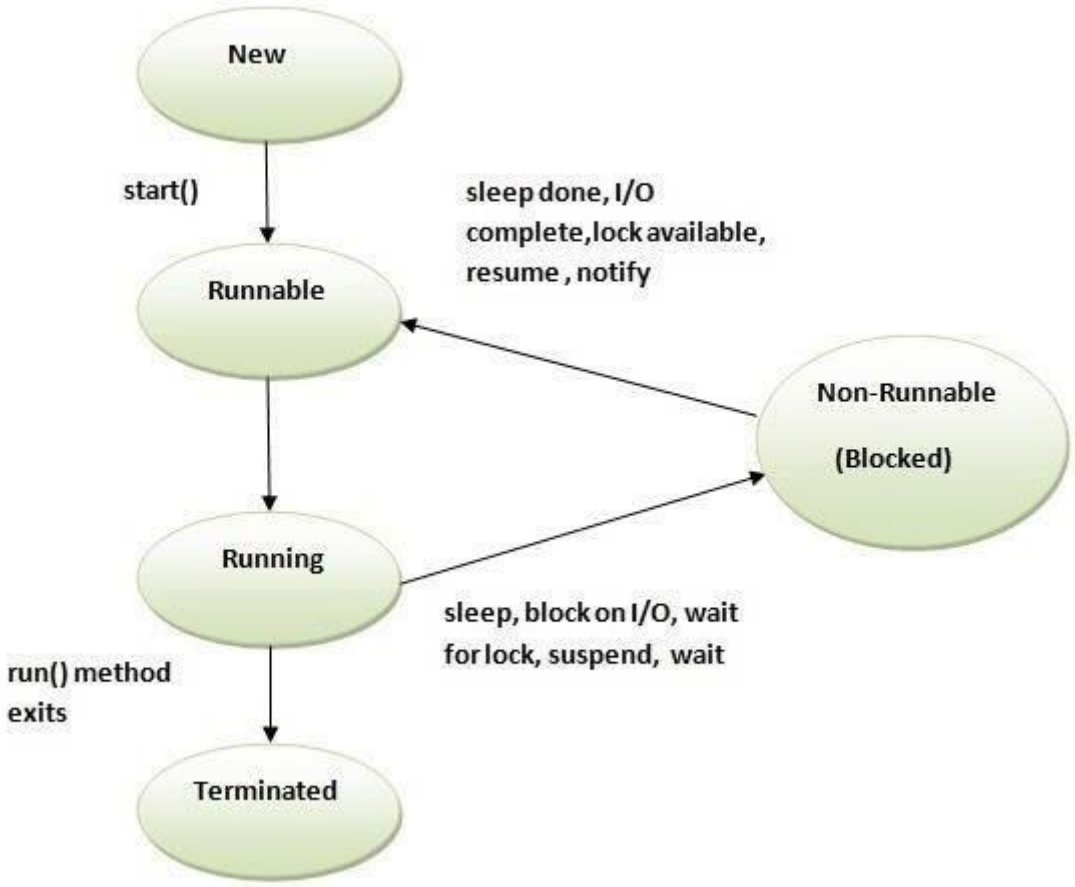
Life cycle of a Thread (Thread States)

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



How to create thread

There are two ways to create a thread:

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

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

oThread()
oThread(String name)
oThread(Runnable r)
oThread(Runnable r,String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Starting a thread:

start() method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

Java Thread Example by extending Thread class

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 } }
```

Output:thread is running...

Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 } }
```

Output:thread is running...

Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread{
public void run(){
  System.out.println("running thread name is:"+Thread.currentThread().getName());
  System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
 }
 public static void main(String args[]){
```

```
TestMultiPriority1 m1=new TestMultiPriority1();
TestMultiPriority1 m2=new TestMultiPriority1();
m1.setPriority(Thread.MIN_PRIORITY);
m2.setPriority(Thread.MAX_PRIORITY);
m1.start();
m2.start();
} }
```

Output:running thread name is:Thread-0
running thread priority is:10
running thread name is:Thread-1
running thread priority is:1

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Example of inter thread communication in java

Let's see the simple example of inter thread communication.

```
class Customer{
int amount=10000;
synchronized void withdraw(int amount){
System.out.println("going to withdraw...");
if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}
synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}
class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
```

```
public void run(){c.deposit(10000);}
}
start();
}}
```

```
Output: going to withdraw...
        Less balance; waiting for deposit...
        going to deposit...
        deposit completed...
        withdraw completed
```

ThreadGroup in Java

Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call.

> ***Note: Now suspend(), resume() and stop() methods are deprecated.***

Java thread group is implemented by *java.lang.ThreadGroup* class.

Constructors of ThreadGroup class

There are only two constructors of ThreadGroup class.

ThreadGroup(String name)
ThreadGroup(ThreadGroup parent, String name)

Let's see a code to group multiple threads.

```
1. ThreadGroup tg1 = new ThreadGroup("Group A");
2. Thread t1 = new Thread(tg1,new MyRunnable(),"one");
3. Thread t2 = new Thread(tg1,new MyRunnable(),"two");
4. Thread t3 = new Thread(tg1,new MyRunnable(),"three");
```

Now all 3 threads belong to one group. Here, tg1 is the thread group name, MyRunnable is the class that implements Runnable interface and "one", "two" and "three" are the thread names.

Now we can interrupt all threads by a single line of code only.

```
1. Thread.currentThread().getThreadGroup().interrupt();
```

java.net

The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols −

- **TCP** − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** − UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

This chapter gives a good understanding on the following two subjects −

- **Socket Programming** − This is the most widely used concept in Networking and it has been explained in very detail.
- **URL Processing** − This would be covered separately.

java.text

The java.text package is necessary for every java developer to master because it has a lot of classes that is helpful in formatting such as dates, numbers, and messages.

java.text Classes

The following are the classes available for java.text package

[table]

Class|Description

SimpleDateFormat|is a concrete class that helps in formatting and parsing of dates.

[/table]

Collection Framework in Java

Collections in java is a framework that provides an architecture to store and manipulate the group of objects. All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

What is framework in java

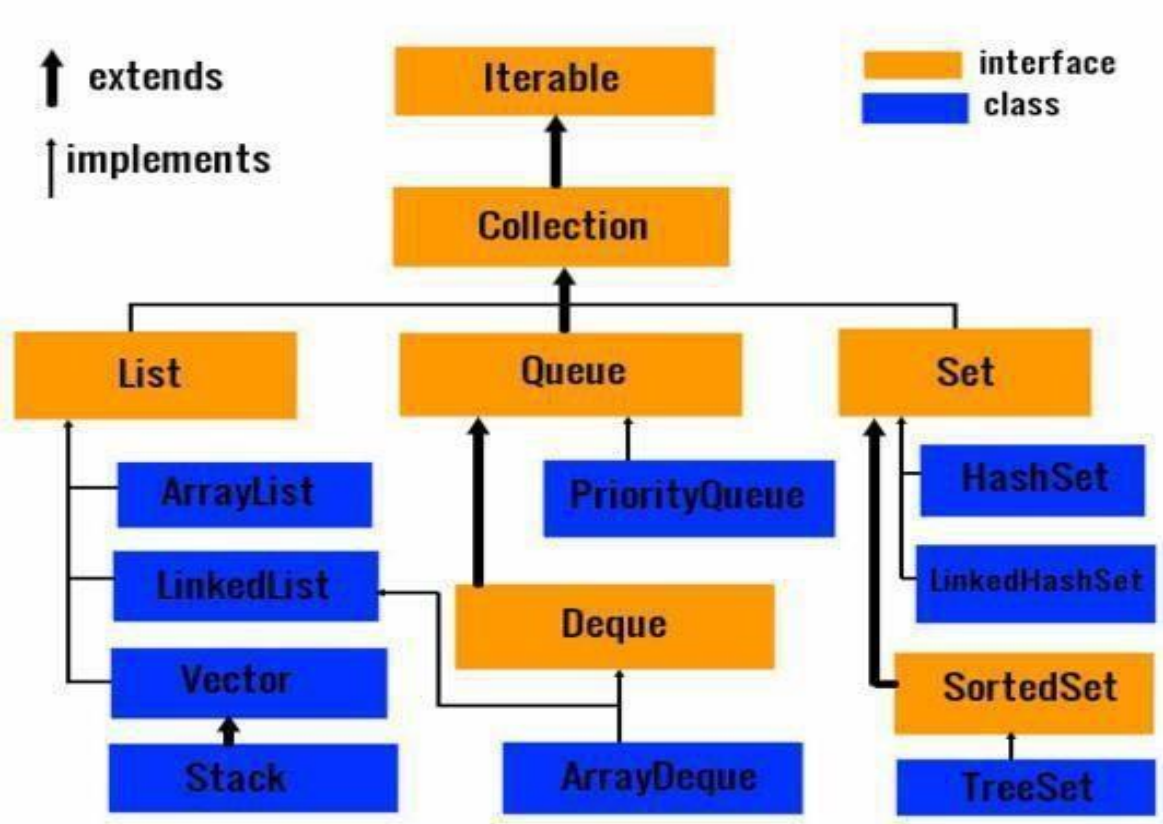
- ✓ provides readymade architecture.
- ✓ represents set of classes and interface.
- ✓ is optional.

What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm

Hierarchy of Collection Framework


extends
implements
Iterable
interface
class
Collection
List
Queue
Set
ArrayList
LinkedList
Vector
Stack
PriorityQueue
Deque
ArrayDeque
HashSet
LinkedHashSet
SortedSet
TreeSet

Java List Interface

List Interface is the sub interface of Collection. It contains index-based methods to insert and delete elements. It is a factory of ListIterator interface.

List Interface declaration

1. **public interface** List<E> **extends** Collection<E>

Methods of Java List Interface

Method	Description
void add(int index, E element)	It is used to insert the specified element at the specified position in a list.
boolean add(E e)	It is used to append the specified element at the end of a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of a list.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void clear()	It is used to remove all of the elements from this list.
boolean equals(Object o)	It is used to compare the specified object with the elements of a list.
int hashcode()	It is used to return the hash code value for a list.
E get(int index)	It is used to fetch the element from the particular position of the list.
boolean isEmpty()	It returns true if the list is empty, otherwise false.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
T[] toArray(T[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean contains(Object o)	It returns true if the list contains the specified element
boolean containsAll(Collection<?> c)	It returns true if the list contains all the specified element
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

E remove(int index)	It is used to remove the element present at the specified position in the list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element.
boolean removeAll(Collection<?> c)	It is used to remove all the elements from the list.
void replaceAll(UnaryOperator operator)	It is used to replace all the elements from the list with the specified element.
void retainAll(Collection<?> c)	It is used to retain all the elements in the list that are present in the specified collection.
E set(int index, E element)	It is used to replace the specified element in the list, present at the specified position.
void sort(Comparator<? super E> c)	It is used to sort the elements of the list on the basis of specified comparator.
Spliterator spliterator()	It is used to create spliterator over the elements in a list.
List<E> subList(int fromIndex, int toIndex)	It is used to fetch all the elements lies within the given range.
int size()	It is used to return the number of elements present in the list.

Java List Example

```
1. import java.util.*;
2. public class ListExample{
3. public static void main(String args[]){
4. List<String> al=new ArrayList<String>();
5. al.add("Amit");
6. al.add("Vijay");
7. al.add("Kumar");
8. al.add(1,"Sachin");
9. System.out.println("An element at 2nd position: "+al.get(2));
10. for(String s:al){
11.  System.out.println(s);
12. }
13. }
14. }
```

Output:

```
An element at 2nd position: Vijay
Amit
Sachi
n
Vijay
```

Java ListIterator Interface

ListIterator Interface is used to traverse the element in a backward and forward

direction. ListIterator Interface declaration

1. **public interface** ListIterator<E> **extends**

   Iterator<E> Methods of Java ListIterator Interface:

Method	Description
void add(E e)	This method inserts the specified element into the list.
boolean hasNext()	This method returns true if the list iterator has more elements while traversing the list in the forward direction.
E next()	This method returns the next element in the list and advances the cursor position.
int nextIndex()	This method returns the index of the element that would be returned by a subsequent call to next()
boolean hasPrevious()	This method returns true if this list iterator has more elements while traversing the list in the reverse direction.
E previous()	This method returns the previous element in the list and moves the cursor position backward.
E previousIndex()	This method returns the index of the element that would be returned by a subsequent call to previous().
void remove()	This method removes the last element from the list that was returned by next() or previous() methods
void set(E e)	This method replaces the last element returned by next() or previous() methods with the specified element.

Example of ListIterator Interface

```
1.  import java.util.*;
2.  public class ListIteratorExample1{
3.  public static void main(String args[]){
4.  List<String> al=new ArrayList<String>();
5.          al.add("Amit");
6.          al.add("Vijay");
7.          al.add("Kumar");
8.          al.add(1,"Sachin");
9.          ListIterator<String> itr=al.listIterator();
10.         System.out.println("Traversing elements in forward direction");
11.         while(itr.hasNext())
12.        {
13.
14.         System.out.println("index:"+itr.nextIndex()+" value:"+itr.next());
15.         }
16.         System.out.println("Traversing elements in backward direction");
17.         while(itr.hasPrevious())
18.         {
```

```
19.
20.         System.out.println("index:"+itr.previousIndex()+" value:"+itr.previous());
21.         }
22. }
23. }
```

Output:

```
Traversing elements in forward
direction index:0 value:Amit
index:1
value:Sachin
index:2
value:Vijay
index:3
value:Kumar
Traversing elements in backward
direction index:3 value:Kumar
```

Example of ListIterator Interface: Book

```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
7.     this.id = id;
8.     this.name = name;
9.     this.author = author;
10.    this.publisher = publisher;
11.    this.quantity = quantity;
12. }
13. }
14. public class ListIteratorExample2 {
15. public static void main(String[] args) {
16.    //Creating list of Books
17.    List<Book> list=new ArrayList<Book>();
18.    //Creating Books
19.    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
20.    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw
       Hill",4);
21.    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
22.    //Adding Books to list
23.    list.add(b1);
24.    list.add(b2);
25.    list.add(b3);
26.    //Traversing list
27.    for(Book b:list){
28.    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
29.    }
30. }
31. }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw
Hill4 103 Operating System Galvin Wiley 6
```

Java ArrayList class

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

- ✓ Java ArrayList class can contain duplicate elements.
- ✓ Java ArrayList class maintains insertion order.
- ✓ Java ArrayList class is non synchronized.
- ✓ Java ArrayList allows random access because array works at the index basis.
- ✓ In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

ArrayList class declaration

Let's see the declaration for java.util.ArrayList class.

Constructors of Java ArrayList Constructor Description	
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Java ArrayList Example

```
import java.util.*;
class TestCollection1
{
public static void main(String args[])
{
ArrayList<String> list=new ArrayList<String>();//Creating arraylist
list.add("Ravi");//Adding object in arraylist
list.add("Vijay");
list.add("Ravi");
list.add("Ajay");
//Traversing list through Iterator
Iterator itr=list.iterator();
while(itr.hasNext())
{
System.out.println(itr.next());
}
}
}
```

Ouput: Ravi
Vijay Ravi
Ajay

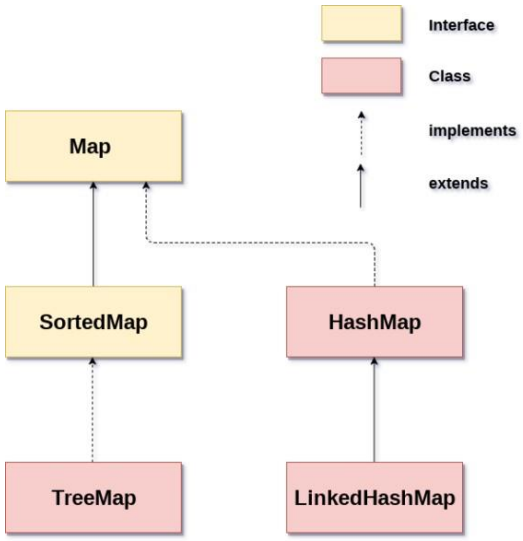
Java Map Interface

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

Java Map Hierarchy

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap. The hierarchy of Java Map is given below:



A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

A Map can't be traversed, so you need to convert it into Set using *keySet()* or *entrySet()* method.

Class	Description
HashMap	HashMap is the implementation of Map, but it doesn't maintain any order.
LinkedHashMap	LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order.
TreeMap	TreeMap is the implementation of Map and SortedMap. It maintains ascending order.

Java Map Example: Non-Generic (Old Style)

```
1. //Non-generic
2. import java.util.*;
3. public class MapExample1 {
4. public static void main(String[] args) {
5.     Map map=new HashMap();
```

```
6.  //Adding elements to map
7.  map.put(1,"Amit");
8.  map.put(5,"Rahul");
9.  map.put(2,"Jai");
10. map.put(6,"Amit");
```

```
    //Traversing Map
    Set set=map.entrySet();//Converting to Set so that we can traverse
    Iterator itr=set.iterator();
    while(itr.hasNext()){
        //Converting to Map.Entry so that we can get key and value separately
        Map.Entry entry=(Map.Entry)itr.next();
        System.out.println(entry.getKey()+" "+entry.getValue());
    }
}
}
```

Output:

```
1 Amit
2 Jai
5 Rahul
6 Amit
```

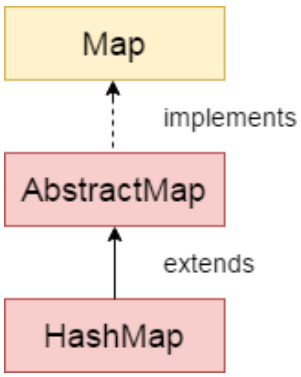
Java Map Example: Generic (New Style)

```
import java.util.*;
class
MapExample2 {
  public static void main(String
args[]) {
    Map<Integer,String> map=new HashMap<Integer,String>();
    map.put(100,"Amit");
    map.put(101,"Vijay");
    map.put(102,"Rahul");
    //Elements can traverse in any order
    for(Map.Entry m:map.entrySet()){
     System.out.println(m.getKey()+" "+m.getValue());
    }
  }
}
```

Output:

```
102 Rahul
100 Amit
101 Vijay
```

Java HashMap class



Java HashMap class implements the map interface by using a hash table. It inherits AbstractMap class and implements Map interface.

Points to remember

- Java HashMap class contains values based on the key.
- Java HashMap class contains only unique keys.
- Java HashMap class may have one null key and multiple null values.
- Java HashMap class is non synchronized.
- Java HashMap class maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Hierarchy of HashMap class

As shown in the above figure, HashMap class extends AbstractMap class and implements Map

interface. HashMap class declaration

Let's see the declaration for java.util.HashMap class.

1. **public class** HashMap<K,V> **extends** AbstractMap<K,V> **implements** Map<K,V>, Cloneable, Serializ able

HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Constructors of Java HashMap class

Constructor	Description
HashMap()	It is used to construct a default HashMap.

HashMap(Map<? extends K,? extends V> m)	It is used to initialize the hash map by using the elements of the given Map object m.

HashMap(int capacity)	It is used to initializes the capacity of the hash map to the given integer value, capacity.
HashMap(int capacity, float loadFactor)	It is used to initialize both the capacity and load factor of the hash map by using its arguments.

Methods of Java HashMap class

Method	Description
void clear()	It is used to remove all of the mappings from this map.
boolean isEmpty()	It is used to return true if this map contains no key-value mappings.
Object clone()	It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
Set entrySet()	It is used to return a collection view of the mappings contained in this map.
Set keySet()	It is used to return a set view of the keys contained in this map.
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null.

V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V get(Object key)	This method returns the object that contains the value associated with the key.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
boolean isEmpty()	This method returns true if the map is empty; returns false if it contains at least one key.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection<V> values()	It returns a collection view of the values contained in the map.
int size()	This method returns the number of entries in the map.

Java HashMap example to add() elements

Here, we see different ways to insert elements.

```
import java.util.*;

    class HashMap1{
     public static void main(String args[]){
       HashMap<Integer,String> hm=new
        HashMap<Integer,String>(); System.out.println("Initial list
        of elements: "+hm); hm.put(100,"Amit");
         hm.put(101,"Vijay");
         hm.put(102,"Rahul");

         System.out.println("After invoking put()
         method "); for(Map.Entry m:hm.entrySet()){
         System.out.println(m.getKey()+"
         "+m.getValue());
         }

         hm.putIfAbsent(103, "Gaurav");
         System.out.println("After invoking putIfAbsent() method ");
         for(Map.Entry m:hm.entrySet()){
             System.out.println(m.getKey()+" "+m.getValue());
            }
         HashMap<Integer,String> map=new HashMap<Integer,String>();
         map.put(104,"Ravi");
         map.putAll(hm);
         System.out.println("After invoking putAll() method ");
         for(Map.Entry m:map.entrySet()){
             System.out.println(m.getKey()+" "+m.getValue());
            }
     }
}
```

```
Initial list of elements: {}
After invoking put()
method 100 Amit
101 Vijay
102 Rahul

After invoking putIfAbsent()

method 100 Amit
101 Vijay
102 Rahul
103 Gaurav

After invoking putAll() method

100 Amit
101 Vijay
102 Rahul
103 Gaurav
104 Ravi
```

Java HashMap example to remove() elements

Here, we see different ways to remove elements.

```
import java.util.*;
public class HashMap2
{
  public static void main(String args[])
   {
   HashMap<Integer,String> map=new
    HashMap<Integer,String>(); map.put(100,"Amit");
    map.put(101,"Vijay");
    map.put(102,"Rahul");
    map.put(103, "Gaurav");
   System.out.println("Initial list of elements:
   "+map);
   //key-based removal
   map.remove(100);
   System.out.println("Updated list of elements: "+map);
   //value-based removal
   map.remove(101);
   System.out.println("Updated list of elements: "+map);
   //key-value pair based removal
   map.remove(102, "Rahul");
   System.out.println("Updated list of elements: "+map);
  }
}
```

Output:

```
Initial list of elements: {100=Amit, 101=Vijay, 102=Rahul,
103=Gaurav} Updated list of elements: {101=Vijay, 102=Rahul,
103=Gaurav} Updated list of elements: {102=Rahul, 103=Gaurav}
Updated list of elements: {103=Gaurav}
```

Java HashMap example to replace() elements

Here, we see different ways to replace elements.

```
import java.util.*;
class HashMap3{
 public static void main(String args[]){
  HashMap<Integer,String> hm=new
    HashMap<Integer,String>(); hm.put(100,"Amit");
    hm.put(101,"Vijay");
    hm.put(102,"Rahul");
   System.out.println("Initial list of
   elements:"); for(Map.Entry
```

```
m:hm.entrySet())
{
  System.out.println(m.getKey()+" "+m.getValue());
}
System.out.println("Updated list of elements:");
hm.replace(102, "Gaurav");
for(Map.Entry m:hm.entrySet())
{
  System.out.println(m.getKey()+" "+m.getValue());
}
System.out.println("Updated list of
elements:"); hm.replace(101, "Vijay",
"Ravi"); for(Map.Entry m:hm.entrySet())
{
  System.out.println(m.getKey()+" "+m.getValue());
}
System.out.println("Updated list of elements:");
hm.replaceAll((k,v) -> "Ajay");
```

```
        for(Map.Entry m:hm.entrySet())
        {
          System.out.println(m.getKey()+" "+m.getValue());
        }
      }
}
```

```
Initial list of elements:
      100 Amit
      101 Vijay
      102 Rahul
Updated list of elements:
      100 Amit
      101 Vijay
      102 Gaurav
Updated list of elements:
      100 Amit
      101 Ravi
      102 Gaurav
Updated list of elements:
      100 Ajay
      101 Ajay
      102 Ajay
```

Difference between HashSet and HashMap

HashSet contains only values whereas HashMap contains an entry(key and value).

Java HashMap Example: Book

```
      import java.util.*;
      class Book {
      int id;
String name,author,publisher;
      int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
        this.id = id;
        this.name = name;
        this.author =
        author;
        this.publisher = publisher;
        this.quantity = quantity;
}
}
      public class MapExample {
      public static void main(String[] args) {
        //Creating map of Books
        Map<Integer,Book> map=new HashMap<Integer,Book>();
        //Creating Books
        Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
        Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw
        Hill",4); Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
        //Adding Books to
        map map.put(1,b1);
        map.put(2,b2);
        map.put(3,b3);
```

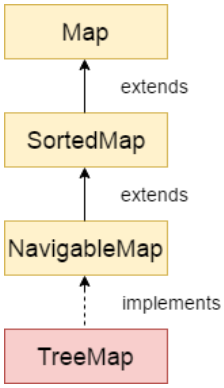
```
//Traversing map
for(Map.Entry<Integer, Book> entry:map.entrySet()){
   int key=entry.getKey();
```

```
            Book b=entry.getValue();
            System.out.println(key+" Details:");
            System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
        }
}
}
```

Output:

```
1 Details:
101 Let us C Yashwant Kanetkar
BPB 8 2 Details:
102 Data Communications & Networking Forouzan Mc Graw
Hill 4 3 Details:
103 Operating System Galvin
Wiley 6
```

Java TreeMap class



Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

- Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- Java TreeMap contains only unique elements.
- Java TreeMap cannot have a null key but can have multiple null values.
- Java TreeMap is non synchronized.
- Java TreeMap maintains ascending order.

TreeMap class declaration

Let's see the declaration for java.util.TreeMap class.

1. **public class** TreeMap<K,V> **extends** AbstractMap<K,V> **implements** NavigableMap<K,V>, Cloneabl e, Serializable

TreeMap class Parameters

Let's see the Parameters for java.util.TreeMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

Constructors of Java TreeMap class

Constructor	Description
TreeMap()	It is used to construct an empty tree map that will be sorted using the natural order of its key.
TreeMap(Comparator<? super K> comparator)	It is used to construct an empty tree-based map that will be sorted using the comparator comp.
TreeMap(Map<? extends K,? extends V> m)	It is used to initialize a treemap with the entries from **m**, which will be sorted using the natural order of the keys.
TreeMap(SortedMap<K,? extends V> m)	It is used to initialize atreemap with the entries from the SortedMap **sm**, which will be sorted in the same order as **sm.**

Methods of Java TreeMap class

Method	Description
Map.Entry<K,V> ceilingEntry(K key)	It returns the key-value pair having the least key, greater than or equal to the specified key, or null if there is no such key.
K ceilingKey(K key)	It returns the least key, greater than the specified key or null if there is no such key.
void clear()	It removes all the key-value pairs from a map.
Object clone()	It returns a shallow copy of TreeMap instance.
Comparator<? super K> comparator()	It returns the comparator that arranges the key in order, or null if the map uses the natural ordering.
NavigableSet<K> descendingKeySet()	It returns a reverse order NavigableSet view of the keys contained in the map.
NavigableMap<K,V> descendingMap()	It returns the specified key-value pairs in descending order.
Map.Entry firstEntry()	It returns the key-value pair having the least key.
Map.Entry<K,V> floorEntry(K key)	It returns the greatest key, less than or equal to the specified key, or null if there is no such key.

void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
SortedMap<K,V> headMap(K toKey)	It returns the key-value pairs whose keys are strictly less than toKey.
NavigableMap<K,V> headMap(K toKey, boolean inclusive)	It returns the key-value pairs whose keys are less than (or equal to if inclusive is true) toKey.
Map.Entry<K,V> higherEntry(K key)	It returns the least key strictly greater than the given key, or null if there is no such key.
K higherKey(K key)	It is used to return true if this map contains a mapping for the specified key.
Set keySet()	It returns the collection of keys exist in the map.
Map.Entry<K,V> lastEntry()	It returns the key-value pair having the greatest key, or null if there is no such key.
Map.Entry<K,V> lowerEntry(K key)	It returns a key-value mapping associated with the greatest key strictly less than the given key, or null if there is no such key.
K lowerKey(K key)	It returns the greatest key strictly less than the given key, or null if there is no such key.
NavigableSet<K> navigableKeySet()	It returns a NavigableSet view of the keys contained in this map.
Map.Entry<K,V> pollFirstEntry()	It removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty.
Map.Entry<K,V> pollLastEntry()	It removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.
V put(K key, V value)	It inserts the specified value with the specified key in the map.
void putAll(Map<? extends K,? extends V> map)	It is used to copy all the key-value pair from one map to another map.

V replace(K key, V value)	It replaces the specified value for a specified key.

boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
NavigableMap<K,V> subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)	It returns key-value pairs whose keys range from fromKey to toKey.
SortedMap<K,V> subMap(K fromKey, K toKey)	It returns key-value pairs whose keys range from fromKey, inclusive, to toKey, exclusive.
SortedMap<K,V> tailMap(K fromKey)	It returns key-value pairs whose keys are greater than or equal to fromKey.
NavigableMap<K,V> tailMap(K fromKey, boolean inclusive)	It returns key-value pairs whose keys are greater than (or equal to, if inclusive is true) fromKey.
boolean containsKey(Object key)	It returns true if the map contains a mapping for the specified key.
boolean containsValue(Object value)	It returns true if the map maps one or more keys to the specified value.
K firstKey()	It is used to return the first (lowest) key currently in this sorted map.
V get(Object key)	It is used to return the value to which the map maps the specified key.
K lastKey()	It is used to return the last (highest) key currently in the sorted map.
V remove(Object key)	It removes the key-value pair of the specified key from the map.
Set<Map.Entry<K,V>> entrySet()	It returns a set view of the mappings contained in the map.
int size()	It returns the number of key-value pairs exists in the hashtable.

Collection values()	It returns a collection view of the values contained in the map.

Java TreeMap Example

```
import java.util.*; class
TreeMap1{
 public static void main(String args[]){
   TreeMap<Integer,String> map=newTreeMap<Integer,String>(); map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");


      for(Map.Entry m:map.entrySet()){ System.out.println(m.getKey()+"
       "+m.getValue());
      }
 }
}
```

```
Output:100 Amit
       101 Vijay
       102 Ravi
       103 Rahul
```

Java TreeMap Example: remove()

```
import java.util.*;
public class TreeMap2 {
   public static void main(String args[]) { TreeMap<Integer,String> map=new
    TreeMap<Integer,String>();
      map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");
      System.out.println("Before invoking remove() method");
      for(Map.Entry m:map.entrySet())
      {
          System.out.println(m.getKey()+" "+m.getValue());
      }
      map.remove(102);
      System.out.println("After invoking remove() method");
      for(Map.Entry m:map.entrySet())
      {
          System.out.println(m.getKey()+" "+m.getValue());
      }
      }
}
```

Output:

```
Before invoking remove() method
100 Amit
101 Vijay
102 Ravi
103 Rahul
After invoking remove() method

100 Amit
101 Vijay
103 Rahul
```

Java TreeMap Example: NavigableMap

```
import java.util.*;
class TreeMap3{
 public static void main(String args[]){
   NavigableMap<Integer,String> map=newTreeMap<Integer,String>(); map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");
      //Maintains descending order System.out.println("descendingMap:
      "+map.descendingMap());
      //Returns key-
value pairs whose keys are less than or equal to the specified key.
      System.out.println("headMap: "+map.headMap(102,true));
      //Returns key-
value pairs whose keys are greater than or equal to the specified key.
      System.out.println("tailMap: "+map.tailMap(102,true));
      //Returns key-value pairs exists in between the specified key. System.out.println("subMap:
      "+map.subMap(100, false, 102, true));
 }
}
```

```
descendingMap: {103=Rahul, 102=Ravi, 101=Vijay, 100=Amit}
headMap: {100=Amit, 101=Vijay, 102=Ravi}
tailMap: {102=Ravi, 103=Rahul}
subMap: {101=Vijay, 102=Ravi}
```

Java TreeMap Example: SortedMap

```
import java.util.*;
class TreeMap4{
 public static void main(String args[]){
   SortedMap<Integer,String> map=newTreeMap<Integer,String>(); map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");
      //Returns key-value pairs whose keys are less than the specified key.
      System.out.println("headMap: "+map.headMap(102));
      //Returns key-
value pairs whose keys are greater than or equal to the specified key.
      System.out.println("tailMap: "+map.tailMap(102));
      //Returns key-value pairs exists in between the specified key. System.out.println("subMap:
```

```
            "+map.subMap(100, 102));
        }
}
```

```
headMap: {100=Amit, 101=Vijay}
    tailMap: {102=Ravi, 103=Rahul}
    subMap: {100=Amit, 101=Vijay}
```

What is difference between HashMap and TreeMap?

HashMap	TreeMap
1) HashMap can contain one null key.	TreeMap cannot contain any null key.
2) HashMap maintains no order.	TreeMap maintains ascending order.

Java TreeMap Example: Book

```
    import java.util.*;
    class Book {
    int id;
String name,author,publisher;
    int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
        this.id = id; this.name =
        name; this.author =
        author;
        this.publisher = publisher;
        this.quantity = quantity;
}
}
    public class MapExample {
    public static void main(String[] args) {
        //Creating map of Books
        Map<Integer,Book> map=new TreeMap<Integer,Book>();
        //Creating Books
        Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
   Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
        Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
        //Adding Books to map
        map.put(2,b2);
        map.put(1,b1);
        map.put(3,b3);

        //Traversing map
        for(Map.Entry<Integer, Book> entry:map.entrySet()){
            int key=entry.getKey(); Book
            b=entry.getValue();
            System.out.println(key+" Details:");
       System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+""+b.q uantity);
        }
}
}
```

Output:

```
1 Details:
101 Let us C Yashwant Kanetkar BPB 8
2 Details:
102 Data Communications & Networking Forouzan Mc Graw Hill 4
```

```
3 Details:
103 Operating System Galvin Wiley 6
```

Set in Java

- Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.
- Basically, Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).
- Set has various methods to add, remove clear, size, etc to enhance the usage of this interface filter_none

edit play_arrow
brightness_4

```
// Java code for adding elements in Set
import java.util.*;
public class Set_example
{
        public static void main(String[] args)
        {
           // Set deonstration using HashSet
           Set<String> hash_Set = new
           HashSet<String>(); hash_Set.add("Geeks");
           hash_Set.add("For");
           hash_Set.add("Geeks");
           hash_Set.add("Example"
           ); hash_Set.add("Set");
           System.out.print("Set output without the duplicates");

           System.out.println(hash_Set);

           // Set deonstration using TreeSet
           System.out.print("Sorted Set after passing into
           TreeSet"); Set<String> tree_Set = new
           TreeSet<String>(hash_Set);
           System.out.println(tree_Set);
        }
}
```

(Please note that we have entered a duplicate entity but it is not displayed in the output. Also, we can directly sort the entries by passing the unordered Set in as the parameter of TreeSet).

Output:

```
Set output without the duplicates[Set, Example, Geeks, for]
Sorted Set after passing into TreeSet[Example, For, Geeks, Set]
```

TreeSet in Java

TreeSet is one of the most important implementations of the SortedSet interface in Java that uses a Tree for storage. The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided. This must be consistent with equals if it is to correctly implement the Set interface. It can also be ordered by a Comparator provided at set creation time, depending on which constructor is used. The TreeSet implements a NavigableSet interface by inheriting AbstractSet class.

Few important features of TreeSet are as follows:

1. TreeSet implements the SortedSet interface so duplicate values are not allowed.
2. Objects in a TreeSet are stored in a sorted and ascending order.

3. TreeSet does not preserve the insertion order of elements but elements are sorted by keys.
4. TreeSet does not allow to insert Heterogeneous objects. It will throw classCastException at Runtime if trying to add hetrogeneous objects.
5. TreeSet serves as an excellent choice for storing large amounts of sorted information which are supposed to be accessed quickly because of its faster access and retrieval time.
6. TreeSet is basically implementation of a self-balancing binary search tree like Red-Black Tree. Therefore operations like add, remove and search take O(Log n) time. And operations like printing n elements in sorted order takes O(n) time.

Constructors of TreeSet class:

1. **TreeSet t = new TreeSet();**
   This will create empty TreeSet object in which elements will get stored in default natural sorting order.
2. **TreeSet t = new TreeSet(Comparator comp);**
   This constructor is used when external specification of sorting order of elements is needed.
3. **TreeSet t = new TreeSet(Collection col);**
   This constructor is used when any conversion is needed from any Collection object to TreeSet object.
4. **TreeSet t = new TreeSet(SortedSet s)**;
   This constructor is used to convert SortedSet object to TreeSet Object.

Synchronized TreeSet:

The implementation in a TreeSet is not synchronized in a sense that if multiple threads access a tree set concurrently, and at least one of the threads modifies the set, it must be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the set. If no such object exists, the set should be “wrapped” using the Collections.synchronizedSortedSet method. This is best done at creation time, to prevent accidental unsynchronized access to the set:

```
TreeSet ts = new TreeSet();
Set syncSet = Collections.synchronziedSet(ts);
```

Below program illustrates the basic opearation of a TreeSet:

```
// Java program to demonstrate insertions in TreeSet
import java.util.*;

class TreeSetDemo {
        public static void main(String[] args)
        {
            TreeSet<String> ts1 = new TreeSet<String>();

            // Elements are added using add()
            method ts1.add("A");
            ts1.add("B");
            ts1.add("C");

            // Duplicates will not get
            insert ts1.add("C");
```

```
        // Elements get stored in default natural
        // Sorting Order(Ascending)
        System.out.println(ts1);
    }
}
```

Output:

```
[A, B, C]
```

Two things must be kept in mind while creating and adding elements into a TreeSet:

- Firstly, insertion of null into a TreeSet throws *NullPointerException* because while insertion of null, it gets compared to the existing elements and null cannot be compared to any value.
- Secondly, if we are depending on default natural sorting order, compulsory the object should be **homogeneous** and **comparable** otherwise we will get **RuntimeException:***ClassCastException*

```
// Java code to illustrate StringBuffer
// class does not implements
// Comparable interface.
import java.util.*;
class TreeSetDemo {

   public static void main(String[] args)
   {
      TreeSet<StringBuffer> ts = new TreeSet<StringBuffer>();

      // Elements are added using add()
      method ts.add(new
      StringBuffer("A")); ts.add(new
      StringBuffer("Z")); ts.add(new
      StringBuffer("L")); ts.add(new
      StringBuffer("B")); ts.add(new
      StringBuffer("O"));

      // We will get RunTimeException :ClassCastException
      // As StringBuffer does not implements Comparable interface
      System.out.println(ts);
   }
}
```

.

NOTE:

1. An object is said to be comparable if and only if the corresponding class implements **Comparable interface**.
2. **String** class and all **Wrapper** classes already implements Comparable interface but StringBuffer class doesn't implements Comparable interface.Hence we got *ClassCastException* in the above example.
3. For an empty tree-set, when trying to insert null as first value, one will get NPE from JDK 7.From
   1.7 onwards null is not at all accepted by TreeSet. However upto JDK 6, null will be

accepted as first value, but any if insertion of any more values in the TreeSet, will also throw NullPointerException.
Hence it was considered as bug and thus removed in JDK 7.

Methods of TreeSet class:

TreeSet implements SortedSet so it has availability of all methods in Collection, Set and SortedSet interfaces. Following are the methods in Treeset interface.

1. void add(Object o): This method will add specified element according to some sorting order in TreeSet. Duplicate entires will not get added.
2. boolean addAll(Collection c): This method will add all elements of specified Collection to the set. Elements in Collection should be homogeneous otherwise ClassCastException will be thrown. Duplicate Entries of Collection will not be added to TreeSet.
3. void clear(): This method will remove all the elements.
4. boolean contains(Object o): This method will return true if given element is present in TreeSet else it will return false.
5. Object first(): This method will return first element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
6. Object last(): This method will return last element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
7. SortedSet headSet(Object toElement): This method will return elements of TreeSet which are less than the specified element.
8. SortedSet tailSet(Object fromElement): This method will return elements of TreeSet which are greater than or equal to the specified element.
9. SortedSet subSet(Object fromElement, Object toElement): This method will return elements ranging from fromElement to toElement. fromElement is inclusive and toElement is exclusive.
10. boolean isEmpty(): This method is used to return true if this set contains no elements or is empty and false for the opposite case.
11. Object clone(): The method is used to return a shallow copy of the set, which is just a simple copied set.
12. int size(): This method is used to return the size of the set or the number of elements present in the set.
13. boolean remove(Object o): This method is used to return a specific element from the set.
14. Iterator iterator(): Returns an iterator for iterating over the elements of the set.
15. Comparator comparator(): This method will return Comparator used to sort elements in TreeSet or it will return null if default natural sorting order is used.
16. **ceiling(E e):** This method returns the least element in this set greater than or equal to the given element, or null if there is no such element.
17. **descendingIterator():** This method returns an iterator over the elements in this set in descending order.
18. **descendingSet():** This method returns a reverse order view of the elements contained in this set.
19. **floor(E e):** This method returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
20. **higher(E e):** This method returns the least element in this set strictly greater than the given element, or null if there is no such element.
21. **lower(E e):** This method returns the greatest element in this set strictly less than the given element, or null if there is no such element.
22. **pollFirst():** This method retrieves and removes the first (lowest) element, or returns null if this set is empty.
23. **pollLast():** This method retrieves and removes the last (highest) element, or returns null if this set is empty.
24. **spliterator():** This method creates a late-binding and fail-fast Spliterator over the

elements in this set.

Java LinkedList class

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to occur.
- Java LinkedList class can be used as a list, stack or queue.

Hierarchy of LinkedList class

As shown in the above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

Doubly Linked List

In the case of a doubly linked list, we can add or remove elements from both sides.

LinkedList class declaration

Let's see the declaration for java.util.LinkedList class.

1. **public class** LinkedList<E> **extends** AbstractSequentialList<E> **implements** List<E>, Deque<E>, Clon eable, Serializable

Constructors of Java LinkedList

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection<? extends E> c)	It is used to construct a list containing the elements of the specified collection, in the order, they are returned by the collection's iterator.

Methods of Java LinkedList

Method	Description
boolean add(E e)	It is used to append the specified element to the end of a list.
void add(int index, E element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(Collection<? extends E> c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean addAll(int index, Collection<? extends E> c)	It is used to append all the elements in the specified collection, starting at the specified position of the list.
void addFirst(E e)	It is used to insert the given element at the beginning of a list.
void addLast(E e)	It is used to append the given element to the end of a list.
void clear()	It is used to remove all the elements from a list.
Object clone()	It is used to return a shallow copy of an ArrayList.

boolean contains(Object o)	It is used to return true if a list contains a specified element.
Iterator<E> descendingIterator()	It is used to return an iterator over the elements in a deque in reverse sequential order.
E element()	It is used to retrieve the first element of a list.
E get(int index)	It is used to return the element at the specified position in a list.
E getFirst()	It is used to return the first element in a list.
E getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.
ListIterator<E> listIterator(int index)	It is used to return a list-iterator of the elements in proper sequence, starting at the specified position in the list.
boolean offer(E e)	It adds the specified element as the last element of a list.
boolean offerFirst(E e)	It inserts the specified element at the front of a list.
boolean offerLast(E e)	It inserts the specified element at the end of a list.
E peek()	It retrieves the first element of a list
E peekFirst()	It retrieves the first element of a list or returns null if a list is empty.
E peekLast()	It retrieves the last element of a list or returns null if a list is empty.
E poll()	It retrieves and removes the first element of a list.
E pollFirst()	It retrieves and removes the first element of a list, or returns null if a list is empty.
E pollLast()	It retrieves and removes the last element of a list, or returns null if a list is empty.

E pop()	It pops an element from the stack represented by a list.
void push(E e)	It pushes an element onto the stack represented by a list.
E remove()	It is used to retrieve and removes the first element of a list.
E remove(int index)	It is used to remove the element at the specified position in a list.
boolean remove(Object o)	It is used to remove the first occurrence of the specified element in a list.
E removeFirst()	It removes and returns the first element from a list.
boolean removeFirstOccurrence(Object o)	It is used to remove the first occurrence of the specified element in a list (when traversing the list from head to tail).
E removeLast()	It removes and returns the last element from a list.
boolean removeLastOccurrence(Object o)	It removes the last occurrence of the specified element in a list (when traversing the list from head to tail).
E set(int index, E element)	It replaces the element at the specified position in a list with the specified element.
Object[] toArray()	It is used to return an array containing all the elements in a list in proper sequence (from first to the last element).
<T> T[] toArray(T[] a)	It returns an array containing all the elements in the proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array.
int size()	It is used to return the number of elements in a list.

Java LinkedList Example

```
import java.util.*;
public class LinkedList1{
 public static void main(String args[]){

  LinkedList<String> al=new
  LinkedList<String>(); al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
```

```
Iterator<String> itr=al.iterator();
while(itr.hasNext()){
```

```
        System.out.println(itr.next());
       }
      }
}
```

Output:

```
Ravi
Vijay
Ravi
Ajay
```

Java LinkedList example to add elements

Here, we see different ways to add elements.

```
import java.util.*;
public class LinkedList2{
 public static void main(String args[]){
 LinkedList<String> ll=new LinkedList<String>();
        System.out.println("Initial list of elements:
        "+ll); ll.add("Ravi");
        ll.add("Vijay");
        ll.add("Ajay");
        System.out.println("After invoking add(E e) method: "+ll);
        //Adding an element at the specific position
        ll.add(1, "Gaurav");
        System.out.println("After invoking add(int index, E element) method: "+ll);
        LinkedList<String> ll2=new LinkedList<String>();
        ll2.add("Sonoo");
        ll2.add("Hanumat");
        //Adding second list elements to the first list
        ll.addAll(ll2);
        System.out.println("After invoking addAll(Collection<? extends E> c) method: "+ll);
        LinkedList<String> ll3=new LinkedList<String>();
        ll3.add("John");
        ll3.add("Rahul");
        //Adding second list elements to the first list at specific position
        ll.addAll(1, ll3);
        System.out.println("After invoking addAll(int index, Collection<? extends E> c) method:
        "+ll);
        //Adding an element at the first position
        ll.addFirst("Lokesh");
        System.out.println("After invoking addFirst(E e) method: "+ll);
        //Adding an element at the last position
        ll.addLast("Harsh");
        System.out.println("After invoking addLast(E e) method: "+ll);

 }
}
```

```
Initial list of elements: []
After invoking add(E e) method: [Ravi, Vijay, Ajay]
After invoking add(int index, E element) method: [Ravi, Gaurav, Vijay, Ajay]
After invoking addAll(Collection<? extends E> c) method:
```

[Ravi, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addAll(int index, Collection<? extends E> c) method:
[Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat]
After invoking addFirst(E e) method:
[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo,
Hanumat] After invoking addLast(E e) method:

```
[Lokesh, Ravi, John, Rahul, Gaurav, Vijay, Ajay, Sonoo, Hanumat, Harsh]
```

Java LinkedList example to remove elements

Here, we see different ways to remove an element.

```
import java.util.*;
public class LinkedList3 {

    public static void main(String [] args)
    {
      LinkedList<String> ll=new LinkedList<String>();
      ll.add("Ravi");
      ll.add("Vijay");
      ll.add("Ajay");
      ll.add("Anuj");
      ll.add("Gaurav");
      ll.add("Harsh");
      ll.add("Virat");
      ll.add("Gaurav");
      ll.add("Harsh");
      ll.add("Amit");
      System.out.println("Initial list of elements: "+ll);
    //Removing specific element from arraylist
        ll.remove("Vijay");
        System.out.println("After invoking remove(object) method: "+ll);
    //Removing element on the basis of specific position
        ll.remove(0);
        System.out.println("After invoking remove(index) method:
        "+ll); LinkedList<String> ll2=new LinkedList<String>();
        ll2.add("Ravi");
        ll2.add("Hanumat");
    // Adding new elements to arraylist
        ll.addAll(ll2);
        System.out.println("Updated list :
        "+ll);
    //Removing all the new elements from
        arraylist ll.removeAll(ll2);
        System.out.println("After invoking removeAll() method: "+ll);
    //Removing first element from the list
        ll.removeFirst();
        System.out.println("After invoking removeFirst() method: "+ll);
     //Removing first element from the list
        ll.removeLast();
        System.out.println("After invoking removeLast() method: "+ll);
     //Removing first occurrence of element from the list
        ll.removeFirstOccurrence("Gaurav");
        System.out.println("After invoking removeFirstOccurrence() method: "+ll);
     //Removing last occurrence of element from the list
        ll.removeLastOccurrence("Harsh");
        System.out.println("After invoking removeLastOccurrence() method: "+ll);
```

```
        //Removing all the elements available in the list
        ll.clear();
        System.out.println("After invoking clear() method: "+ll);
    }
}
```

```
Initial list of elements: [Ravi, Vijay, Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit]
After invoking remove(object) method: [Ravi, Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav,
Harsh, Amit] After invoking remove(index) method: [Ajay, Anuj, Gaurav, Harsh, Virat,
Gaurav, Harsh, Amit] Updated list : [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh, Amit,
Ravi, Hanumat]
After invoking removeAll() method: [Ajay, Anuj, Gaurav, Harsh, Virat, Gaurav, Harsh,
Amit] After invoking removeFirst() method: [Gaurav, Harsh, Virat, Gaurav, Harsh, Amit]
After invoking removeLast() method: [Gaurav, Harsh, Virat, Gaurav,
Harsh] After invoking removeFirstOccurrence() method: [Harsh, Virat,
```

Java LinkedList Example to reverse a list of elements

```
import java.util.*;
public class LinkedList4{
 public static void main(String args[]){

  LinkedList<String> ll=new LinkedList<String>();
        ll.add("Ravi");
        ll.add("Vijay");
        ll.add("Ajay");
        //Traversing the list of elements in reverse
        order Iterator i=ll.descendingIterator();
        while(i.hasNext())
        {
           System.out.println(i.next());
        }

 }
}
```

Output:

```
Ajay
Vijay
Ravi
```

Java LinkedList Example: Book

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author =
    author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class LinkedListExample {
public static void main(String[] args) {
```

```
//Creating list of Books
List<Book> list=new LinkedList<Book>();
//Creating Books
Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw
Hill",4);
```

```
        Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
        //Adding Books to list list.add(b1);
        list.add(b2);
        list.add(b3);
        //Traversing list
        for(Book b:list){
        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
        }
}
}
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4 103
Operating System Galvin Wiley 6
```

Simple example of StringTokenizer class

Let's see the simple example of StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

```
import java.util.StringTokenizer;
public class Simple{
 public static void main(String args[]){
```

```
StringTokenizer st = new StringTokenizer("my name is khan"," ");
 while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
 } } }
```

Output:my
name
is
khan

Example of nextToken(String delim) method of StringTokenizer class

```
import java.util.*;
public class Test {
  public static void main(String[] args) {
    StringTokenizer st = new StringTokenizer("my,name,is,khan");
    // printing next token
    System.out.println("Next token is : " + st.nextToken(","));
  }    }
```

Output:Next token is : my

java.util.Random

- For using this class to generate random numbers, we have to first create an instance of this class and then invoke methods such as nextInt(), nextDouble(), nextLong() etc using that instance.
- We can generate random numbers of types integers, float, double, long, booleans using this class.
- We can pass arguments to the methods for placing an upper bound on the range of the numbers to be generated. For example, nextInt(6) will generate numbers in the range 0 to 5 both inclusive.

```
// A Java program to demonstrate random number generation
// using java.util.Random;
import java.util.Random;

public class generateRandom{

   public static void main(String args[])
   {
      // create instance of Random class
      Random rand = new Random();

      // Generate random integers in range 0 to 999
      int rand_int1 = rand.nextInt(1000);
      int rand_int2 = rand.nextInt(1000);
```

```
        // Print random integers
        System.out.println("Random Integers: "+rand_int1);
        System.out.println("Random Integers: "+rand_int2);

        // Generate Random doubles
        double rand_dub1 = rand.nextDouble();
        double rand_dub2 = rand.nextDouble();

        // Print random doubles
        System.out.println("Random Doubles: "+rand_dub1);
        System.out.println("Random Doubles: "+rand_dub2);
    }}
```

Output:

```
Random Integers: 547
Random Integers: 126
Random Doubles: 0.8369779739988428
Random Doubles: 0.5497554388209912
```

Java Scanner class

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them. The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression.

Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

Commonly used methods of Scanner class

There is a list of commonly used Scanner class methods:

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.

public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Java Scanner Example to get input from console

Let's see the simple example of the Java Scanner class which reads the int, string and double value as an input:

```
import java.util.Scanner;
class ScannerTest{
public static void main(String args[]){
 Scanner sc=new Scanner(System.in);
 System.out.println("Enter your rollno");
 int rollno=sc.nextInt();
 System.out.println("Enter your name");
 String name=sc.next();
 System.out.println("Enter your fee");
 double fee=sc.nextDouble();
 System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
 sc.close();
} } Output:
```

```
Enter your rollno
111
Enter your name
Ratan
Enter
450000
Rollno:111 name:Ratan fee:450000
```

Java Calendar Class

Java Calendar class is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc. It inherits Object class and implements the Comparable interface.

Java Calendar class declaration

Let's see the declaration of java.util.Calendar class.

2. **public abstract class** Calendar **extends** Object
3. **implements** Serializable, Cloneable, Comparable<Calendar>

Java Calendar Class Example

```
import java.util.Calendar;
public class CalendarExample1 {
  public static void main(String[] args) {
  Calendar calendar = Calendar.getInstance();
  System.out.println("The current date is : " + calendar.getTime());
  calendar.add(Calendar.DATE, -15);
  System.out.println("15 days ago: " + calendar.getTime());
  calendar.add(Calendar.MONTH, 4);
  System.out.println("4 months later: " + calendar.getTime());
  calendar.add(Calendar.YEAR, 2);
  System.out.println("2 years later: " + calendar.getTime());
  } }
```

Output:

```
The current date is : Thu Jan 19 18:47:02 IST 2017
15 days ago: Wed Jan 04 18:47:02 IST 2017
4 months later: Thu May 04 18:47:02 IST 2017
2 years later: Sat May 04 18:47:02 IST 2019
```

UNIT-IV

Java AWT Tutorial

Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
1. import java.awt.*;
2. class First extends Frame{
3. First(){
4. Button b=new Button("click me");
5. b.setBounds(30,100,80,30);// setting button position
6. add(b);//adding button into frame
7. setSize(300,300);//frame size 300 width and 300 height
8. setLayout(null);//no layout manager
9. setVisible(true);//now frame will be visible, by default not visible
10. }
11. public static void main(String args[]){
12. First f=new First();
13. } }
```

The setBounds(int x axis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
1.  import java.awt.*;
2.  class First2{
3.  First2(){
4.  Frame f=new Frame();
5.  Button b=new Button("click me");
6.  b.setBounds(30,50,80,30);
7.  f.add(b);
8.  f.setSize(300,300);
9.  f.setLayout(null);
10. f.setVisible(true);
11. }
12. public static void main(String args[]){
13. First2 f=new First2();
14. }}
```

AWT Controls:

AWT Label:

The object of Label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly.

AWT Label Class Declaration

1. **public class** Label **extends** Component **implements** Accessible

Java Label Example

```
1. import java.awt.*;
2. class LabelExample{
3. public static void main(String args[]){
4.     Frame f= new Frame("Label Example");
5.     Label l1,l2;
6.     l1=new Label("First Label.");
7.     l1.setBounds(50,100, 100,30);
8.     l2=new Label("Second Label.");
9.     l2.setBounds(50,150, 100,30);
10.    f.add(l1); f.add(l2);
11.    f.setSize(400,400);
12.    f.setLayout(null);
13.    f.setVisible(true);
14. }
15. }
```

Output:

Java AWT Button

The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

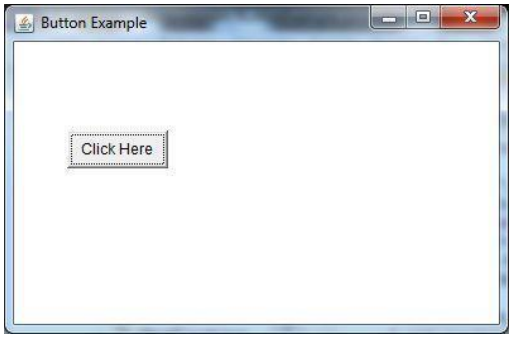
AWT Button Class declaration

1. **public class** Button **extends** Component **implements** Accessible

Java AWT Button Example

```
1.  import java.awt.*;
2.  public class ButtonExample {
3.  public static void main(String[] args) {
4.      Frame f=new Frame("Button Example");
5.      Button b=new Button("Click Here");
6.      b.setBounds(50,100,80,30);
7.      f.add(b);
8.      f.setSize(400,400);
9.      f.setLayout(null);
10.     f.setVisible(true);
11. }
12. }
```

Output:



Java AWT Scrollbar

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

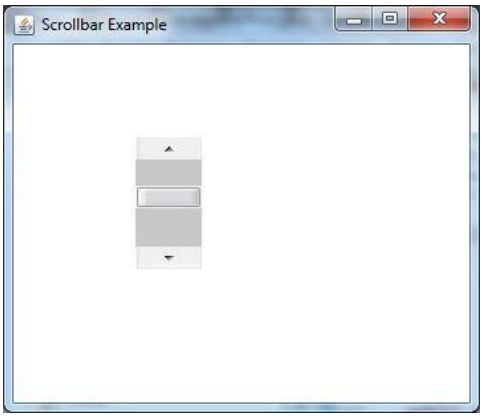
AWT Scrollbar class declaration

1. **public class** Scrollbar **extends** Component **implements** Adjustable, Accessible

Java AWT Scrollbar Example

```
1.  import java.awt.*;
2.  class ScrollbarExample{
3.  ScrollbarExample(){
4.          Frame f= new Frame("Scrollbar Example");
5.          Scrollbar s=new Scrollbar();
6.          s.setBounds(100,100, 50,100);
7.          f.add(s);
8.          f.setSize(400,400);
9.          f.setLayout(null);
10.         f.setVisible(true);
11. }
12. public static void main(String args[]){
13.      new ScrollbarExample();
14. }
15. }
```

Output:



Text Components:

Java AWT TextField

The object of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

AWT TextField Class Declaration

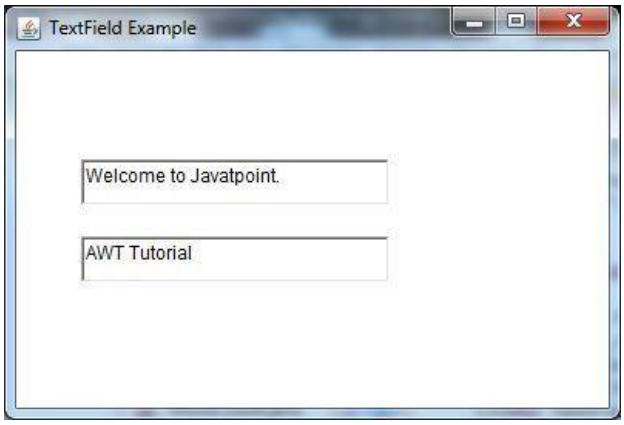
1. **public class** TextField **extends** TextComponent

Java AWT TextField Example

```
1. import java.awt.*;
2. class TextFieldExample{
3. public static void main(String args[]){
4.     Frame f= new Frame("TextField Example");
5.     TextField t1,t2;
6.     t1=new TextField("Welcome to Javatpoint.");
7.     t1.setBounds(50,100, 200,30);
8.     t2=new TextField("AWT Tutorial");
9.     t2.setBounds(50,150, 200,30);
10.    f.add(t1); f.add(t2);
11.    f.setSize(400,400);
```

```
12.     f.setLayout(null);
13.     f.setVisible(true);
14. }
15. }
```

Output:



Java AWT TextArea

The object of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

AWT TextArea Class Declaration

```
1. public class TextArea extends TextComponent
```

Java AWT TextArea Example

```
1.  import java.awt.*;
2.  public class TextAreaExample
3.  {
4.       TextAreaExample(){
5.          Frame f= new Frame();
6.          TextArea area=new TextArea("Welcome to javatpoint"); 7.
            area.setBounds(10,30, 300,300);
8.          f.add(area);
```

```
9.          f.setSize(400,400);
10.         f.setLayout(null);
11.         f.setVisible(true);
12.     }
13. public static void main(String args[])
14. {
15.    new TextAreaExample();
16. }
17. }
```

Output:

Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

AWT Checkbox Class Declaration

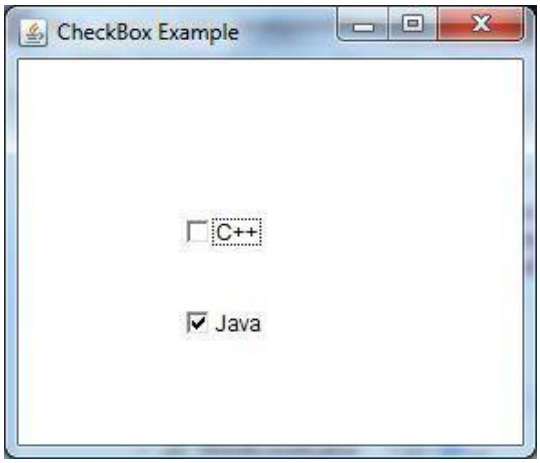
```
1. public class Checkbox extends Component implements ItemSelectable, Accessible
```

Java AWT Checkbox Example

```
1.  import java.awt.*;
2.  public class CheckboxExample
3. {
4.       CheckboxExample(){
```

```
5.          Frame f= new Frame("Checkbox Example");
6.          Checkbox checkbox1 = new Checkbox("C++");
7.          checkbox1.setBounds(100,100, 50,50);
8.          Checkbox checkbox2 = new Checkbox("Java", true);
9.          checkbox2.setBounds(100,150, 50,50);
10.         f.add(checkbox1);
11.         f.add(checkbox2);
12.         f.setSize(400,400);
13.         f.setLayout(null);
14.         f.setVisible(true);
15.     }
16. public static void main(String args[])
17. {
18.     new CheckboxExample();
19. }
20. }
```

Output:



Java AWT CheckboxGroup

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Note: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

AWT CheckboxGroup Class Declaration

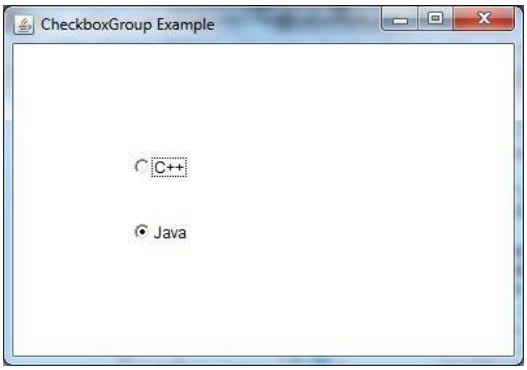
1. **public class** CheckboxGroup **extends** Object **implements** Serializable

Java AWT CheckboxGroup Example

```
1.  import java.awt.*;
2.  public class CheckboxGroupExample
3. {
4.       CheckboxGroupExample(){
5.       Frame f= new Frame("CheckboxGroup Example");
6.        CheckboxGroup cbg = new CheckboxGroup();
7.        Checkbox checkBox1 = new Checkbox("C++", cbg, false);
8.        checkBox1.setBounds(100,100, 50,50);
9.        Checkbox checkBox2 = new Checkbox("Java", cbg, true);
10.       checkBox2.setBounds(100,150, 50,50);
11.       f.add(checkBox1);
12.       f.add(checkBox2);
13.       f.setSize(400,400);
14.       f.setLayout(null);
15.       f.setVisible(true);
16.    }
17. public static void main(String args[])
```

```
18. {
19.     new CheckboxGroupExample();
20. }
21. }
```

Output:



Java AWT Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

AWT Choice Class Declaration

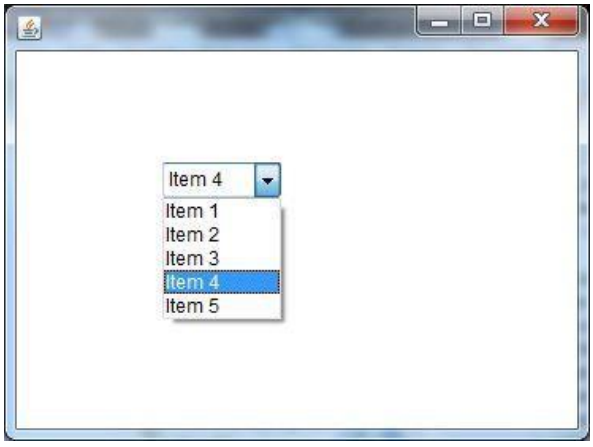
```
1. public class Choice extends Component implements ItemSelectable, Accessible
```

Java AWT Choice Example

```
1.  import java.awt.*;
2.  public class ChoiceExample
3. {
4.       ChoiceExample(){
5.       Frame f= new Frame();
6.       Choice c=new Choice();
7.       c.setBounds(100,100, 75,75);
8.       c.add("Item 1");
9.       c.add("Item 2");
```

```
10.         c.add("Item 3");
11.         c.add("Item 4");
12.         c.add("Item 5");
13.         f.add(c);
14.         f.setSize(400,400);
15.         f.setLayout(null);
16.         f.setVisible(true);
17.      }
18. public static void main(String args[])
19. {
20.    new ChoiceExample();
21. }
22. }
```

Output:



Java AWT List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

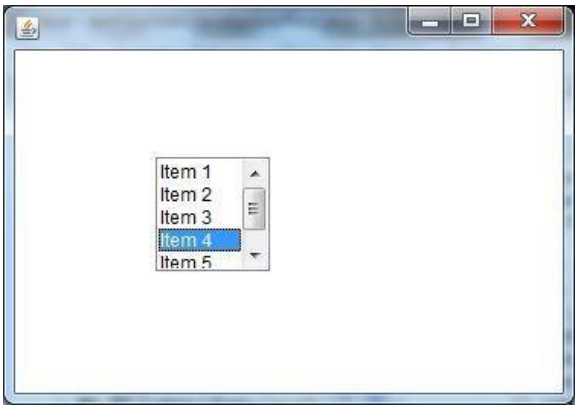
AWT List class Declaration

1. **public class** List **extends** Component **implements** ItemSelectable, Accessible

Java AWT List Example

```
1.  import java.awt.*;
2.  public class ListExample
3. {
4.      ListExample(){
5.         Frame f= new Frame();
6.         List l1=new List(5);
7.         l1.setBounds(100,100, 75,75);
8.         l1.add("Item 1");
9.         l1.add("Item 2");
10.        l1.add("Item 3");
11.        l1.add("Item 4");
12.        l1.add("Item 5");
13.        f.add(l1);
14.        f.setSize(400,400);
15.        f.setLayout(null);
16.        f.setVisible(true);
17.     }
18. public static void main(String args[])
19. {
20.   new ListExample();
21. }
22. }
```

Output:



Java AWT Dialog

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.

Unlike Frame, it doesn't have maximize and minimize buttons.

Frame vs Dialog

Frame and Dialog both inherits Window class. Frame has maximize and minimize buttons but Dialog doesn't have.

AWT Dialog class declaration

1. **public class** Dialog **extends** Window

Java AWT Dialog Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class DialogExample {
4.      private static Dialog d;
5.      DialogExample() {
6.          Frame f= new Frame();
7.          d = new Dialog(f , "Dialog Example", true);
8.          d.setLayout( new FlowLayout() );
9.          Button b = new Button ("OK");
10.         b.addActionListener ( new ActionListener()
11.         {
12.             public void actionPerformed( ActionEvent e )
13.             {
14.                 DialogExample.d.setVisible(false);
15.             }
16.         });
```

```
17.         d.add( new Label ("Click button to continue."));
18.         d.add(b);
19.         d.setSize(300,300);
20.         d.setVisible(true);
21.     }
22.     public static void main(String args[])
23.     {
24.         new DialogExample();
25.     }
26. }
```

Output:

Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

AWT MenuItem class declaration

```
1. public class MenuItem extends MenuComponent implements Accessible
```

AWT Menu class declaration

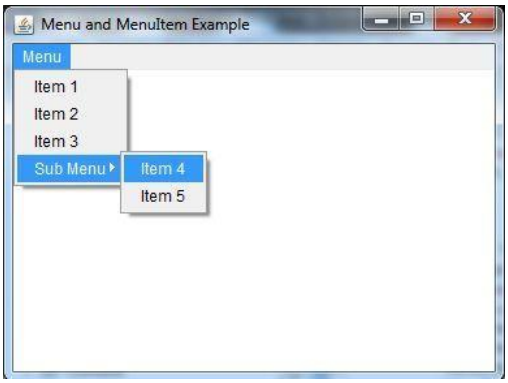
```
1. public class Menu extends MenuItem implements MenuContainer, Accessible
```

Java AWT MenuItem and Menu Example

```
1.  import java.awt.*;
2.  class MenuExample
3. {
4.       MenuExample(){
```

```
5.          Frame f= new Frame("Menu and MenuItem Example");
6.          MenuBar mb=new MenuBar();
7.          Menu menu=new Menu("Menu");
8.          Menu submenu=new Menu("Sub Menu");
9.          MenuItem i1=new MenuItem("Item 1");
10.         MenuItem i2=new MenuItem("Item 2");
11.         MenuItem i3=new MenuItem("Item 3");
12.         MenuItem i4=new MenuItem("Item 4");
13.         MenuItem i5=new MenuItem("Item 5");
14.         menu.add(i1);
15.         menu.add(i2);
16.         menu.add(i3);
17.         submenu.add(i4);
18.         submenu.add(i5);
19.         menu.add(submenu);
20.         mb.add(menu);
21.         f.setMenuBar(mb);
22.         f.setSize(400,400);
23.         f.setLayout(null);
24.         f.setVisible(true);
25. }
26. public static void main(String args[])
27. {
28. new MenuExample();
29. }
30. }
```

Output:



Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- ***Button***
  - public void addActionListener(ActionListener a){}
- ***MenuItem***
  - public void addActionListener(ActionListener a){}
- ***TextField***
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- ***TextArea***
  - public void addTextListener(TextListener a){}
- ***Checkbox***
  - public void addItemListener(ItemListener a){}
- ***Choice***
  - public void addItemListener(ItemListener a){}
- ***List***
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java event handling by implementing ActionListener

```
1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent extends Frame implements ActionListener{
4. TextField tf;
5. AEvent()
6. {
```

```
7.   //create components
8.   tf=new TextField();
9.  tf.setBounds(60,50,170,20);
10. Button b=new Button("click me");
11. b.setBounds(100,120,80,30);
12.
13. //register listener
14. b.addActionListener(this);//passing current instance
15.
16. //add components and set size, layout and visibility
17. add(b);add(tf);
18. setSize(300,300);
19. setLayout(null);
20. setVisible(true);
21. }
22. public void actionPerformed(ActionEvent e){
23. tf.setText("Welcome");
24. }
25. public static void main(String args[]){
26. new AEvent();
27. }
28. }
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.

Output:

Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

Java

MouseListener

Example 1

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample extends Frame implements MouseListener{
```

4. Label l;

```
5.  MouseListenerExample(){
6.      addMouseListener(this);
7.
8.      l=new Label();
9.      l.setBounds(20,50,100,20);
10.     add(l);
11.     setSize(300,300);
12.     setLayout(null);
13.     setVisible(true);
14. }
15. public void mouseClicked(MouseEvent e) {
16.     l.setText("Mouse Clicked");
17. }
18. public void mouseEntered(MouseEvent e) {
19.     l.setText("Mouse Entered");
20. }
21. public void mouseExited(MouseEvent e) {
22.     l.setText("Mouse Exited");
23. }
24. public void mousePressed(MouseEvent e) {
25.     l.setText("Mouse Pressed");
26. }
27. public void mouseReleased(MouseEvent e) {
28.     l.setText("Mouse Released");
29. }
```

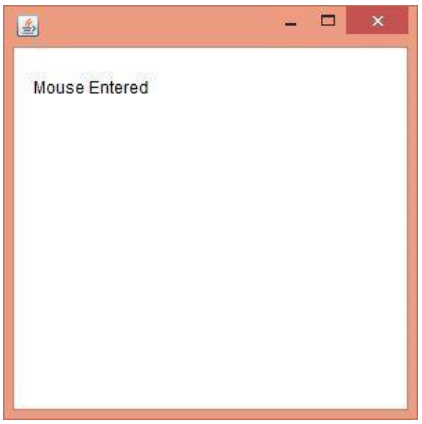
30. **public static void** main(String[] args) {

31. **new** MouseListenerExample();

32. }

33. }

Output:



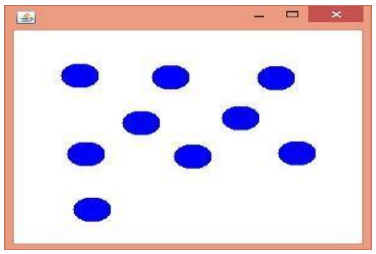
Java MouseListener

Example 2

1. **import** java.awt.*;
2. **import** java.awt.event.*;
3. **public class** MouseListenerExample2 **extends** Frame **implements** MouseListener{
4. MouseListenerExample2(){
5. addMouseListener(**this**);
6.
7. setSize(300,300);
8. setLayout(**null**);
9. setVisible(**true**);
10. }
11. **public void** mouseClicked(MouseEvent e) {

```
12.         Graphics g=getGraphics();
13.         g.setColor(Color.BLUE);
14.         g.fillOval(e.getX(),e.getY(),30,30);
15.     }
16.     public void mouseEntered(MouseEvent e) {}
17.     public void mouseExited(MouseEvent e) {}
18.     public void mousePressed(MouseEvent e) {}
19.     public void mouseReleased(MouseEvent e) {}
20.
21. public static void main(String[] args) {
22.     new MouseListenerExample2();
23. }
24. }
```

Output:



Java KeyListener Interface

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.

Methods of KeyListener interface

The signature of 3 methods found in KeyListener interface are given below:

1. **public abstract void** keyPressed(KeyEvent e);
2. **public abstract void** keyReleased(KeyEvent e);

3. **public abstract void** keyTyped(KeyEvent e);

Java

KeyListener

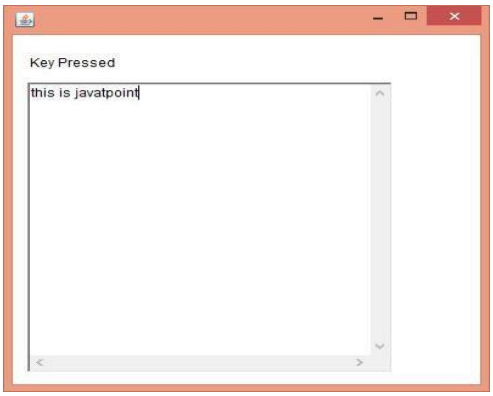
Example 1

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class KeyListenerExample extends Frame implements KeyListener{
4.      Label l;
5.      TextArea area;
6.      KeyListenerExample(){
7.
8.          l=new Label();
9.          l.setBounds(20,50,100,20);
10.         area=new TextArea();
11.         area.setBounds(20,80,300, 300);
12.         area.addKeyListener(this);
13.
14.         add(l);add(area);
15.         setSize(400,400);
16.         setLayout(null);
17.         setVisible(true);
18.     }
19.     public void keyPressed(KeyEvent e) {
20.         l.setText("Key Pressed");
21.     }
```

22. **public void** keyReleased(KeyEvent e) {

```
23.         l.setText("Key Released");
24.     }
25.     public void keyTyped(KeyEvent e) {
26.         l.setText("Key Typed");
27.     }
28.
29.     public static void main(String[] args) {
30.         new KeyListenerExample();
31.     }
32. }
```

Output:



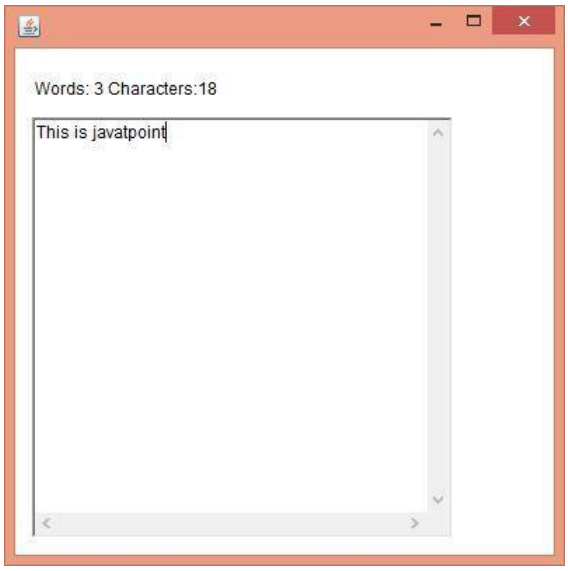
Java KeyListener

Example 2: Count Words & Characters

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class KeyListenerExample extends Frame implements KeyListener{
4.     Label l;
5.     TextArea area;
```

```
6.      KeyListenerExample(){
7.
8.          l=new Label();
9.          l.setBounds(20,50,200,20);
10.         area=new TextArea();
11.         area.setBounds(20,80,300, 300);
12.         area.addKeyListener(this);
13.
14.         add(l);add(area);
15.         setSize(400,400);
16.         setLayout(null);
17.         setVisible(true);
18.     }
19.     public void keyPressed(KeyEvent e) {}
20.     public void keyReleased(KeyEvent e) {
21.         String text=area.getText();
22.         String words[]=text.split("\\s");
23.         l.setText("Words: "+words.length+" Characters:"+text.length());
24.     }
25.     public void keyTyped(KeyEvent e) {}
26.
27.     public static void main(String[] args) {
28.         new KeyListenerExample();
29.     }
30. }
```

Output:



Java Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The adapter classes are found in **java.awt.event** package. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

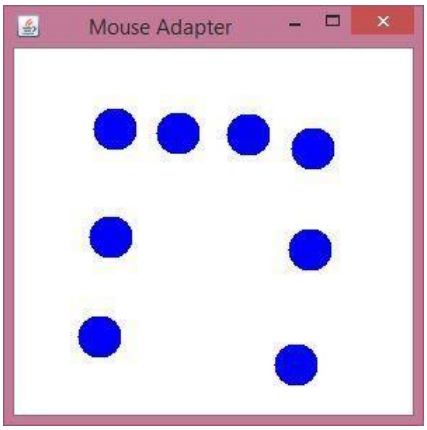
java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

Java MouseAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseAdapterExample extends MouseAdapter{
4.    Frame f;
5.    MouseAdapterExample(){
6.      f=new Frame("Mouse Adapter");
7.      f.addMouseListener(
this); 8.
9.      f.setSize(300,300);
10.     f.setLayout(null);
11.     f.setVisible(true);
12.   }
13.   public void mouseClicked(MouseEvent e) {
14.     Graphics g=f.getGraphics();
15.     g.setColor(Color.BLUE);
16.     g.fillOval(e.getX(),e.getY(),30,30);
17.   }
18.
19. public static void main(String[] args) {
20.   new MouseAdapterExample();
21. }
22. }
```

Output:

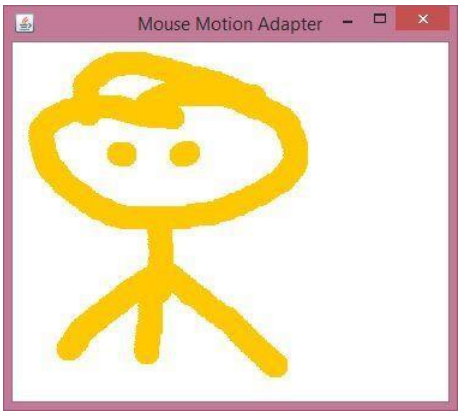


Java MouseMotionAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseMotionAdapterExample extends MouseMotionAdapter{
4.     Frame f;
5.     MouseMotionAdapterExample(){
6.         f=new Frame("Mouse Motion Adapter");
7.         f.addMouseMotionListene
r(this); 8.
9.         f.setSize(300,300);
10.        f.setLayout(null);
11.        f.setVisible(true);
12.    }
13. public void mouseDragged(MouseEvent e) {
14.    Graphics g=f.getGraphics();
15.    g.setColor(Color.ORANGE);
16.    g.fillOval(e.getX(),e.getY(),20,20);
17. }
18. public static void main(String[] args) {
19.    new MouseMotionAdapterExample();
20. }
21. }
```

Output

:



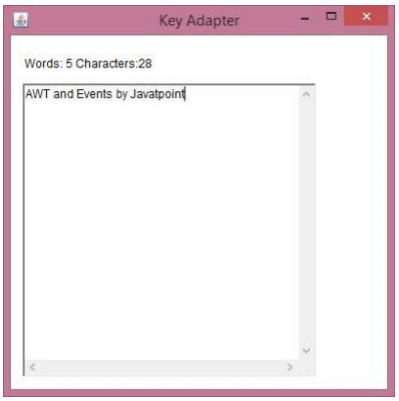
Java KeyAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class KeyAdapterExample extends KeyAdapter{
4.    Label l;
5.    TextArea area;
6.    Frame f;
7.    KeyAdapterExample(){
8.       f=new Frame("Key Adapter");

9.       l=new Label();
10.      l.setBounds(20,50,200,20);
11.      area=new TextArea();
12.      area.setBounds(20,80,300, 300);
13.
         area.addKeyListener(
this); 14.
15.      f.add(l);f.add(area);
16.      f.setSize(400,400);
17.      f.setLayout(null);
18.      f.setVisible(true);
19.   }
20.   public void keyReleased(KeyEvent e) {
21.      String text=area.getText();
22.      String words[]=text.split("\\s");
23.      l.setText("Words: "+words.length+" Characters:"+text.length());
24.   }
25.
26.   public static void main(String[] args) {
27.      new KeyAdapterExample();
28.   }
29. }
```

Output

:



UNIT-V

Java AWT Tutorial

Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (association)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
1. import java.awt.*;
2. class First extends Frame{
3. First(){
4. Button b=new Button("click me");
5. b.setBounds(30,100,80,30);// setting button position
6. add(b);//adding button into frame
7. setSize(300,300);//frame size 300 width and 300 height
8. setLayout(null);//no layout manager
9. setVisible(true);//now frame will be visible, by default not visible
10. }
11. public static void main(String args[]){
12. First f=new First();
13. } }
```

The setBounds(int x axis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
1. import java.awt.*;
2. class First2{
3. First2(){
4. Frame f=new Frame();
5. Button b=new Button("click me");
6. b.setBounds(30,50,80,30);
7. f.add(b);
8. f.setSize(300,300);
9. f.setLayout(null);
10. f.setVisible(true);
11. }
12. public static void main(String args[]){
13. First2 f=new First2();
14. }}
```

AWT Controls:

AWT Label:

The object of Label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly.

AWT Label Class Declaration

1. **public class** Label **extends** Component **implements** Accessible

Java Label Example

```
1. import java.awt.*;
2. class LabelExample{
3. public static void main(String args[]){
4.     Frame f= new Frame("Label Example");
5.     Label l1,l2;
6.     l1=new Label("First Label.");
7.     l1.setBounds(50,100, 100,30);
8.     l2=new Label("Second Label.");
9.     l2.setBounds(50,150, 100,30);
10.    f.add(l1); f.add(l2);
11.    f.setSize(400,400);
12.    f.setLayout(null);
13.    f.setVisible(true);
14. }
15. }
```

Output:

Java AWT Button

The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

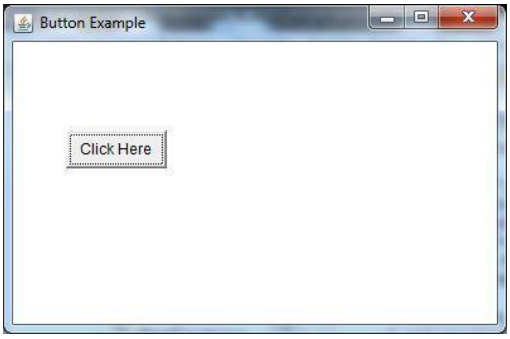
AWT Button Class declaration

1. **public class** Button **extends** Component **implements** Accessible

Java AWT Button Example

```
1.  import java.awt.*;
2.  public class ButtonExample {
3.  public static void main(String[] args) {
4.      Frame f=new Frame("Button Example");
5.      Button b=new Button("Click Here");
6.      b.setBounds(50,100,80,30);
7.      f.add(b);
8.      f.setSize(400,400);
9.      f.setLayout(null);
10.     f.setVisible(true);
11. }
12. }
```

Output:



Java AWT Scrollbar

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

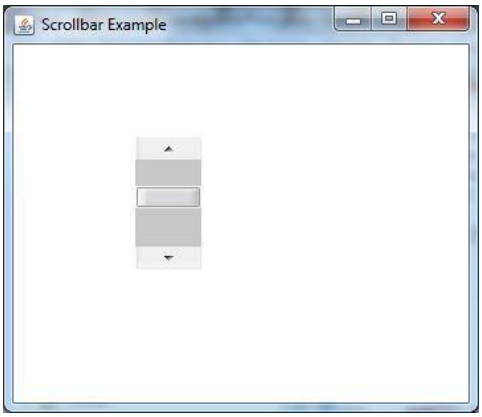
AWT Scrollbar class declaration

1. **public class** Scrollbar **extends** Component **implements** Adjustable, Accessible

Java AWT Scrollbar Example

```
1.  import java.awt.*;
2.  class ScrollbarExample{
3.  ScrollbarExample(){
4.          Frame f= new Frame("Scrollbar Example");
5.          Scrollbar s=new Scrollbar();
6.          s.setBounds(100,100, 50,100);
7.          f.add(s);
8.          f.setSize(400,400);
9.          f.setLayout(null);
10.         f.setVisible(true);
11. }
12. public static void main(String args[]){
13.     new ScrollbarExample();
14. }
15. }
```

Output:



Text Components:

Java AWT TextField

The object of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

AWT TextField Class Declaration

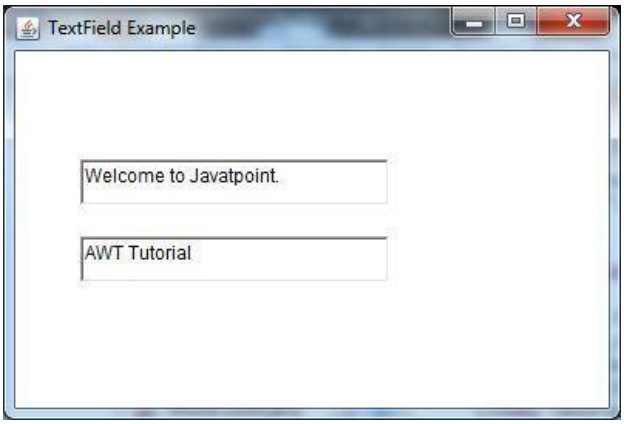
1. **public class** TextField **extends** TextComponent

Java AWT TextField Example

```
1. import java.awt.*;
2. class TextFieldExample{
3. public static void main(String args[]){
4.     Frame f= new Frame("TextField Example");
5.     TextField t1,t2;
6.     t1=new TextField("Welcome to Javatpoint.");
7.     t1.setBounds(50,100, 200,30);
8.     t2=new TextField("AWT Tutorial");
9.     t2.setBounds(50,150, 200,30);
10.    f.add(t1); f.add(t2);
11.    f.setSize(400,400);
```

```
12.     f.setLayout(null);
13.     f.setVisible(true);
14. }
15. }
```

Output:



Java AWT TextArea

The object of a TextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

AWT TextArea Class Declaration

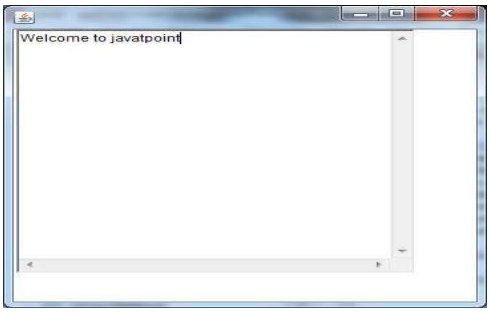
```
1. public class TextArea extends TextComponent
```

Java AWT TextArea Example

```
1.  import java.awt.*;
2.  public class TextAreaExample
3.  {
4.       TextAreaExample(){
5.          Frame f= new Frame();
6.          TextArea area=new TextArea("Welcome to javatpoint"); 7.
            area.setBounds(10,30, 300,300);
8.          f.add(area);
```

```
9.          f.setSize(400,400);
10.         f.setLayout(null);
11.         f.setVisible(true);
12.     }
13. public static void main(String args[])
14. {
15.    new TextAreaExample();
16. }
17. }
```

Output:



Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

AWT Checkbox Class Declaration

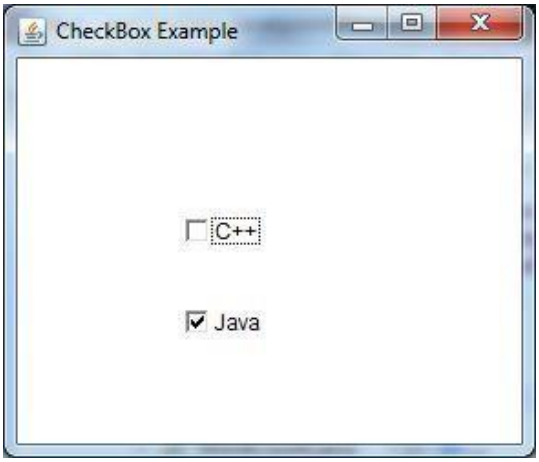
```
1. public class Checkbox extends Component implements ItemSelectable, Accessible
```

Java AWT Checkbox Example

```
1.  import java.awt.*;
2.  public class CheckboxExample
3. {
4.       CheckboxExample(){
```

```
5.          Frame f= new Frame("Checkbox Example");
6.          Checkbox checkbox1 = new Checkbox("C++");
7.          checkbox1.setBounds(100,100, 50,50);
8.          Checkbox checkbox2 = new Checkbox("Java", true);
9.          checkbox2.setBounds(100,150, 50,50);
10.         f.add(checkbox1);
11.         f.add(checkbox2);
12.         f.setSize(400,400);
13.         f.setLayout(null);
14.         f.setVisible(true);
15.     }
16. public static void main(String args[])
17. {
18.    new CheckboxExample();
19. }
20. }
```

Output:



Java AWT CheckboxGroup

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Note: CheckboxGroup enables you to create radio buttons in AWT. There is no special control for creating radio buttons in AWT.

AWT CheckboxGroup Class Declaration

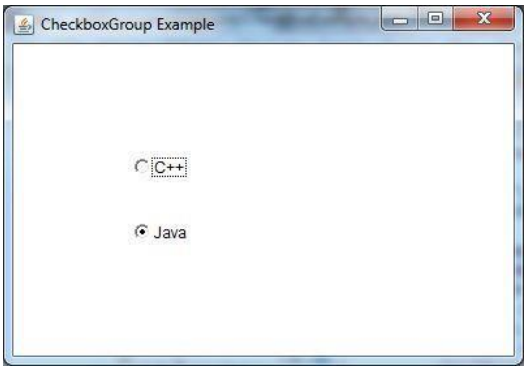
1. **public class** CheckboxGroup **extends** Object **implements** Serializable

Java AWT CheckboxGroup Example

```
1.  import java.awt.*;
2.  public class CheckboxGroupExample
3. {
4.       CheckboxGroupExample(){
5.       Frame f= new Frame("CheckboxGroup Example");
6.        CheckboxGroup cbg = new CheckboxGroup();
7.        Checkbox checkBox1 = new Checkbox("C++", cbg, false);
8.        checkBox1.setBounds(100,100, 50,50);
9.        Checkbox checkBox2 = new Checkbox("Java", cbg, true);
10.       checkBox2.setBounds(100,150, 50,50);
11.       f.add(checkBox1);
12.       f.add(checkBox2);
13.       f.setSize(400,400);
14.       f.setLayout(null);
15.       f.setVisible(true);
16.    }
17. public static void main(String args[])
```

```
18. {
19.     new CheckboxGroupExample();
20. }
21. }
```

Output:



Java AWT Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

AWT Choice Class Declaration

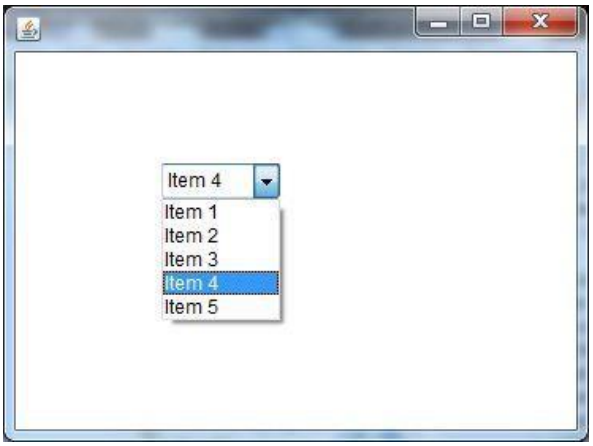
```
1. public class Choice extends Component implements ItemSelectable, Accessible
```

Java AWT Choice Example

```
1.  import java.awt.*;
2.  public class ChoiceExample
3. {
4.       ChoiceExample(){
5.       Frame f= new Frame();
6.       Choice c=new Choice();
7.       c.setBounds(100,100, 75,75);
8.       c.add("Item 1");
9.       c.add("Item 2");
```

```
10.         c.add("Item 3");
11.         c.add("Item 4");
12.         c.add("Item 5");
13.         f.add(c);
14.         f.setSize(400,400);
15.         f.setLayout(null);
16.         f.setVisible(true);
17.      }
18. public static void main(String args[])
19. {
20.    new ChoiceExample();
21. }
22. }
```

Output:



Java AWT List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

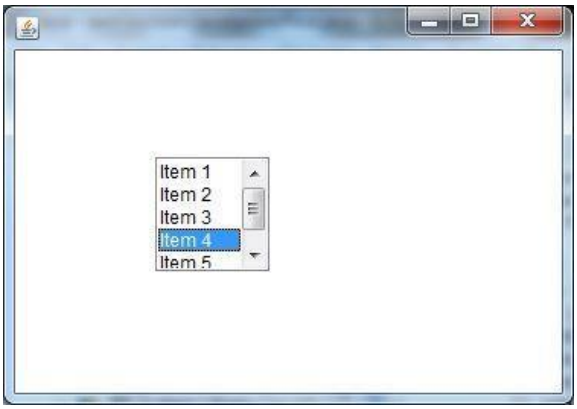
AWT List class Declaration

1. **public class** List **extends** Component **implements** ItemSelectable, Accessible

Java AWT List Example

```
1.  import java.awt.*;
2.  public class ListExample
3. {
4.      ListExample(){
5.         Frame f= new Frame();
6.         List l1=new List(5);
7.         l1.setBounds(100,100, 75,75);
8.         l1.add("Item 1");
9.         l1.add("Item 2");
10.        l1.add("Item 3");
11.        l1.add("Item 4");
12.        l1.add("Item 5");
13.        f.add(l1);
14.        f.setSize(400,400);
15.        f.setLayout(null);
16.        f.setVisible(true);
17.     }
18. public static void main(String args[])
19. {
20.    new ListExample();
21. }
22. }
```

Output:



Java AWT Dialog

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.

Unlike Frame, it doesn't have maximize and minimize buttons.

Frame vs Dialog

Frame and Dialog both inherits Window class. Frame has maximize and minimize buttons but Dialog doesn't have.

AWT Dialog class declaration

1. **public class** Dialog **extends** Window

Java AWT Dialog Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class DialogExample {
4.      private static Dialog d;
5.      DialogExample() {
6.          Frame f= new Frame();
7.          d = new Dialog(f , "Dialog Example", true);
8.          d.setLayout( new FlowLayout() );
9.          Button b = new Button ("OK");
10.         b.addActionListener ( new ActionListener()
11.         {
12.             public void actionPerformed( ActionEvent e )
13.             {
14.                 DialogExample.d.setVisible(false);
15.             }
16.         });
```

```
17.         d.add( new Label ("Click button to continue."));
18.         d.add(b);
19.         d.setSize(300,300);
20.         d.setVisible(true);
21.     }
22.     public static void main(String args[])
23.     {
24.         new DialogExample();
25.     }
26. }
```

Output:

Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

AWT MenuItem class declaration

```
1. public class MenuItem extends MenuComponent implements Accessible
```

AWT Menu class declaration

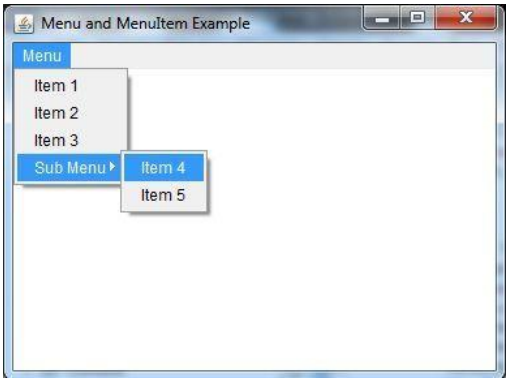
```
1. public class Menu extends MenuItem implements MenuContainer, Accessible
```

Java AWT MenuItem and Menu Example

```
1.  import java.awt.*;
2.  class MenuExample
3. {
4.       MenuExample(){
```

```
5.          Frame f= new Frame("Menu and MenuItem Example");
6.          MenuBar mb=new MenuBar();
7.          Menu menu=new Menu("Menu");
8.          Menu submenu=new Menu("Sub Menu");
9.          MenuItem i1=new MenuItem("Item 1");
10.         MenuItem i2=new MenuItem("Item 2");
11.         MenuItem i3=new MenuItem("Item 3");
12.         MenuItem i4=new MenuItem("Item 4");
13.         MenuItem i5=new MenuItem("Item 5");
14.         menu.add(i1);
15.         menu.add(i2);
16.         menu.add(i3);
17.         submenu.add(i4);
18.         submenu.add(i5);
19.         menu.add(submenu);
20.         mb.add(menu);
21.         f.setMenuBar(mb);
22.         f.setSize(400,400);
23.         f.setLayout(null);
24.         f.setVisible(true);
25. }
26. public static void main(String args[])
27. {
28. new MenuExample();
29. }
30. }
```

Output:



Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **MenuItem**
  - public void addActionListener(ActionListener a){}
- **TextField**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **TextArea**
  - public void addTextListener(TextListener a){}
- **Checkbox**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java event handling by implementing ActionListener

```
1. import java.awt.*;
2. import java.awt.event.*;
3. class AEvent extends Frame implements ActionListener{
4. TextField tf;
5. AEvent()
6. {
```

```
7.   //create components
8.   tf=new TextField();
9.  tf.setBounds(60,50,170,20);
10. Button b=new Button("click me");
11. b.setBounds(100,120,80,30);
12.
13. //register listener
14. b.addActionListener(this);//passing current instance
15.
16. //add components and set size, layout and visibility
17. add(b);add(tf);
18. setSize(300,300);
19. setLayout(null);
20. setVisible(true);
21. }
22. public void actionPerformed(ActionEvent e){
23. tf.setText("Welcome");
24. }
25. public static void main(String args[]){
26. new AEvent();
27. }
28. }
```

public void setBounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, textfield etc.

Output:

Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

Java

MouseList

ener

Example 1

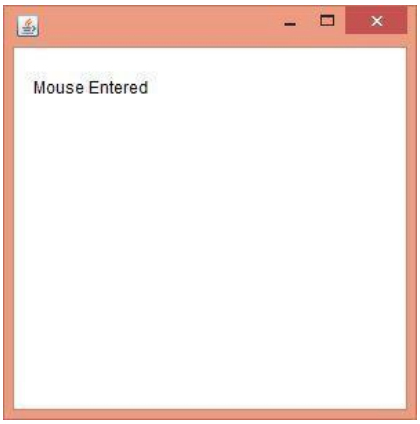
```
1. import java.awt.*;
```

```
2. import java.awt.event.*;
3. public class MouseListenerExample extends Frame implements MouseListener{
4.     Label l;
```

```
5.    MouseListenerExample(){
6.        addMouseListener(this);
7.
8.        l=new Label();
9.        l.setBounds(20,50,100,20);
10.       add(l);
11.       setSize(300,300);
12.       setLayout(null);
13.       setVisible(true);
14.   }
15.   public void mouseClicked(MouseEvent e) {
16.       l.setText("Mouse Clicked");
17.   }
18.   public void mouseEntered(MouseEvent e) {
19.       l.setText("Mouse Entered");
20.   }
21.   public void mouseExited(MouseEvent e) {
22.       l.setText("Mouse Exited");
23.   }
24.   public void mousePressed(MouseEvent e) {
25.       l.setText("Mouse Pressed");
26.   }
27.   public void mouseReleased(MouseEvent e) {
28.       l.setText("Mouse Released");
29.   }
```

```
30. public static void main(String[] args) {
31.     new MouseListenerExample();
32. }
33. }
```

Output:



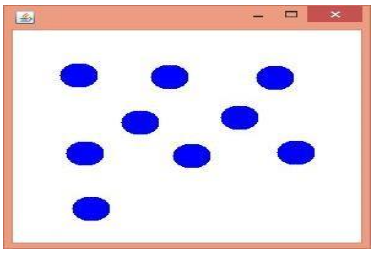
Java MouseListener

Example 2

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseListenerExample2 extends Frame implements MouseListener{
4.     MouseListenerExample2(){
5.         addMouseListener(this);
6.
7.         setSize(300,300);
8.         setLayout(null);
9.         setVisible(true);
10.    }
11.    public void mouseClicked(MouseEvent e) {
```

```
12.         Graphics g=getGraphics();
13.         g.setColor(Color.BLUE);
14.         g.fillOval(e.getX(),e.getY(),30,30);
15.     }
16.     public void mouseEntered(MouseEvent e) {}
17.     public void mouseExited(MouseEvent e) {}
18.     public void mousePressed(MouseEvent e) {}
19.     public void mouseReleased(MouseEvent e) {}
20.
21. public static void main(String[] args) {
22.     new MouseListenerExample2();
23. }
24. }
```

Output:



Java KeyListener Interface

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.

Methods of KeyListener interface

The signature of 3 methods found in KeyListener interface are given below:

1. **public abstract void** keyPressed(KeyEvent e);
2. **public abstract void** keyReleased(KeyEvent e);

3. **public abstract void** keyTyped(KeyEvent e);

Java

KeyListen

er

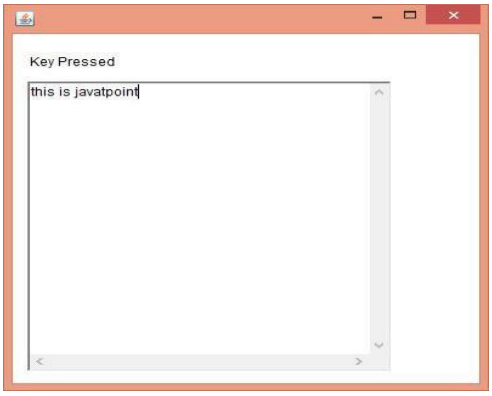
Example 1

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class KeyListenerExample extends Frame implements KeyListener{
4.     Label l;
5.     TextArea area;
6.     KeyListenerExample(){
7.
8.         l=new Label();
9.         l.setBounds(20,50,100,20);
10.        area=new TextArea();
11.        area.setBounds(20,80,300, 300);
12.        area.addKeyListener(this);
13.
14.        add(l);add(area);
15.        setSize(400,400);
16.        setLayout(null);
17.        setVisible(true);
18.    }
19.    public void keyPressed(KeyEvent e) {
20.        l.setText("Key Pressed");
```

```
21.     }
22.     public void keyReleased(KeyEvent e) {
```

```
23.         l.setText("Key Released");
24.     }
25.     public void keyTyped(KeyEvent e) {
26.         l.setText("Key Typed");
27.     }
28.
29.     public static void main(String[] args) {
30.         new KeyListenerExample();
31.     }
32. }
```

Output:



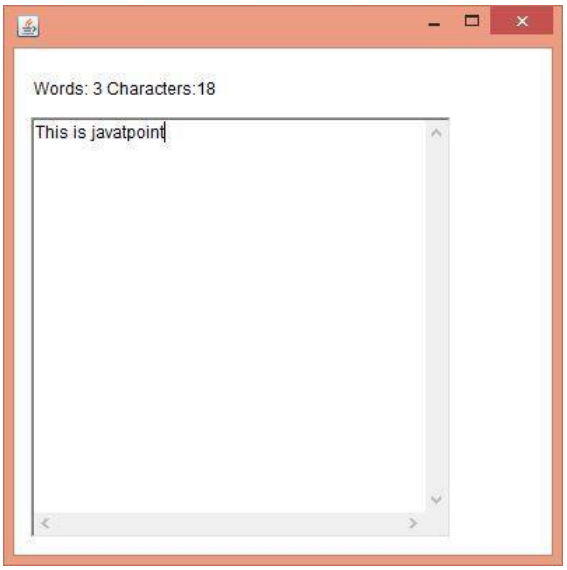
Java KeyListener

Example 2: Count Words & Characters

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class KeyListenerExample extends Frame implements KeyListener{
4.     Label l;
5.     TextArea area;
```

```
6.      KeyListenerExample(){
7.
8.          l=new Label();
9.          l.setBounds(20,50,200,20);
10.         area=new TextArea();
11.         area.setBounds(20,80,300, 300);
12.         area.addKeyListener(this);
13.
14.         add(l);add(area);
15.         setSize(400,400);
16.         setLayout(null);
17.         setVisible(true);
18.     }
19.     public void keyPressed(KeyEvent e) {}
20.     public void keyReleased(KeyEvent e) {
21.         String text=area.getText();
22.         String words[]=text.split("\\s");
23.         l.setText("Words: "+words.length+" Characters:"+text.length());
24.     }
25.     public void keyTyped(KeyEvent e) {}
26.
27.     public static void main(String[] args) {
28.         new KeyListenerExample();
29.     }
30. }
```

Output:



Java Adapter Classes

Java adapter classes *provide the default implementation of listener interfaces*. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it *saves code*.

The adapter classes are found in **java.awt.event** package. The Adapter classes with their corresponding listener interfaces are given below.

java.awt.event Adapter classes

Adapter class	Listener interface
WindowAdapter	WindowListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
HierarchyBoundsAdapter	HierarchyBoundsListener

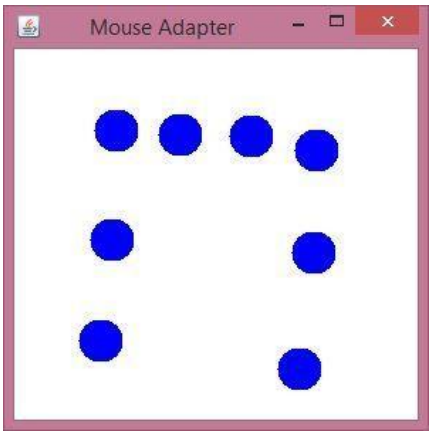
java.awt.dnd Adapter classes

Adapter class	Listener interface
DragSourceAdapter	DragSourceListener
DragTargetAdapter	DragTargetListener

Java MouseAdapter Example

```
1. import java.awt.*;
2. import java.awt.event.*;
3. public class MouseAdapterExample extends MouseAdapter{
4.     Frame f;
5.     MouseAdapterExample(){
6.         f=new Frame("Mouse Adapter");
7.         f.addMouseListener(this);
8.
9.         f.setSize(300,300);
10.        f.setLayout(null);
11.        f.setVisible(true);
12.    }
13.    public void mouseClicked(MouseEvent e) {
14.        Graphics g=f.getGraphics();
15.        g.setColor(Color.BLUE);
16.        g.fillOval(e.getX(),e.getY(),30,30);
17.    }
18.
19. public static void main(String[] args) {
20.    new MouseAdapterExample();
21. }
22. }
```

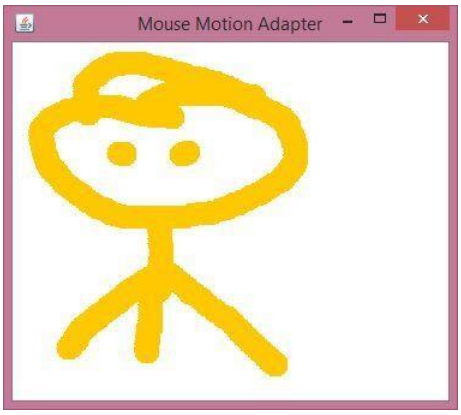
Output:



Java MouseMotionAdapter Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class MouseMotionAdapterExample extends MouseMotionAdapter{
4.     Frame f;
5.     MouseMotionAdapterExample(){
6.        f=new Frame("Mouse Motion Adapter");
7.        f.addMouseMotionListener(this);
8.
9.        f.setSize(300,300);
10.       f.setLayout(null);
11.       f.setVisible(true);
12.    }
13. public void mouseDragged(MouseEvent e) {
14.    Graphics g=f.getGraphics();
15.    g.setColor(Color.ORANGE);
16.    g.fillOval(e.getX(),e.getY(),20,20);
17. }
18. public static void main(String[] args) {
19.    new MouseMotionAdapterExample();
20. }
21. }
```

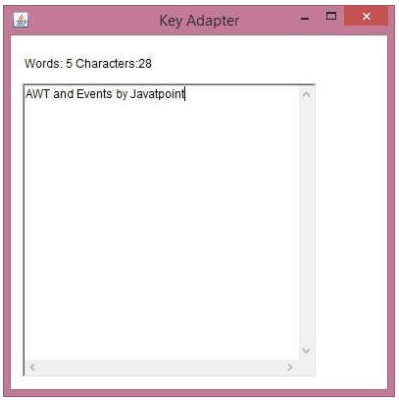
Output:



Java KeyAdapter Example

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  public class KeyAdapterExample extends KeyAdapter{
4.     Label l;
5.     TextArea area;
6.     Frame f;
7.     KeyAdapterExample(){
8.        f=new Frame("Key Adapter");
9.        l=new Label();
10.       l.setBounds(20,50,200,20);
11.       area=new TextArea();
12.       area.setBounds(20,80,300, 300);
13.       area.addKeyListener(this);
14.
15.       f.add(l);f.add(area);
16.       f.setSize(400,400);
17.       f.setLayout(null);
18.       f.setVisible(true);
19.    }
20.    public void keyReleased(KeyEvent e) {
21.       String text=area.getText();
22.       String words[]=text.split("\\s");
23.       l.setText("Words: "+words.length+" Characters:"+text.length());
24.    }
25.
26.    public static void main(String[] args) {
27.       new KeyAdapterExample();
28.    }
29. }
```

Output:


Key Adapter
Words: 5 Characters:28
AWT and Events by Javatpoint

UNIT-V

Java Layout Managers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:


```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class Border {
5.  JFrame f;
6.  Border(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("NORTH");;
10.     JButton b2=new JButton("SOUTH");;
11.     JButton b3=new JButton("EAST");;
12.     JButton b4=new JButton("WEST");;
13.     JButton b5=new JButton("CENTER");;
14.
15.     f.add(b1,BorderLayout.NORTH);
16.     f.add(b2,BorderLayout.SOUTH);
17.     f.add(b3,BorderLayout.EAST);
18.     f.add(b4,BorderLayout.WEST);
19.     f.add(b5,BorderLayout.CENTER);
```

```
20.
21.     f.setSize(300,300);
22.     f.setVisible(true);
23. }
24. public static void main(String[] args) {
25.     new Border();
26. }
27. }
```

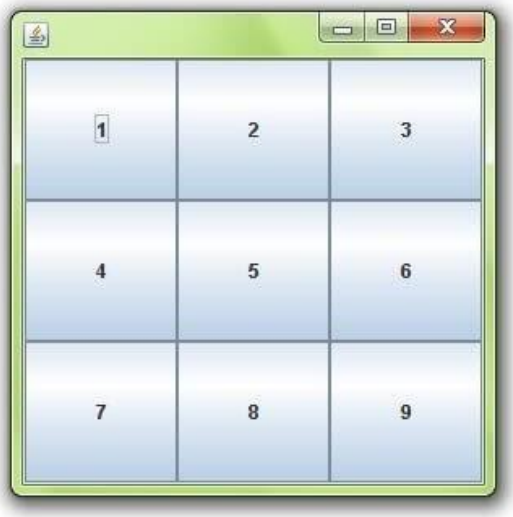
Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class




```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class MyGridLayout{
5. JFrame f;
```

```
6.  MyGridLayout(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("1");
10.     JButton b2=new JButton("2");
11.     JButton b3=new JButton("3");
12.     JButton b4=new JButton("4");
13.     JButton b5=new JButton("5");
14.         JButton b6=new JButton("6");
15.         JButton b7=new JButton("7");
16.     JButton b8=new JButton("8");
17.         JButton b9=new JButton("9");
18.
19.     f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
20.     f.add(b6);f.add(b7);f.add(b8);f.add(b9);
21.
22.     f.setLayout(new GridLayout(3,3));
23.     //setting grid layout of 3 rows and 3 columns
24.
25.     f.setSize(300,300);
26.     f.setVisible(true);
27. }
28. public static void main(String[] args) {
29.     new MyGridLayout();
30. }
31. }
```

Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.

2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class


```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class MyFlowLayout{
5.  JFrame f;
6.  MyFlowLayout(){
7.      f=new JFrame();
8.
9.      JButton b1=new JButton("1");
10.     JButton b2=new JButton("2");
11.     JButton b3=new JButton("3");
12.     JButton b4=new JButton("4");
13.     JButton b5=new JButton("5");
14.
15.     f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
16.
17.     f.setLayout(new FlowLayout(FlowLayout.RIGHT));
18.     //setting flow layout of right alignment
19.
20.     f.setSize(300,300);
21.     f.setVisible(true);
22. }
23. public static void main(String[] args) {
```

```
24.     new MyFlowLayout();
25. }
26. }
```

Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

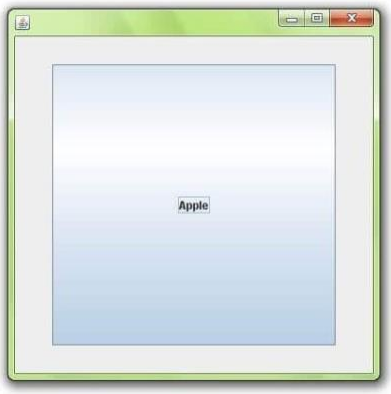
Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout class




```
1. import java.awt.*;
2. import java.awt.event.*;
```

```
3.
4. import javax.swing.*;
5.
6.  public class CardLayoutExample extends JFrame implements ActionListener{
7.  CardLayout card;
8.  JButton b1,b2,b3;
9.  Container c;
10.     CardLayoutExample(){
11.
12.         c=getContentPane();
13.         card=new CardLayout(40,30);
14. //create CardLayout object with 40 hor space and 30 ver space
15.         c.setLayout(card);
16.16.
17.         b1=new JButton("Apple");
18.         b2=new JButton("Boy");
19.         b3=new JButton("Cat");
20.         b1.addActionListener(this);
21.         b2.addActionListener(this);
22.         b3.addActionListener(this);
23.
24.         c.add("a",b1);c.add("b",b2);c.add("c",b3);
25.
26.     }
27.     public void actionPerformed(ActionEvent e) {
28.     card.next(c);
29.     }
30.
31.     public static void main(String[] args) {
32.         CardLayoutExample cl=new CardLayoutExample();
33.         cl.setSize(400,400);
34.         cl.setVisible(true);
35.         cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
36.     }
37. }
```

Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Fields

Modifier and Type	Field	Description
double[]	columnWeights	It is used to hold the overrides to the column weights.
int[]	columnWidths	It is used to hold the overrides to the column minimum width.
protected Hashtable<Component,GridBagConstraints>	comptable	It is used to maintains the association between a component and its gridbag constraints.
protected GridBagConstraints	defaultConstraints	It is used to hold a gridbag constraints instance containing the default values.
protected GridBagLayoutInfo	layoutInfo	It is used to hold the layout information for the gridbag.
protected static int	MAXGRIDSIZE	No longer in use just for backward compatibility
protected static int	MINSIZE	It is smallest grid that can be laid out by the grid bag layout.
protected static int	PREFERREDSIZE	It is preferred grid size that can be laid out by the grid bag layout.
int[]	rowHeights	It is used to hold the overrides to the row minimum heights.
double[]	rowWeights	It is used to hold the overrides to the row weights.

Useful Methods

Modifier and Type	Method	Description
Void	addLayoutComponent(Component comp, Object constraints)	It adds specified component to the layout, using the specified constraints object.
Void	addLayoutComponent(String name, Component comp)	It has no effect, since this layout manager does not use a per-component string.
protected void	adjustForGravity(GridBagConstraints constraints, Rectangle r)	It adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.
protected void	AdjustForGravity(GridBagConstraints constraints, Rectangle r)	This method is for backwards compatibility only
protected void	arrangeGrid(Container parent)	Lays out the grid.
protected void	ArrangeGrid(Container parent)	This method is obsolete and supplied for backwards compatibility
GridBagConstraints	getConstraints(Component comp)	It is for getting the constraints for the specified component.
Float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x axis.
Float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y axis.
int[][]	getLayoutDimensions()	It determines column widths and row heights for the layout grid.
protected GridBagLayoutInfo	getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
protected GridBagLayoutInfo	GetLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.

Point	getLayoutOrigin()	It determines the origin of the layout area, in the graphics coordinate space of the target container.
double[][]	getLayoutWeights()	It determines the weights of the layout grid's columns and rows.
protected Dimension	getMinSize(Container parent, GridBagLayoutInfo info)	It figures out the minimum size of the master based on the information from getLayoutInfo.
protected Dimension	GetMinSize(Container parent, GridBagLayoutInfo info)	This method is obsolete and supplied for backwards compatibility only

Example

```
1.  import java.awt.Button;
2.  import java.awt.GridBagConstraints;
3.  import java.awt.GridBagLayout;
4.
5.  import javax.swing.*;
6.  public class GridBagLayoutExample extends JFrame{
7.     public static void main(String[] args) {
8.        GridBagLayoutExample a = new GridBagLayoutExample(); 9.
          }
10.       public GridBagLayoutExample() {
11.    GridBagLayout grid = new GridBagLayout();
12.          GridBagConstraints gbc = new GridBagConstraints();
13.          setLayout(grid);
14.          setTitle("GridBag Layout Example");
15.          GridBagLayout layout = new GridBagLayout();
16.    this.setLayout(layout);
17.    gbc.fill = GridBagConstraints.HORIZONTAL;
18.    gbc.gridx = 0;
19.    gbc.gridy = 0;
20.    this.add(new Button("Button One"), gbc);
21.    gbc.gridx = 1;
22.    gbc.gridy = 0;
23.    this.add(new Button("Button two"), gbc);
24.    gbc.fill = GridBagConstraints.HORIZONTAL;
25.    gbc.ipady = 20;
26.    gbc.gridx = 0;
```

```
27.     gbc.gridy = 1;
28.     this.add(new Button("Button Three"), gbc);
29.     gbc.gridx = 1;
30.     gbc.gridy = 1;
31.     this.add(new Button("Button Four"), gbc);
32.     gbc.gridx = 0;
33.     gbc.gridy = 2;
34.     gbc.fill = GridBagConstraints.HORIZONTAL;
35.     gbc.gridwidth = 2;
36.     this.add(new Button("Button Five"), gbc);
37.     setSize(300, 300);
38.           setPreferredSize(getSize());
39.           setVisible(true);
40.           setDefaultCloseOperation(EXIT_ON_CLOSE);
41.
42.       }
43.
44. }
```

Output:

Applets

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

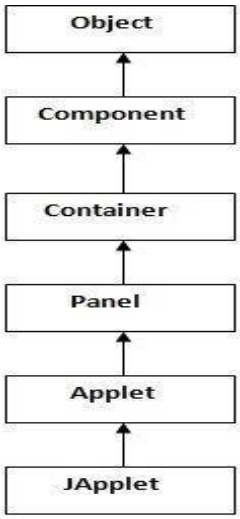
- o It works at client side so less response time.
- o Secured
- o It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- o Plugin is required at client browser to execute applet.

Lifecycle of Java Applet Hierarchy of Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle method for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
1. //First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("welcome",150,150);
}
}
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
1. //First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{
public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}
}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

c:\>javac First.java
c:\>appletviewer First.java

Difference between Applet and Application programming

	Java Applet	Java Application
User graphics	Inherently graphical	Optional
Memory requirements	Java application requirements plus web browser requirements	Minimal java application requirements
Distribution	Linked via HTML and transported via HTTP	Loaded from the file system or by a custom class loading process
Environmental input	Browser client location and size; parameters embedded in the host HTML document	command-line parameters
Method expected by the virtual Machine	init- initialization method start-startup method stop pause/ deactive method destroy-termination method paint-drawing method	Main - startup method
Typical applications	public-access order-entry systems for the web, online multimedia persentations, web page animation	Network server, multimedia kiosks, developer tools, appliance and consumer electronics control and navigation.

Parameter in Applet

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter(). Syntax:

1. **public** String getParameter(String parameterName)

Example of using parameter in Applet:

```
1. import java.applet.Applet;
2. import java.awt.Graphics;
3. public class UseParam extends Applet
4. {
5. public void paint(Graphics g)
```

```
6. {
7. String str=getParameter("msg");
8. g.drawString(str,50, 50);
9. } }
```

myapplet.html

```
1. <html>
2. <body>
3. <applet code="UseParam.class" width="300" height="300">
4. <param name="msg" value="Welcome to applet">
5. </applet>
6. </body>
7. </html>
```

File name :file.txt Path: file.txt
Absolute path:C:\Users\akki\IdeaProjects\codewriting\src\file.txt Parent:null
Exists :true
Is writeable:true Is readabletrue
Is a directory:false File Size in bytes 20

Connceting to DB

What is JDBCDriver?

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server. For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

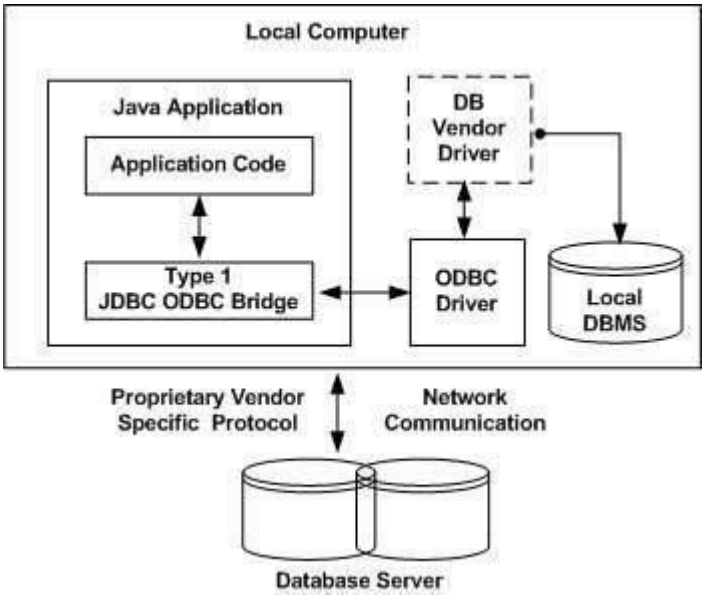
The *Java.sql* package that ships with JDK, contains various classes with their behaviours defined and their actual implementaions are done in third-party drivers. Third party vendors implements the *java.sql.Driver* interface in their database driver.

JDBC Drivers Types

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below

Type 1: JDBC-ODBCBridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.
When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.



The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API

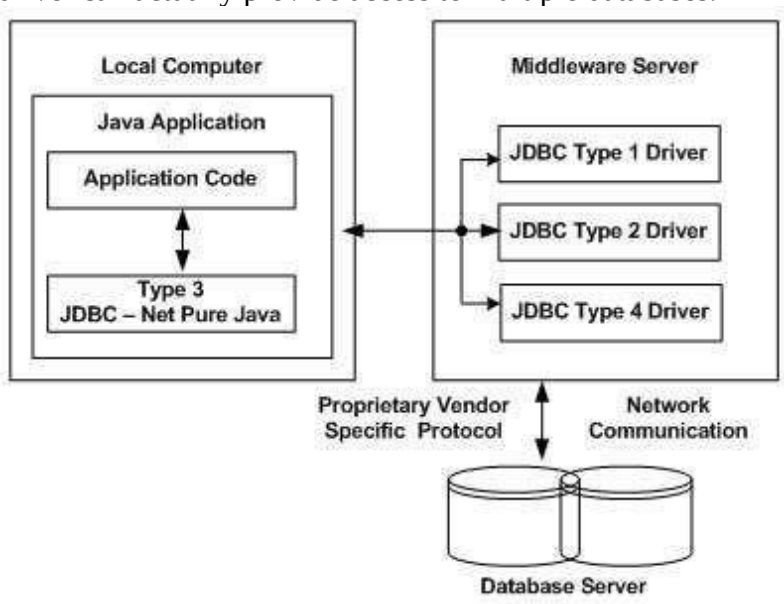
In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

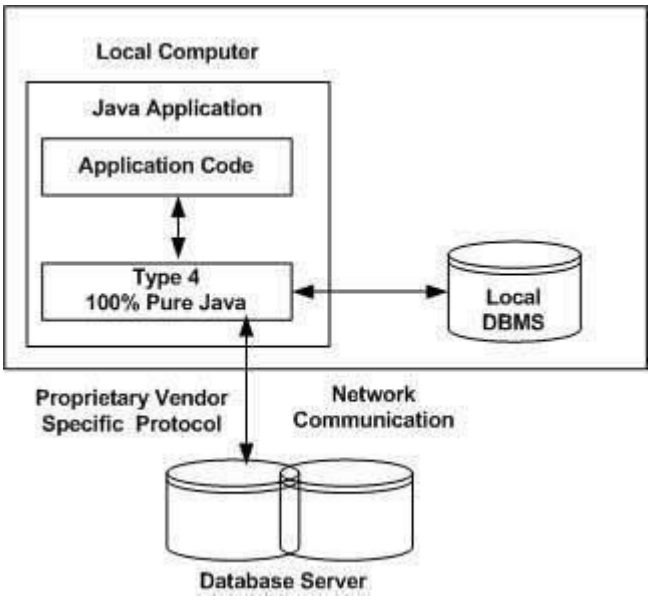


You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.
This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.
MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.



Which Driver should be Used?

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4. If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

Example to connect to the mysql database in java

For connecting java application with the mysql database, you need to follow few steps to perform database connectivity. In this example we are using MySql as the database. So we need to know following information for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**. 2.**Connection**

URL: The connection URL for the mysql database is
jdbc:mysql://localhost:3306/sonoo where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, you need to replace the sonoo with your database name.

3. **Username:** The default username for the mysql database is **root**.

4. **Password:** Password is given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database sonoo;

2. use sonoo;

3. create table emp(id **int**(10),name varchar(40),age **int**(3)); **Example to Connect Java Application with mysql database**

In this example, sonoo is the database name, root is the username and password.

```
import java.sql.*;
class MysqlCon
{
public static void main(String args[])
{
      Try
{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection( "jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)); con.close();
}
catch(Exception e){ System.out.println(e);
}
}
}
```

The above example will fetch all the records of emp table.

VIDYA JYOTHI INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

CLASS: II year II SEM Course:JP Batch: 2017-21

S.No	Reg.No	MID-1 Threshold 60%								MID-2 Threshold 60%								Threshold 60% (45M)
				PART-B								PART-B						
		Assignment1 (5M)	Theory (20M)	M1- Q1 (2M) CO1	M1-Q2 (2M) CO2	M1- Q3 (2M) CO3	M1- Q4 (5M) CO1	M1-Q5 (5M) CO2	M1-Q6 (4M) CO3	Assignment2 (5M)	Theory (20M)	M2- Q1 (2M) CO3	M2-Q2 (2M) CO4	M2- Q3 (2M) CO5	M2-Q4 (4M) CO3	M2-Q5 (5M) CO4	M2-Q6 (5M) CO5	End Exam (75M)
1	16911A1255	5	9	2	1	2	1	1	2	5	8	1	1	2	2	1	1	16
2	17911A1201	5	16	1	2	1	4	4	4	5	15	1	1	1	4	4	4	60
3	17911A1202	5	13	2	2	2	2	2	3	5	12	2	2	1	3	2	2	49
4	17911A1203	5	17	2	1	1	5	4	4	5	16	2	1	1	3	5	4	45
5	17911A1205	5	14	2	2	1	3	3	3	5	13	2	2	1	3	2	3	47
6	17911A1206	5	17	2	2	1	4	4	4	5	16	2	2	1	4	4	3	47
7	17911A1208	5	19	1	2	2	5	5	4	5	18	1	2	2	4	4	5	53
8	17911A1209	5	9	2	1	1	2	2	1	5	8	2	1	1	0	2	2	50
9	17911A1211	5	12	1	1	1	4	3	2	5	11	1	1	0	2	4	3	46
10	17911A1212	5	15	2	1	2	5	3	2	5	14	2	0	2	2	5	3	48
11	17911A1213	5	11	1	1	1	3	2	3	5	10	0	1	1	3	3	2	26
12	17911A1214	5	18	2	2	1	5	4	4	5	17	2	1	1	4	5	4	54
13	17911A1215	5	14	1	1	1	3	4	4	5	13	1	1	0	4	3	4	47
14	17911A1216	5	8	1	1	1	1	1	3	5	7	1	1	1	2	1	1	50
15	17911A1218	5	14	2	1	1	4	2	4	5	13	2	1	1	4	3	2	36
16	17911A1219	5	8	2	1	1	1	1	2	5	7	2	1	1	2	1	0	15
17	17911A1220	5	16	2	2	2	4	3	3	5	15	2	2	2	3	3	3	48
18	17911A1221	5	17	2	2	2	4	3	4	5	16	2	2	2	3	4	3	51
19	17911A1222	5	5	1	1	1	1	0	1	5	4	1	1	0	1	1	0	10
20	17911A1223	5	11	1	1	1	2	3	3	5	10	1	0	1	3	2	3	48
21	17911A1224	5	11	1	2	1	3	2	2	5	10	0	2	1	2	3	2	49
22	17911A1225	5	16	1	2	1	4	4	4	5	15	1	1	1	4	4	4	47
23	17911A1226	5	11	1	2	2	3	1	2	5	10	1	2	1	2	3	1	29
24	17911A1227	5	18	2	2	2	4	4	4	5	17	2	2	2	3	4	4	52
25	17911A1228	5	20	2	2	2	5	5	4	5	19	2	2	2	4	4	5	51

26	17911A1229	5	11	1	1	2	2	2	3	5	10	1	1	2	3	2	1	26
27	17911A1230	5	10	1	1	1	3	2	2	5	9	1	1	1	2	2	2	48
28	17911A1231	5	15	2	2	2	2	4	3	5	14	2	2	2	2	2	4	44
29	17911A1232	5	15	1	1	1	5	3	4	5	14	1	1	0	4	5	3	47
30	17911A1233	5	9	1	1	1	2	2	2	5	8	1	0	1	2	2	2	26
31	17911A1234	5	11	1	1	1	3	2	3	5	10	0	1	1	3	3	2	50
32	17911A1235	5	13	1	2	1	3	4	2	5	12	1	1	1	2	3	4	45
33	17911A1236	4	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	3
34	17911A1237	5	17	1	0	2	5	5	4	5	16	1	0	1	4	5	5	49
35	17911A1238	5	13	2	1	1	4	3	2	5	12	2	1	1	1	4	3	51
36	17911A1239	5	7	0	1	0	3	1	2	5	6	0	1	0	2	2	1	29
37	17911A1240	5	12	1	1	1	4	2	3	5	11	1	1	1	3	4	1	46
38	17911A1241	5	20	2	2	2	5	5	4	5	19	2	2	2	4	4	5	53
39	17911A1242	5	12	1	2	3	2	2	2	5	11	1	2	3	1	2	2	49
40	17911A1243	5	7	1	1	1	1	1	2	5	6	1	1	0	2	1	1	47
41	17911A1244	5	15	2	1	1	5	4	2	5	14	2	0	1	2	5	4	26
42	17911A1245	5	10	1	2	1	2	1	3	5	9	0	2	1	3	2	1	28
43	17911A1246	5	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0
44	17911A1248	5	15	2	1	2	4	3	3	5	14	2	1	2	3	4	2	54
45	17911A1249	5	12	1	2	1	3	3	2	5	11	1	2	1	2	2	3	51
46	17911A1250	5	11	2	1	1	1	3	3	5	10	2	1	1	2	1	3	48
47	17911A1251	5	11	1	1	1	2	2	4	5	10	1	1	0	4	2	2	47
48	17911A1253	5	13	2	2	1	1	3	4	5	12	2	1	1	4	1	3	39
49	17911A1254	5	16	2	1	1	4	4	4	5	15	1	1	1	4	4	4	44
50	17911A1255	5	17	1	1	2	4	5	4	5	16	1	0	2	4	4	5	47
51	17911A1256	5	15	1	2	1	4	3	4	5	14	1	2	0	4	4	3	52
52	17911A1257	5	16	1	2	1	4	4	4	5	15	1	2	1	3	4	4	47
53	17911A1258	5	15	2	2	2	3	3	3	5	14	2	2	2	3	2	3	60
54	17911A1259	5	11	2	2	0	4	1	2	5	10	2	2	0	2	4	0	47
Average marks		5.0	12.7	1.4	1.4	1.3	3.1	2.7	2.9	4.9	11.8	1.3	1.2	1.1	2.7	2.9	2.6	42.2
No of students attemped		54	54	54	54	54	54	54	54	54	54	54	54	54	54	54	54	54
%of students scored 60% and		100.00	61.11	94.44	94.44	92.59	66.67	57.41	92.59	100.00	53.70	87.04	85.19	79.63	94.44	96.30	0.00	70.37
CO ATTAINMENT LEVEL		3.0	1.0	3.0	3.0	3.0	1.0	0.0	3.0	3.0	0.0	3.0	3.0	2.0	3.0	3.0	0.0	3.0

ASSESSMENT OF COs FOR THE COURSE							
COs	Method	value	CO Attainment	Assignments	CO Attainment (Internal - Theory)	CO Attainment (End Exam)	Overall CO Attainment
CO1	MI Q1	3.0	2.0	3.0	2.1	3.00	2.89
	MI Q5	1.0					
CO2	MI Q2	3.0	1.5				
	MI Q6	0.0					
CO3	MI Q3	3.0	3.0				
	MI Q7	3.0					
	M2 Q1	3.0					
	M2 Q4	3.0					
CO4	M2 Q2	3.0	3.0				
	M2 Q5	3.0					
CO5	M2 Q3	2.0	1.0				
	M2						

Course End Survey Form

JAVA PROGRAMMING

1. Are you able to understand OOP concepts to apply basic Java constructs?46 responses

Slight 2
Moderate 3
Substantial 41

2. Are you able to analyze different forms of inheritance and handle different kinds of file I/O?46 responses

Slight 3
Moderate 3
Substantial 40

3. Are you able to evaluate the usage of Exception Handling and Multithreading in complex Java programs?46 responses

Slight 3
Moderate 1
Substantial 42

4. Are you able to contrast different GUI layouts and design GUI applications?46 responses

Slight 1
Moderate 4
Substantial 41

5. Are you able to construct a full-fledged Java GUI application, and Applet with database connectivity?46 responses

Slight 3
Moderate 4
Substantial 39