Lecture Note

on

Programming with Java (MCA401)

Prepared by

Dr. Suvendra Kumar Jayasingh

Assistant Professor

Department of MCA

IMIT, Cuttack

E Mail: sjayasingh@gmail.com, M - 9861016676

SYLLABUS

MCA 401 Programming with Java

Module 1 (10 Hours)
Features of Java, Data types, operators & expressions, control structures, arrays, Classes, objects & methods, constructors, garbage collection, access qualifiers, string handling – string operations, character extraction, string comparison, searching and modifying strings, String Buffer, packages and interfaces, Wrapper classes.

Module 2 (10 Hours)
Inheritance: single and multilevel inheritance, method overriding, abstract class, use of super and final keywords. Exception Handling: Exception types, uncaught exceptions, multiple catch clauses, nested try statements, built-in exceptions, creating your own exceptions. Multithreading: Java thread model, creating multiple threads, thread priorities, synchronization, interthread communication, suspending, resuming and stopping threads.

Module 3 (10 Hours)
Applets: Local & Remote Applets, Applet Architecture, Passing Parameters to Applets, Applet Graphics, Adapter Class. I/O Streams: Console I/O – reading console input, writing console output, Files I/O – Byte Streams, Character Streams, Collection Interfaces & Classes, Delegation Event Model

Module 4 (10 Hours)
AWT Classes: Window fundamentals, working with graphics, working with color & fonts. AWT controls, layout managers & working with menus, JFrames. Swing Classes, Java Beans, Servlet classes & Life Cycle.

Module 5 (6 Hours)
(as per choice of faculty)
Portion covered can be tested through Internal evaluation only not to be included in University examination)

Text Books:

1. Herbert Schildt, The Complete Reference Java 2, Fourth Edition, Tata McGraw Hill-2001
2. Liang Y.Daniel, Introduction to Java Programming (7th Edition), 2009, Pearson Education.

Reference Books:

1. Steven Holzner, Java 1.2, BPB-1998
2. E. Balaguruswami, Programming with Java - Second Edition, Tata McGraw Hill-1998.
3. Mughal K.A., Rasmussen R.W., A Programmer's Guide to Java Certification, Addison-Wesley, 2000

CONTENTS

History of Java

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". Java was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given significant points that describe 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) Initially designed for small, embedded systems in electronic appliances like set-top boxes.

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

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

5) Why Oak? Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

6) In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.

Flavours of Java

Java is available in market in three flavours.

1. J2SE – Java 2 Standard Edition – Core Java – For Desktops
2. J2EE – Java 2 Enterprise Edition – Advanced Java – For enterprises

3. J2ME – Java 2 Micro Edition – For Mobiles

J2EE comprises of the following frameworks.

1. JDBC –Java Database Connectivity
2. RMI – Remote Method Invocation
3. JSP – Java Server Pages
4. Servlet
5. EJB – Enhanced Java Beans
6. JTA – Java Transaction API
7. JMS – Java Message Service
8. JC – Java Connection
9. JCA – Java Card API
10. JMA – Java Mail API
11. JNDI – Java Naming and Directory Interface
12. JIDL – Java Integrating Development Language
13. JSTL – Java Standard Tag Library
14. JSF – Java Server Faces
15. Swing
16. JAAS – Java Authorization and Authentication Service
17. Struts

Difference between C++ and Java

Sl. No.	C++	Java
1	It is not pure OOP	It is pure OOP
2	It supports pointer	It does not support pointer
3	It supports multiple inheritance	It does not support multiple inheritance
4	It uses 8 bit representation (ASCII)	It uses 16 bit representation (UNICODE)
5	Memory release is manual	Garbage collection is automatic
6	It is platform dependent	It is platform independent
7	There id garbage in C++	There is nothing garbage in Java
8	It supports static binding	It supports dynamic binding

9	It can not be used for development of web application	It can be used for development of web application
10	It is a compiled language	It is both compiled and interpreted
11	It supports global data and method	There is nothing global in java
12	It can have only one main method	It can have many main methods
13	It supports header files	It does not support header files
14	Faster	Slower
15	Less secure	More secure
16	It supports operator overloading	It does not support operator overloading

Writing a program in Java:-

- The source editor in java can be written in any editor.
- It contains one or multiple no of classes.

Nomenclatre of Classes and Methods in Java

Class in Java:-

Normally Java contains 2 types of classes such as

- Predefined class
- User defined class

Predefined class:- The 1st character of every keyword in the predefined class is capital.

Example – System

DataInputStream

ArithmeticException

ActionEvent

String

ArrayIndexOutOfBoundsException

MouseEvent

Applet

Thread

User defined class:- There is no rule for this nomenclature.

Method name in Java:-

- Predefined method
- User defined method

Predefined method:- The 1st character is small and in subsequent words, the 1st character is capital.

Example : setActionCommand()

addActionListener()

itemStateChanged()

getIconHeight()

print()

println()

User defined method:- There is no rule for this nomenclature.

Naming and Saving in Java program:-

- Must be saved with java extensions.
- The primary name of the program must be named as the class is declared public.
- Primary name can be any name of the class is not public.
- While saving in notepad, sure that you choose "all files" options from file type.

Compiling and Running a Java:-

Compile:- D:\> javac filename.java

Running a java program:-

Run:- D:\> java filename

Program: 1

Write a program(Wap) in Java to print "Hi…Welcome to Java World".

```
class a
{
    public static void main(String args[])
    {
        System.out.println("Hi…Welcome to Java World");
    }
}
```

Program: 2

Wap in Java to perform addition of two integers.

```
Class a
Public static void main(String args[])
{
int n1=5, n2=10;, n3;
n3=na+n2;
System.out.println(" The sum of two integers is :"+n3);
}
}
```

Program: 3

Wap in Java to display prime numbers between 1 and 200.

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

```
{
        int i=0;
        int n=0;
        String prime=" ";
        for(i=1;i<=200;i++)
        {
                int c=0;
                for(n=i;n>=1;n--)
                {
                        if(i%n==0)
                        {
                                c=c+1;
                        }
                }
                if(c==2)
                {
                        prime=prime+i+" ";
                }
        }
        System.out.println("Prime Numbers from 1 to 200: ");
        System.out.println(prime);
}
}
```

Program: 4

Wap in Java to print Floyd's Triangle.

```
class floyd
{
	public static void main(String s[])
	{
		int i,j,n=1;
		for(i=1;i<=4;i++)
		{
			for(j=1;j<=i;j++)
			{
				System.out.print(n);
				n++;
			}
			System.out.println(" ");
		}
	}
}
```

Program: 5

Wap in Java to print Pascal's Triangle.

```
class pascal
{
public static void main(String s[])
{
```

```
        int r=5,i,j,k,n=1;
        for(i=0;i<r;i++)
        {
                for(k=r;k>i;k--)
                {
                        System.out.print(" ");
                }
                n=1;
                for(j=0;j<=i;j++)
                {
                        System.out.print(n+" ");
                        n=n*(i-j)/(j+1);
                }
        System.out.println();
}
}
}
```

Program: 6

Wap in Java to display Fibonacci series.

```
class abc
{
        void fibo()
        {
                int a,b,c,i;
```

```
            a=0;b=1;c=0;
            for(i=0;i<7;i++)
            {
                  System.out.println(a);
                  c=a+b;
                  a=b;
                  b=c;
            }
      }
}
class fibos
{
      public static void main(String s[])
      {
            abc a=new abc();
            a.fibo();
      }
}
```

Program: 7

Wap in java to print pattern of *.

```
class star
{
      public static void main(String s[])
      {
```

```
        int i,j;
        for(i=1;i<=6;i++)
        {
            for(j=1;j<i;j++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Program: 8

Wap in Java to print another pattern of *.

```
class StarT
{
    public static void main(String s[])
    {
        int i,j,k;
        for(i=1;i<=5;i++)
        {
            for(j=4;j<=i;j--)
            {
                System.out.print(" ");
            }
```

```
                for(k=1;k<=(2*i-1);k++)
                {
                        System.out.print("*");
                }
                System.out.println("");
        }
    }
}
```

Program: 9

Wap in Java to show use of Arithmetic Operator.

```
class ArithmeticOperator
{
    public static void main(String s[])
    {
        int i=1+1;
        int n=i*3;
        int m=n/4;
        int p=m-i;
        System.out.println("i= "+i+" n= "+n+" m= "+m+" p= "+p);
    }
}
```

Program: 10

Wap in Java to show use of Assignment Operator.

```
class AssignmentOperator
```

```
{
    public static void main(String s[])
    {
        int x=12,y=13,z=16;
        System.out.println("Assignment Value");
        x+=2;
        y-=2;
        z*=2;
        System.out.println("x= "+x+" y= "+y+" z= "+z);
    }
}
```

Program: 11

Wap in Java to show use of Conditional Operator.

```
class ConditionalOperator
{
    public static void main(String s[])
    {
        int a=10;
        int b=20;
        int x=(a<b)?a:b;
        int y=(a>b)?a:b;
        System.out.println("Number= "+x);
        System.out.println("Number= "+y);
    }
```

```
}
```

Program: 12

Wap in Java to show use of Bitwise Operator.

```
class BitwiseOperator
{
	public static void main(String s[])
	{
		int a=6,b=5;
		int c=a&b;
		System.out.println("a= "+a+" b= "+b+" c= "+c);
		int d=a/b;
		System.out.println("d= "+d);
	}
}
```

Program: 13

Wap in Java to show use of Shift Operator.

```
class ShiftOperator
{
	public static void main(String s[])
	{
		int i=20;
		int result=i>>2;
		System.out.println("Shift Right");
		System.out.println("result= "+result);
```

```
    }
}
```

Program: 14

Wap in Java to show use of instanceof Operator.

```
class Test
{
    public static void main(String s[])
    {
        Test t=new Test();
        System.out.println(t instanceof Test);
    }
}
```

Classes in Java

1. It is created by using the class keyword.
2. It contains 2 types of members
    a. Data Member
    b. Method Member
3. Data Member follows either of the three initializations
    a. Auto Initialization
    b. Inline Initialization
    c. Complete Initialization
4. For method members
    a. Return type is mandatory
    b. No scope resolution operator
5. Accessibility of a member out of the classis controlled by access specifiers.
6. Java supports 4 types of access specifiers.
    a. private
    b. public
    c. protected
    d. package
7. Default access specifier is package
8. Java does not support group access specifier.
9. Requires and ensures that every class has at least one constructor.
10. If a class does not have a constructor, compiler creates and supplies a blank constructor/ default constructor/ zero argument constructor.
11. Destructor of a class is identified by the name finalize().
12. There is no garbage value or junk value in Java.
13. A class can have multiple constructors which is constructor overloading.

Auto Intialization : If we do not assign value to the data member, the compiler supplies default constructor to initialize the data members with default values. This is known as auto initialization.

Inline Initilization : Java allows to assign values to data members at the time of declaration. This is known as inline initialization.

Complete Initialization: If we initialize some data members and do not initialize the others, complier completes the initialization by assigning default values to the rest of the un initialized data members.

Program: 15

Wap in Java to show use of Auto initialization.

```
class aa
{
	int a1,a2;
	void show()
	{
		System.out.println(a1);
		System.out.println(a2);
	}
}
class auto
{
	public static void main(String s[])
	{
		aa oa=new aa();
		oa.show();
	}
}
```

Program: 16

Wap in Java program to show use of inline initialization.

```
class aa
{
	int a1=5,a2=20;
	void show()
	{
		System.out.println(a1);
		System.out.println(a2);
	}
}
class inline
{
	public static void main(String s[])
	{
		aa oa=new aa();
		oa.show();
	}
}
```

Program: 17

Wap in Java to show use of complete initialization.

```
class aa
{
	int a1=5,a2;
```

```
    void show()
    {
        System.out.println(a1);
        System.out.println(a2);
    }
}
class complete
{
    public static void main(String s[])
    {
        aa oa=new aa();
        oa.show();
    }
}
```

Program: 18

Wap in Java to use local variables and data members.

```
class hello
{
    String s="Data Member";
    void display()
    {
        System.out.println(s);
    }
    public static void main(String args[])
```

```
        {
                String x="Local Variable";
                System.out.println(x);
                hello oa=new hello();
                oa.display();
        }
}
```

Creating Object in Java

1. Objects are created by using the 'new' operator
2. New does three things.
    a. It reads the class proto type.
    b. It allocates memory to data members.
    c. It returns the allocated address.
3. The address of the object is stored ina variable of the same class type.
4. While creating an object, we must identify the constructor to be executed for member initialization.
5. Multiple objects of a class can be created.
6. Each object has its own copy of data members, whereas all the objects will share a single copy of method members.
7. Memebrs of an object are accessed as object name. member name.

Program: 19

Wap in Java to create an object.

```
class a
{
        void show()
        {
                System.out.println("Hi…I am Sagar");
        }
}
class obj
{
        public static void main(String s[])
        {
```

```
        a oa=new oa();
        oa.show();
    }
}
```

Program: 20

Wap in Java for data hiding.

```
class test
{
    private String name;
    private int age;
    public String getName()
    {
        return name;
    }
    public int age()
    {
        return age;
    }
    public void setName(String x)
    {
        name=x;
    }
    public void setAge(int y)
```

```
        {
                age=y;
        }
}
class DataHiding
{
        public static void main(String s[])
        {
    test ob=new test();
    ob.setName("XYZ");
    ob.setAge(21);
    System.out.println("Name= "+ob.getName());
    System.out.println("Age= "+ob.age());
 }
}
```

Program: 21

Wap in Java for Polymorphism.

```
class animal
{
        void sound()
        {
                System.out.println("Animal is making a sound");
        }
}
```

```
class Horse
{
	void sound()
	{
		System.out.println("Neigh");
	}
	public static void main(String s[])
	{
		Horse obj=new Horse();
		obj.sound();
	}
}
```

Program: 22

Wap in Java for constructor overloading.

```
class emp
{
	int age;
	String nm;
	emp()
	{
		age=75;
		nm="XYZ";
	}
	emp(int age, String nm)
```

```
	{
		this.age=age;
		this.nm=nm;
	}
	void disp()
	{
		System.out.println("Name= "+nm+" Age= "+age);
	}
}
class demo
{
	public static void main(String s[])
	{
		emp d1=new emp();
		emp d2=new emp(70,"ABC");
		d1.disp();
		d2.disp();
	}
}
```

Program: 23

Wap in Java for function overloading.

```
class test
{
	void disp(char c)
```

```
        {
                System.out.println(c);
        }
        void disp(char c,int n)
        {
                System.out.println(c +" "+ n);
        }
}
class dispovr
{
        public static void main(String s[])
        {
                test t=new test();
                t.disp('A');
                t.disp('A',10);
        }
}
```

Program: 24

Wap in Java for type casting.

```
class atest
{
        public static void main(String s[])
        {
                double d=100.04;
```

```
            long l=(long)d;
            int i=(int)l;
            System.out.println("Double Value= "+d);
            System.out.println("Long Value= "+l);
            System.out.println("Int Value= "+i);
        }
}
```

Path Setting

Path Setting-

- Open Command Prompt
- Copy the path of bin folder within java
- Write in Command Prompt: set path= %path%;C:\Program Files\Java\jdk1.8.0_65\bin;

Path Setting(Permanently)-

- Go to My computer ->properties
- Advanced tab
- Environment variable
- New tab of your variable in **path** tab
- Write path in variable name
- Write path of bin folder of java in variable value
- Then click on OK

Path setting through Batch File-

set path=%path%;C:\Program Files\Java\jdk1.8.0_65\bin;

Save as: **filename.bat**

Run in command prompt: **filename**

Java Garbage Collection

1. In java, garbage means unreferenced objects.
2. Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
3. 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

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

How can an object be unreferenced?

There are many ways:

1. By nulling the reference
2. By assigning a reference to another
3. By anonymous object etc.

Java Garbage Collection Scenario

1) By nulling a reference:

```
Employee e=new Employee();

e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();

Employee e2=new Employee();

e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By anonymous object:

```
new Employee();
```

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

protected void finalize(){}

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

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(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Program: 25

WAP to show 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();
 }
```

}

Output:

object is garbage collected

object is garbage collected

Wrapper Classes

Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.

The table below shows the primitive type and the equivalent wrapper class:

Primitive Data Type	**Wrapper Class**
Byte	Byte
Short	Short
Int	Integer
Long	Long
Float	Float
Boolean	Boolean
Double	Double
Char	Character

Sometimes we must use wrapper classes, for example when working with Collection objects, such as ArrayList, where primitive types cannot be used (the list can only store objects):

Example

ArrayList<int> myNumbers = new ArrayList<int>(); // Invalid

ArrayList<Integer> myNumbers = new ArrayList<Integer>(); // Valid

Creating Wrapper Objects

To create a wrapper object, use the wrapper class instead of the primitive type. To get the value, you can just print the object:

Program: 26

WAP to explaim wrapper class.

```
public class MyClass {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

Program: 27

Java program to convert primitive into objects

```
//Autoboxing example of int to Integer
public class WrapperExample1{
public static void main(String args[]){
```

```
//Converting int into Integer

int a=20;

Integer i=Integer.valueOf(a);//converting int into Integer explicitly

Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally



System.out.println(a+" "+i+" "+j);

}}
```

Output:

20 20 20

this

1. It is a system defined reference.
2. It is created and maintained by system.
3. It holds the address of the currently active object.
4. Java compiler will modify member access statements in a method by attaching the word this.member_name.
5. It is used to distinguish between local variable and data member if they are having the same name.
6. Data members are initialized by constructors, but local variables must be explicitly initialized by programmer before use.

this()

1. It is used as a method call.
2. Can only be used inside a constructor of same class type.
3. It must be the first line of a constructor

Program: 28

Wap in Java to use this as a reference.

```
class a
{
    int a,b;
    a(int x)
    {
        a=x;
    }
    void show()
    {
        int a=15;
```

```
            System.out.println(a);
            System.out.println(b);
            System.out.println(this.a);
        }
}
class rthis
{
        public static void main(String s[])
        {
            a oa=new a(5);
            oa.show();
        }
}
```

Program: 29

Wap in Java to use this() as a method.

```
class b
{
        int a,b;
        b(int x)
        {
            a=x;
        }
        b(int x, int y)
        {
```

```
        this(x);
        b=y;
    }
    void show()
    {
        System.out.println(a);
        System.out.println(b);
    }
}
class mthis
{
    public static void main(String s[])
    {
        b ob=new b(5);
        ob.show();
        b ob1=new b(7,14);
        ob1.show();
    }
}
```

Inheritance in java

1. It allows a class in java to acquire the members of another class.
2. It establishes a parent child relationship.
3. It is established by the “extends” keyword.
4. Child class uses the inherited members as if they are its own members.
5. Parent class decides which member can be inherited by using access specifier.
6. Child class can not restrict the scope of inherited members unlike C++.
7. Whenever an object of child class is created, automatically an object of parent class is created.
8. Child class gets to access a system defined reference by the name “super”.
9. “super” is a system defined reference which holds the address of associated parent object.
10. If the child class has a member exactly by the same name as a parent class member, the parent class member is suppressed within the child class.
11. Compiler automatically inserts the line “super()” as the first line of each child class constructor to ensure initialization of parent class data members.
12. Super has 3 uses
    a. as an object reference (super.a, super.show())
    b. as a method call (super(), super(5), super(5,10))
    c. to access suppressed parent class members
13. The parent class reference variable can hold the address of a child class object, but the reverse is not true.
14. Using a parent class reference, which is holding the child class address, we can access only those members which are inherited by the child.

Program: 30

Write a sample inheritance program in Java.

```
class inherit
{
    int a=5,b=6;
    void show()
```

```
        {
                System.out.println("Hi...");
        }
}
class init extends inherit
{
        int x=10,y=12;
        void display()
        {
                System.out.println("Bye!!!");
                System.out.println(x);
                System.out.println(a);
        }
}
class inheritance
{
        public static void main(String s[])
        {
                init ob=new init();
                ob.display();
        }
}
```

Program: 31

Wap in Java to use super as a reference.

```
class ab
{
	int a1=5;
}
class ba extends ab
{
	int b1,b2;
	int a1;
	void display()
	{
		System.out.println(a1);
		System.out.println(super.a1);
	}
}
class xyz
{
	public static void main(String s[])
	{
		ba b=new ba();
		b.display();
	}
}
```

Program: 32

Wap in Java to use super() as a method.

```
class a
{
        int a1=5;
}
class b extends a
{
        int b1,b2;
        b(int x)
        {
                super();
                b1=x;
                b2=2*x;
        }
        void display()
        {
                System.out.println(b1);
                System.out.println(b2);
        }
}
class smethod
{
        public static void main(String s[])
```

```
        {
                b ob=new b(10);
                ob.display();
        }
}
```

Program: 33

Wap in Java for method overriding.

```
class Company
{
        void address()
        {
                System.out.println("Address of Company");
        }
}
class eBay extends Company
{
        void address()
        {
                super.address();
                System.out.println("Address of eBay");
        }
}
class sample
{
```

```
    public static void main(String s[])
    {
        eBay a=new eBay();
        a.address();
    }
}
```

Package

1. It acts and behaves like a folder in operating system.
2. It is used to
    a. Resolve name conflicts across classes
    b. Keep classes organized
3. Are container of classes.
4. A class identified as a member of a package , it must be accessed as packagename.classname **for all purpose**.
5. All purpose refers to
    a. Inheritance
    b. Object creation
    c. Executing the program by using class name
6. A package can contain other packages called sub packages.
7. To adentify a class as a member of a package, we must
    a. Declare the line
    package package_name
    as the first line of source code.
    b. After compilation, store the .class in a folder by the same name as package name.
8. To access a class out of the package, it must be declared public.
9. To access a member of a class out of the package, the class and the member are to be declared public.
10. In case of packages, relative referencing is not permitted.
11. If the java program with package concept is compiled as follows, then the compiler will create a folder by the same name as that of the package name and stores the .class file in that folder.
    a. D:\> javac –d . x.java
    b. D:\> javac –d .. x,java
    c. D:\> javac –d c:\ x.java

Subpackages

1. Sub packages are independent packages.

2. They are created and stored using the same rules followed in case of package.

Program: 34

Wap in Java program to use package.

```
package p1;
class first
{
	public static void main(String s[])
	{
		int x=10, y=20;
		System.out.println(x);
		System.out.println(y);
	}
}
```

Program: 35

Wap in Java program to use subpackage.

```
package p1.p2;
class subp
{
	public static void main(String s[])
	{
		int x=1, y=2;
		System.out.println(x);
		System.out.println(y);
	}
```

}

import

1. It acts as an indication to the compiler.
2. It identifies the actual name of a class.
3. There are no import statement in a ,class file.
4. It is different from “include” as there is nothing global in java.
5. It allows to write the class name as its short name.

Program: 36

Wap in Java to show use of import.

```
package p1;

public class A

{

        public void msg()

        {

                System.out.println(‘Hello”);

        }

}

package p3;

import p1.A;

class B

{

        public static void main(String s[])

        {

                A ob=new A();

                ob.msg();
```

```
        }
}
```

Program: 37

Wap in Java to use package in object creation.

```
package p1;
class p
{
        int x=5, y=6;
        void show()
        {
                System.out.println(x);
                System.out.println(y);
        }
}
class pack
{
        public static void main(String s[])
        {
                p1.p op = new p1.p();
                op.show();
        }
}
```

Program: 38

Wap in Java to use package in inheritance.

```
package p1;
class a
{
	int x=100;
}
class b extends p1.a
{
	int y=200;
	void show()
	{
		System.out.println(y);
		System.out.println(x);
	}
}
class packinherit
{
	public static void main(String s[])
	{
		p1.b ob=new p1.b();
		ob.show();
	}
}
```

Program: 39

Wap in Java to use 2 packages in inheritance.

```
package p1;
public class x
{
	public int a=45;
}
package p3;
class y extends p1.x
{
	void show()
	{
		System.out.println(a);
	}
}
class z
{
	public static void main(String s[])
	{
		p3.y op=new p3.y();
		op.show();
	}
}
```

Array in java

1. It behaves in the same manner as in C or C++.
2. The creation is different.
3. There is no static creation.
4. It must be created by using 'new' operator.
5. Example of array creation
    a. datatype array_name[]=new datatype[size];
    b. int no[]=new int[5];
    c. int x=5;
       int arr[]=new int[x];
    d. datatype[] = new datatype[size];
       int[] no=new int[];
    e. int no[]={5,15,25,35,45,55,65};

Program: 40

Wap in Java to use of array.

```
class TestArray
{
    public static void main(String s[])
    {
        int a[]={3,4,5,6};
        for(int i=0;i<a.length;i++)
            System.out.println(a[i]);
    }
}
```

Program: 41

Wap in Java to input 2 integers from keyboard and display their sum.

```
import java.io.*;
```

```
class k
{
	public static void main(String s[])
	{
	DataInputStream ob=new DataInputStream(System.in);
	int x, y, z;
	try
	{
		x=Integer.parseInt(ob.readLine());
		y=Integer.parseInt(ob.readLine());
	}
	catch(Exception e) { }
	z=x+y;
	System.out.println("Sum= "+z);
	}
}
```

Program: 42

Wap in Java to input 10 integers and sorting them in ascending order.

```
import java.util.*;
class AO
{
	public static void main(String s[])
	{
		int temp;
```

```
Scanner oa=new Scanner(System.in);
int a[]=new int[10];
System.out.println("Enter 10 integers: ");
for(int i=0;i<10;i++)
{
        a[i]=oa.nextInt();
}
for(int i=0;i<10;i++)
{
        for(int j=i+1;j<10;j++)
        {
                if(a[i]>a[j])
                {
                        temp=a[i];
                        a[i] =a[j];
                        a[j]=temp;
                }
        }
}
System.out.println("Ascending Order ");
for(int i=0; i<9;i++)
{
        System.out.println(a[i]+" ");
}
```

```
            System.out.println(a[9]);
        }
}
```

Program: 43

Wap in Java to input a number and display sum of its digit.

```
import java.util.*;
class sum
{
        public static void main(String s[])
        {
                int m,n,sum=0;
                Scanner a= new Scanner(System.in);
                System.out.println("Enter the number: ");
                try{
                        m=a.nextInt();
                }catch(Exception e) { }
                while(m>0)
                {
                        n=m%10;
                        sum=sum+n;
                        m=m/10;
                }
                System.out.println("Sum of digits: "+sum);
        }
```

}

Access Modifiers (static, final, abstract etc.)

1. Are different from access specifiers
2. Assigns a special priviledge /power to a member
3. Must be explicitly specified with the member
4. There is no default access modifier
5. There is no group access modifier

Different access modifiers are

1. static
2. final
3. abstract
4. volatile
5. transient
6. synchronized

static

1. Makes a member instance independent
2. Is created/allotted memory when class prototype is loaded
3. Are accessed as classname.member name.
4. Can be applied to data member, method or block of code

Data Member	Method Member	Block
1. Are identified as class variables 2. Only one copy is created 3. Can be initialized inline or by using static block.	1. Can only access static data or static methods. 2. To access non static members, it must create an object and access 3. Can not access	1. Is a set of instructions 2. Executed only once when class is loaded. 3. Used to initialize static variables.

	“this” or “super”	

Program: 44

Wap in Java for static data member.

```
class ANCHAL
{
	int a;
	static int b=5;
	void show()
	{
		System.out.println(a);
		System.out.println(b);
	}
	static void display()
	{
		System.out.println(b);
	}
}
class SAGAR
{
	public static void main(String s[])
	{
		ANCHAL ob = new ANCHAL();
		ob.show();
```

```
        ob.display();
    }
}
```

Program: 45

Wap in Java for static block.

```
class Example
{
    static int num;
    static
    {
        num=45;
    }
    public static void main(String s[])
    {
        System.out.println("Value of num: "+num);
    }
}
```

final

1. Marks the content of a member as unmodifiable.
2. Can not be used with "abstract" access modifier.
3. Can be applied to data member, method member or a class.

Data member	Method Member	Class
1. Marks it as a constant	1. Prevents overriding	1. Prevents inheritance

2. Must be initialized inline 3. Like static, only one copy is created.		

Program: 46

Wap in Java to show use of final.

```
final class BIKE
{
    final int limit=60;
    final void run()
    {
        System.out.println("Limit= "+limit);
        System.out.println("Running.....");
    }
    public static void main(String s[])
    {
        BIKE ob= new BIKE();
        ob.run();
    }
}
```

abstarct:

1. It can not be used with final.
2. It can be applied to either a method or class.

Method	Class
1. Indicates that the method does not have a body 2. Makes it mandatory for the child class to provide a body by overriding.	1. An incomplete class. 2. Can not be instantiated directly. 3. Must be declared abstract if a. It contains at least one abstract method. b. It inherits from an abstract class and does not override all the abstract methods. c. It inherits from an interface and does not over ride all the methods.

Example of abstract mehods:

```
abstract class a
{
abstract void p1();
void p2()
        {
        }
}
```

Program: 47

Wap in Java for abstract method members.

```
abstract class Hello
{
        void show()
```

```
        {
                System.out.println("It is SUPER show");
        }
        abstract void display();
}
class Helloo extends Hello
{
        void display()
        {
                System.out.println("It is SUB show");
        }
}
class MN
{
        public static void main(String s[])
        {
                Helloo ob=new Helloo();
                ob.show();
                ob.display();
        }
}
```

Program: 48

Wap in Java for abstract class.

```
abstract class shape
```

```
{
        abstract void draw();
}
class circle extends shape
{
        void show()
        {
                System.out.println("Drawing Circle");
        }
}
class Test
{
        public static void main(String s[])
        {
                shape a=new circle();
                a.draw();
        }
}
```

Interface

1. Is created, saved and compiled into a .class just like a class.
2. Interfaces can be within package and sub package.
3. Contains data members and method members.
4. All Data members are final.
5. All Method members are abstract.
6. Created by using the keyword “interface”.
7. Default access specifier is public for the members.
8. Allows multiple inheritance.
9. Child class inherits by the keyword “implements”
10. Can inherit from another interface, but never from a class.
11. It does not need any constructor as there is no variable.

Example of interface:

```
interface a

{

int n1=5;

void show();

}
```

Keyword	class a extends b	class a implements b	NOT ALLOWED	class a Implements b, c	class a extends b implements c	class a extends b implements c, d	Interface a extends b	NOT ALLOV
Parent	Class	Interface	Class, Class	Interface, Interface	Class, Interafce	Class, Interface, Interface	Interface	Class
Child	Class	Class	Class	Class	Class	Class	Interface	Interfac

Program: 49

Wap in Java for interface.

```
interface MyInterface
{
  public void show();
}
class Demo implements MyInterface
{
  public void show()
  {
        System.out.println("Implementation of method show");
  }
  public static void main(String s[])
  {
        MyInterface obj = new Demo();
        obj.show();
  }
}
```

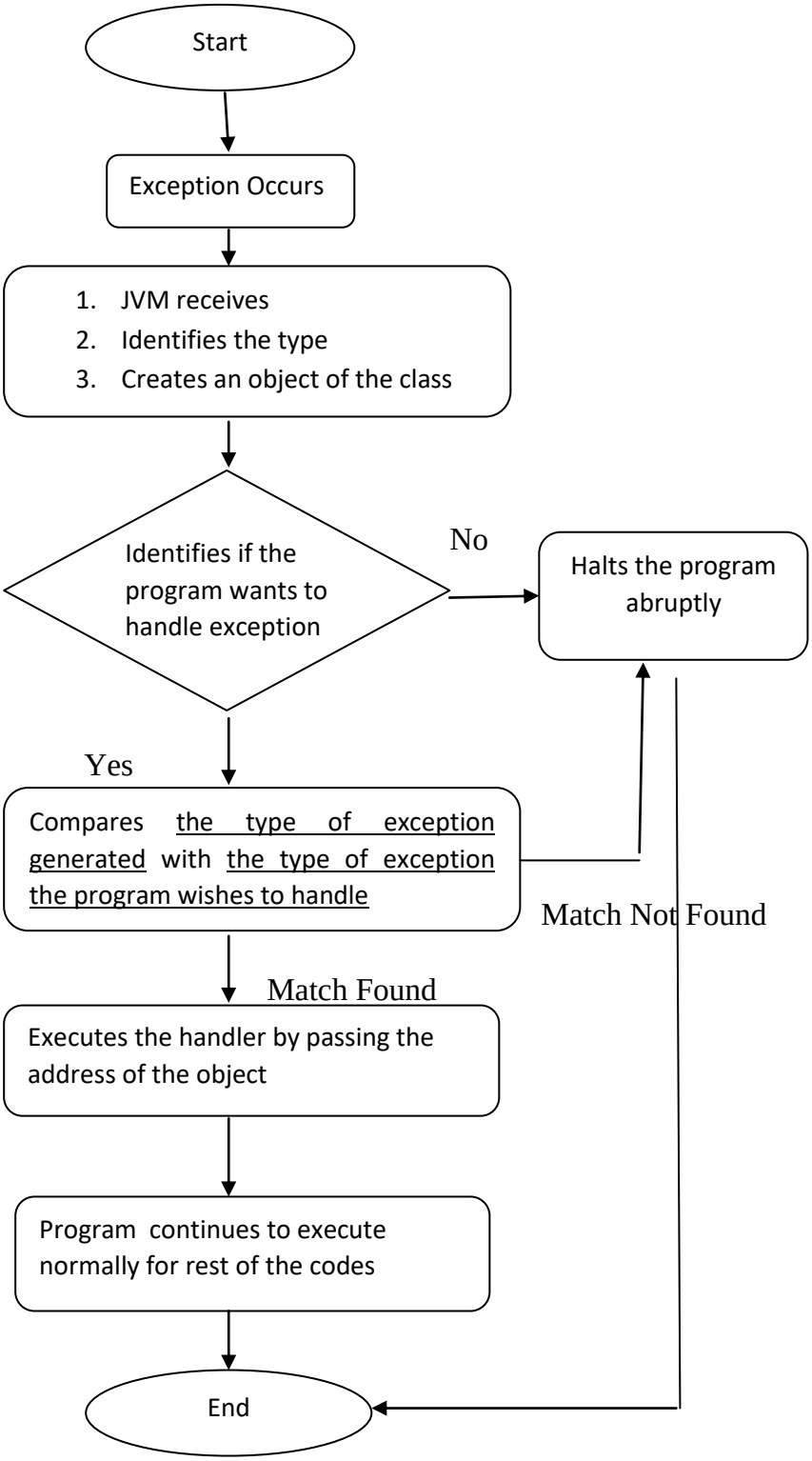
Exception Handling

1. Exception: It is an abnormal scenario which halts a program's execution abruptly.
2. It occurs at run time.
3. As much as possible, exception must be handled.
4. Every exception is defined as a class in java.
5. JVM allows a program to continue to execute normally if exception has been handled.
6. Handling: is specifying a set of instructions and associating with an exception type.
7. Types of Exception in Java : Exceptions are of 2 types
    a. Checked Exception
    b. Unchecked Exception

Checked Exception	Unchecked Exception
1. The complier forces us to handle 2. Examples: a. IOEXception b. SQLException c. ClassNotFoundException	1. Compiler does not force to handle 2. Examples: a. ArithmeticException b. ArrayIndexOutofBoundsException c. NullPOinterException

8. There are 5 terns for exception handling in java
    a. try
    b. catch
    c. throw
    d. throws
    e. finally

Exception Handling Flow Chart



try:

1. It is a block of code.
2. It indicates to the JVM that in case of an exception, in any of the statements of the block, the program wishes to handle it.
3. **It must immediately be followed by at least a "catch" or a "finally" block.**
4. In case of an exception in any of the statements of the block, the rest of the try block is ignored and JVM searches for a matching catch block.
5. Multiple try blocks can exist in a program.
6. Try blocks can be nested.

catch:

1. It is block of code.
2. It associates the handler with an exception.
3. **It must immediately follow a try**
4. Is uniquely identified by an exception type.
5. A catch block can handle exception specified with it and all the subclasses of the specified exception class.
6. For multiple catch blocks, the order of positioning must be child to parent.
7. In case of an exception, in the associated try, generated exception is compared with specified exception of the catch block
    a. If no match occurs
        i. Catch block is ignored
        ii. Next catch is compared
    b. If match occurs
        i. Block is executed
        ii. Remaining catch blocks are ignored
        iii. Program continues to execute from the first line after the last catch block.
8. We need to specify a variable name in the catch block to hold the address of the exception object which was created by the system.

throw:

1. It is used to create/ generate an exception.
2. It is useful for a programmer when he/she wants to handle the exception at 2 levels.
    a. Calling level
    b. Called level
3. It guarantees an exception

throws:

1. It is used as an alert to the caller that the called method may generate an exception.
2. It does not guarantee an exception.
3. It is useful where a method has a probability of generating a checked exception and the method does not want to handle it.

finally:

1. It is a block of code.
2. It contains a set of instructions that will definitely execute.
3. It is useful in scenario which has multiple exit points and we want a specific code to be executed irrespective of the exit taken.
4. It must follow a try block.
5. There can be only one finally block per try
6. It can be used with catch blocks also, but must be positioned after the last catch.
7. If a return statement is encountered, then the return becomes effective only after the finally block is executed.

Program: 50

Wap in Java to show the use of try….catch.

```
class Example1
{
  public static void main(String s[])
   {
    int num1, num2;
```

```
    try
    {
      num1 = 0;
      num2 = 62 / num1;
      System.out.println(num2);
      System.out.println("Hey I'm at the end of try block");
    }
    catch (ArithmeticException e)
    {
      System.out.println("You should not divide a number by zero");
    }
    System.out.println("I'm out of try-catch block in Java.");
  }
}
```

Program: 51

Wap in Java for throw.

```
class ThrowsExecp
{
   static void fun() throws IllegalAccessException
   {
      System.out.println("Inside fun(). ");
      throw new IllegalAccessException("demo");
   }
   public static void main(String args[])
```

```
    {
        try
        {
            fun();
        }
        catch(IllegalAccessException e)
        {
            System.out.println("caught in main.");
        }
    }
}
```

Another Program:

```
class a
{
    static void fun()
    {
     try{
        System.out.println("Inside fun(). ");
        int a=5/0;
          }catch(Exception e)
                {
               throw e;
               }
    }
```

```
   public static void main(String args[])
   {
      try
      {
         fun();
      }
      catch(Exception e)
      {
         System.out.println("caught in main.");
      }
   }
}
```

Program: 52

Wap in Java for throws.

```
class Throws
{
   static void fun() throws IllegalAccessException
   {
      System.out.println("Inside fun(). ");
      throw new IllegalAccessException("demo");
   }
   public static void main(String args[])
   {
```

```
        try
        {
            fun();
        }
        catch(IllegalAccessException e)
        {
            System.out.println("caught in main.");
        }
    }
}
```

Another program

```
class a
{
    static void fun() throws ArithmeticException
    {
         System.out.println("Inside fun(). ");
        int a=5/0;
    }
    public static void main(String args[])
    {
        try
        {
            fun();
        }
```

```
        catch(ArithmeticException e)
        {
            System.out.println("caught in main.");
        }
    }
}
```

Program: 53

Wap in Java for finally.

```
class TFB
{
  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...");
  }
}
```

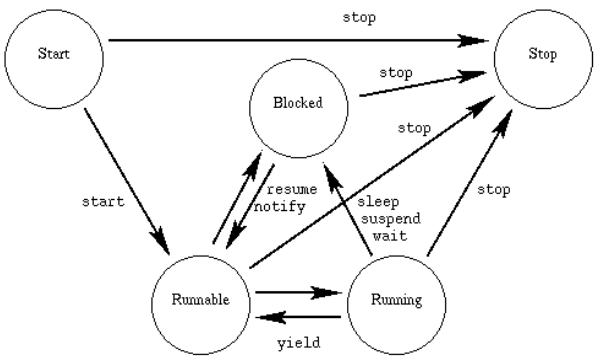
Thread

1. It identifies an execution path.
2. It is an invisible component.
3. It binds the instructions in a specific order.
4. It allows one portion of the program to be executed at any point of time.
5. Every program is given a thread by default and it is known as Main thread.
6. Every thread has its own set of instructions to execute.
7. If more than one part of the same program is to be executed, we need to have multiple threads – This concept is known as **MULTI THREADING.**
8. Threads, created by the programmer are known as **Child Threads.**
9. At execution, each thread runs independent of each other.
10. It is to be ensured by the programmer that the main thread is the last thread to finish execution.
11. A thread is allocated CPU time on the basis of a value known as **priority value.**
12. The priority value of a thread can be from 1 to 10.
13. The default priority value of a thread is 5.
14. Programmer must not leave any orphan thread.

Steps to create a thread

1. Identify the instructions to be executed by the child thread.
2. Create an object of the child thread.
3. Start the execution of the child thread.

States of Thread



Thread is a predefined class which contains the following methods.

Sl. No.	Methods of Thread Class
1	String — **getName**() Returns this thread's name.
2	int — **getPriority**() Returns this thread's priority.
3	boolean — **isAlive**() Tests if this thread is still running.
4	void — **join**() Waits for this thread to die (terminate).
5	void — **run**() If this thread was constructed using a separate Runnable object, then that Runnable object's run method is called; otherwise, this method does nothing and returns. If thread class is extended and run() method is over-ridden in sub-class then the over-ridden run() method is called.
6	void — **setName**(String name) Changes the name of this thread to be equal to the argument name.
7	static void — **sleep**(long millis) throws InterruptedException Causes the currently executing thread to sleep for the specified number of milliseconds.
8	static void — **sleep**(long millis, int nanos) throws InterruptedException Causes the currently executing thread to sleep (cease execution) for the specified number of milliseconds plus the specified number of nanoseconds.
9	void — **start**() Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.
10	static void — **yield**() Causes the currently executing thread object to temporarily pause and allow other threads to execute.
11	static Thread — **currentThread**() Returns a reference to the currently executing thread object.

12. void setPriority(int value) : is used to allocate a priority value to a thread.

13. void notify() : is used to resume execution of a waiting thread.

14. void notifyAll() : is used to resume the execution of all waiting threads simultaneously.

Program: 54

Wap in Java for Multithreading.

```
class myt extends Thread
{
	String nm;
	myt(String x)
	{
		System.out.println("Child Thread Created");
		nm=x;
	}
	public void run()
	{
		System.out.println("Child Thread Start");
		for(int i=0;i<10;i++)
		{
			System.out.println(nm+" ;"+i);
			try
			{
				Thread.sleep(300);
			}
			catch(Exception e)
			{ }
		}
		System.out.println("Child Thread Finished");
```

```
        }
}
class tdemo
{
        public static void main(String s[])
        {
                System.out.println("Main Starts");
                myt ob=new myt("Sagar");
                System.out.println("Still Single Thread");
                ob.start();
                System.out.println("Main Continues");
                for(int p=0;p<=10;p++)
                {
                        System.out.println("Main: "+p);
                }
                try
                {
                        Thread.sleep(400);
                }
                catch(Exception e)
                { }
                System.out.println("Main Method Completed");
        }
}
```

Program: 55

Wap in Java to show the use of setPriority() for 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();
     }
}
```

Program: 56

Wap in Java to show use of Thread.sleep().

```
class TestSleepMethod1 extends Thread
```

```
{
 public void run()
{
  for(int i=1;i<5;i++)
{
   try{
Thread.sleep(500);
}
catch(InterruptedException e)
{
System.out.println(e);
}
    System.out.println(i);
  }
 }
 public static void main(String args[])
{
  TestSleepMethod1 t1=new TestSleepMethod1();
 TestSleepMethod1 t2=new TestSleepMethod1();


  t1.start();
 t2.start();
 }
}
```

Runnable interface

1. Drawbacks of inheriting from Thread class
    a. One class can not be used to act as a container and as a Thread simultaneously.
    b. One class can not inherit from 2 classes as Java does not support multiple in heritance.
2. Runnable interface contains only run() method.
3. For other Thread methods, we have to create one object of Thread class and access them.

Program: 57

Wap in Java to show use of Runnable interface.

```
class Multi3 implements Runnable
{
    String nm;
    Thread ob;
    myt(String x)
    {
        System.out.println("Child Thread Created");
        nm=x;
        ob=new Thread(this);
    }
    public void run()
    {
        System.out.println("Child Thread Start");
        for(int i=0;i<10;i++)
        {
```

```
                System.out.println(nm+" ;"+i);
                try
                {
                        Thread.sleep(300);
                }
                catch(Exception e)
                { }
        }
        System.out.println("Child Thread Finished");
    }
}
class tdemo
{
    public static void main(String s[])
    {
        System.out.println("Main Starts");
        Multi3 obj=new Multi3("Sagar");
        System.out.println("Still Single Thread");
        obj.ob.start();
        System.out.println("Main Continues");
        for(int p=0;p<=10;p++)
        {
            System.out.println("Main: "+p);
        }
```

```
            try
            {
                  Thread.sleep(400);
            }
            catch(Exception e)
            { }
            System.out.println("Main Method Completed");
      }
}
```

Applet

1. It is a java class that executes in a browser.
2. It is stored in a web server, but are executed in a remote computer.
3. It requires a java compatible web browser.
4. It is not independent.
5. It is embedded into a web page(html file).
6. Life cycle of Applet corresponds to the life cycle of embedding web page.
7. It is used to empower a web page by enhancing the features of Threading, Animation, Data base Connectivity etc.
8. It uses GUI to interact with the user.

Life cycle of HTML page	Life cycle of Applet
1. Loads 2. Gets activated 3. Displays the contents 4. Gets deactivated 5. Gets Destroyed	1. init() 2. start() 3. paint() 4. stop() 5. destroy()

init()

1. It is called only once at the first time the page (html page) is loaded.
2. A programmer generally does all initialization activities.

start()

1. It is called multiple times.
2. First time, after init() method is cakled.
3. Subsequently, every time the page is activated.

paint():

1. It is called multiple times.
2. Every time after the start method .
3. Every time the page is resized/ restarted.
4. Forcefully, by the user.
5. Receives an object of Graphics class by default.

stop():

1. It is called multiple times.
2. It is called whenever the page is deactivated.

destroy():

1. It is called once.
2. It is called before the object is garbage collected.

Writing an Applet program

1. Write a class by sub classing the java.applet. Applet class
2. Override the methods as per need.
3. Save and compile into a .class file.

Executing the Applet

1. Write a .html file
2. Place the Applet by using <applet> tag.
3. Open a browser.
4. Open the html file.

Simple html program

```
<html>
<head>
<title>
My new Program with Applet
</title>
</head>
<body>
<br>
<br>
<hr>
<h1>
This is before the Applet
</h1>
<applet code=a width=300 height=200>
```

```
</applet>
<h2>
This is after applet
</h2>
<br>
<hr>
<hr>
</body>
</html>
```

Graphics class

1. It is a member of java.awt package
2. It contains methods using which we can draw text and shapes on a graphical component.
3. It is an abstract class.
4. To create an object, we must call a method getGraphics() on a graphical component.
5. getGraphics() is a method of java.lang.Image class
6. Signature of getGraphics

```
public abstract Graphics getGraphics()
```

Methods of Graphics class

1. public abstract void drawString(String str, int x, int y): is used to draw the specified string.
2. public void drawRect(int x, int y, int width, int height): draws a rectangle with the specified width and height.
3. public abstract void fillRect(int x, int y, int width, int height): is used to fill rectangle with the default color and specified width and height.
4. public abstract void drawOval(int x, int y, int width, int height): is used to draw oval with the specified width and height.

5. public abstract void fillOval(int x, int y, int width, int height): is used to fill oval with the default color and specified width and height.
6. public abstract void drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).
7. public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.
8. public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used draw a circular or elliptical arc.
9. public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to fill a circular or elliptical arc.
10. public abstract void setColor(Color c): is used to set the graphics current color to the specified color.
11. public abstract void setFont(Font font): is used to set the graphics current font to the specified font.

Example

Color c= new Color(170, 250, 100)

R G B

g.setColor(c);

g,setColor(Color.red);
g.setColor(Color.blue);
g.getColor(Color.green);
g.setColor(Color.cyan);
g,setColor(Color.yellow);
g.setColor(Color.violet);

Example:

Graphicsdemo.java

```
import java.applet.Applet;
import java.awt.*;
```

```
public class GraphicsDemo extends Applet{

public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);

g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);

}
}
```

a.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

Program: 58

Write a sample Applet program in Java.

```
import java.applet.*;

import java.awt.*;
```

```
<applet code=myapp.class width=200 height=200> </applet>
public class myapp extends Applet
{
	public void init()
	{
		System.out.println("1. init()");
	}
	public void start()
	{
		System.out.println("2. start()");
	}
	public void stop()
	{
		System.out.println("3. stop()");
	}
	public void paint(Graphics g)
	{
		g.drawString("Applet : Running",100,100);
	}
	public void destroy()
	{
		System.out.println("4. destroy()");
	}
}
```

Program: 59

Wap in Java to display Hello inside a rectangle on one Applet.

```
import java.applet.*;
import java.awt.*;
public class dd extends Applet
{
  public void init()
  {
     setBackground(Color.cyan);
     setForeground(Color.blue);
   }
   public void paint(Graphics g)
   {
      g.drawRect(100,100,80,60);
      g.drawString("Hello",120,130);
    }
}
```

Program: 60

Wap in java to design a face on one Applet.

```
import java.awt.*;
public class face extends Applet
{
public void paint(Graphics g)
{
        g.drawOval(200,200,300,340);
```

```
        g.drawOval(260,315,60,30);
        g.drawOval(380,315,60,30);
        g.drawLine(210,312,190,300);
        g.drawLine(490,312,510,300);
        g.drawLine(200,362,185,355);
        g.drawLine(500,362,515,355);
        g.drawLine(190,300,185,355);
        g.drawLine(510,300,515,355);
        g.fillOval(285,325,10,10);
        g.fillOval(405,325,10,10);
        g.drawArc(245,260,30,10,180,0);
        g.drawLine(350,340,330,400);
        g.fillArc(300,440,100,30,180,180);
        g.fillArc(350,270,5,50,0,360);
        g.drawLine(350,340,370,400);
        g.drawArc(330,396,40,10,0,180);
        g.drawLine(280,520,230,590);
        g.drawLine(420,520,470,590);
        g.drawLine(230,590,470,590);
        g.drawLine(350,200,350,170);
        g.drawLine(370,202,380,171);
        g.drawLine(330,202,320,171);
        g.drawLine(310,205,291,174);
        g.drawLine(390,205,409,174);
        g.drawLine(413,215,437,185);
        g.drawLine(285,215,265,185);
}
<applet code=face.class width=400 height=300> </applet>
```

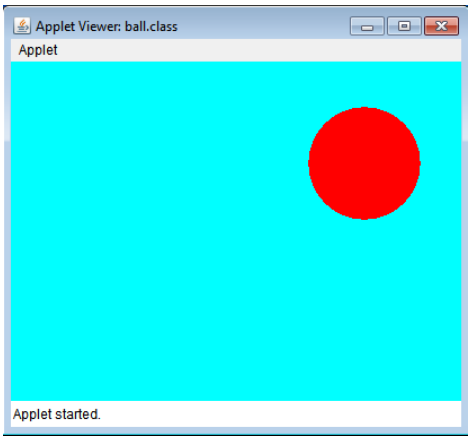
Program: 61

Wap in java to move a ball from left to right on one Applet.

```
import java.applet.*;
```

```
import java.awt.*;
public class ball extends Applet
{
public void init()
{
setBackground(Color.cyan);
setForeground(Color.red);
}
public void paint(Graphics g)
{
for(int i=0;i<900;i++)
{
g.setColor(Color.red);
g.fillOval(i,40,100,100);
try{
Thread.sleep(50);
}catch(Exception e)
{}
g.setColor(Color.cyan);
g.fillOval(i,40,100,100);
}
}
}
//<applet code=ball.class width=400 height=300> </applet>
```

OUTPUT



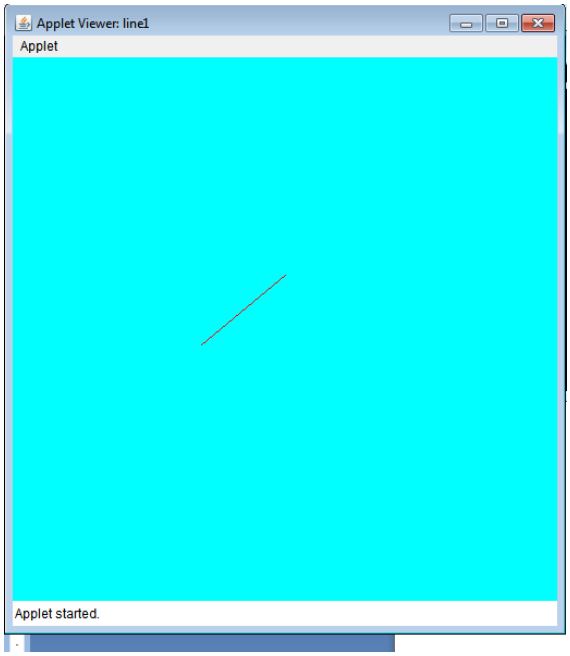
Program: 62

Wap in java to rotate a line about one of its end point.

```
import java.awt.*;
import java.applet.*;
//<applet code=line1 width=500 height=500> </applet>
public class line1 extends Applet
{
public void init()
{
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
int x=250;
```

```
int y=200;
for(int i=0;i<=800;i++)
{
double j=i*Math.PI/360;
int x1=(int)(250-100*Math.sin(j));
int y1=(int)(200+100*Math.cos(j));
g.setColor(Color.red);
g.drawLine(x1,y1,x,y);
try{
Thread.sleep(50);
}catch(Exception e)
{}
g.setColor(Color.cyan);
g.drawLine(x1,y1,x,y);
}
}
}
```

OUTPUT



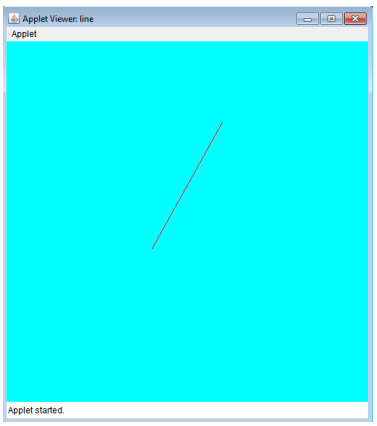
Program: 63

Wap in java to rotate a line about its centre on one Applet.

```
import java.awt.*;
import java.applet.*;
//<applet code=line width=500 height=500> </applet>
public class line extends Applet
{
public void init()
{
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
for(int i=0;i<=800;i++)
```

```
{
double j=i*Math.PI/360;
int x=(int)(250+100*Math.sin(j));
int y=(int)(200-100*Math.cos(j));
int x1=(int)(250-100*Math.sin(j));
int y1=(int)(200+100*Math.cos(j));
g.setColor(Color.red);
g.drawLine(x1,y1,x,y);
try{
Thread.sleep(50);
}catch(Exception e)
{}
g.setColor(Color.cyan);
g.drawLine(x1,y1,x,y);
}
}
}
```

OUTPUT



Program: 64

Wap in java to design a clock on one Applet.

```
import java.awt.*;
import java.applet.*;
import java.util.*;
import java.text.*;
//<applet code=clc width=700 height=700></applet>
public class clc extends Applet implements Runnable
{
int w=400,h=400;
Thread t=null;
boolean ts;
int hr=0,mn=0,sc=0;
String s=" ";
public void init()
{
setBackground(Color.black);
}
public void start()
{
t=new Thread(this);
t.setPriority(Thread.MIN_PRIORITY);
ts=false;
t.start();
}
public void stop()
{
ts=true;
}
public void run()
{
try{
```

```
while(true)
{
Calendar cal=Calendar.getInstance();
hr=cal.get(Calendar.HOUR_OF_DAY);
mn=cal.get(Calendar.MINUTE);
sc=cal.get(Calendar.SECOND);
SimpleDateFormat f=new SimpleDateFormat("hh:mm:ss",Locale.getDefault());
Date d=cal.getTime();
s=f.format(d);
repaint();
t.sleep(1000);
}
}catch(Exception e)
{}
}
void line(double a,int r,Graphics g)
{
a-=0.5*Math.PI;
int x=(int)(r*Math.cos(a));
int y=(int)(r*Math.sin(a));
g.drawLine(w/2,h/2,w/2+x,h/2+y);
}
public void paint(Graphics g)
{
g.setColor(Color.gray);
line(2*Math.PI*hr/12,w/7,g);
line(2*Math.PI*mn/60,w/5,g);
line(2*Math.PI*sc/60,w/3,g);
g.setColor(Color.white);
g.drawString(s,200,h-10);
}
}
```

Program: 65

Wap to use Applet with Thread.

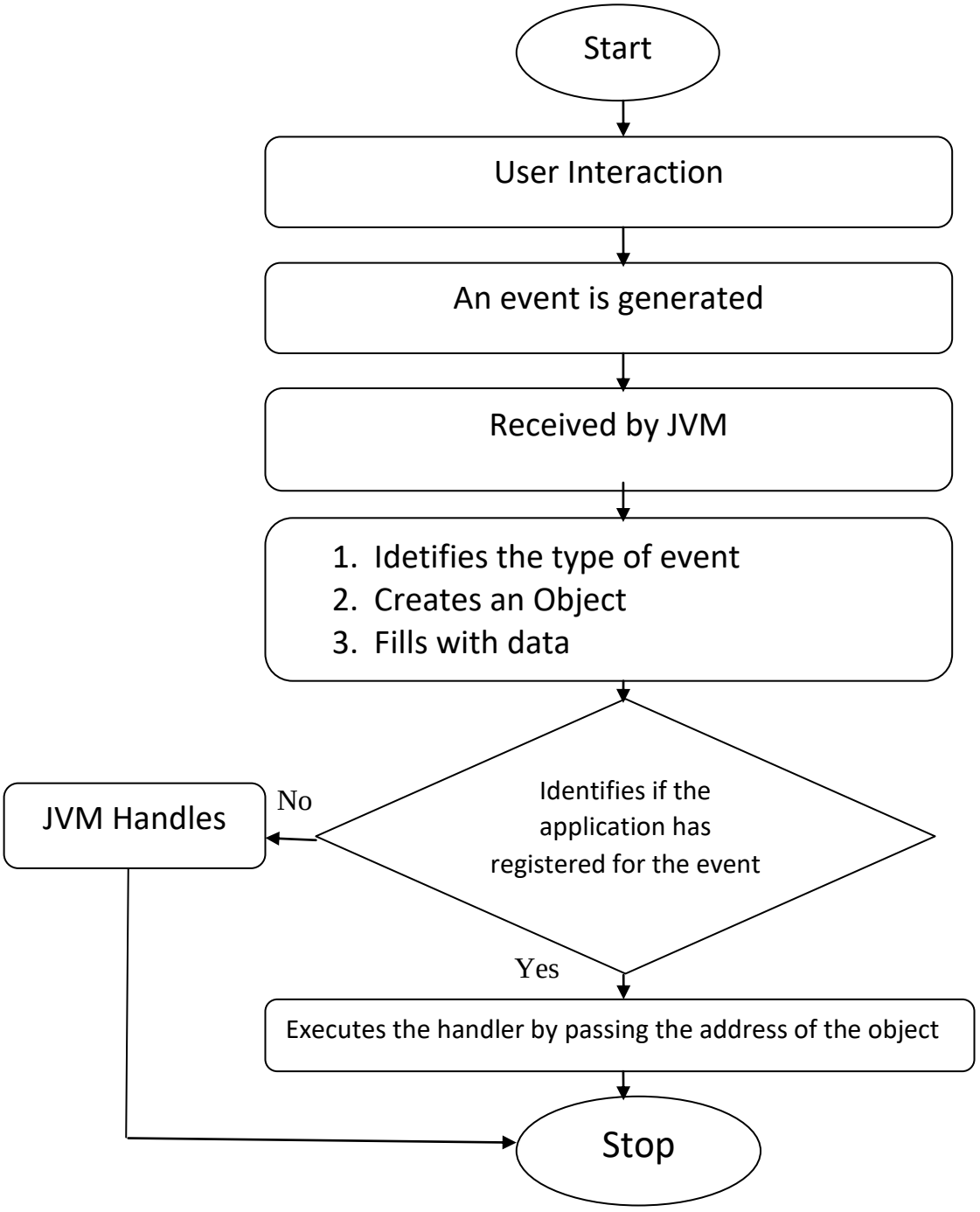
```
import java.awt.*;
import java.applet.*;
//<applet code=hpyli width=300 height=200> </applet>
public class hpyli extends Applet implements Runnable
{
Thread ob;
String nm="HAPPY HOLI ";
boolean flag;
public void init()
{
setBackground(Color.cyan);
setForeground(Color.blue);
setFont(new Font("Arial",Font.BOLD,24));
}
public void paint(Graphics g)
{
g.drawString(nm,40,120);
}
public void start()
{
flag=true;
ob=new Thread(this);
ob.start();
}
public void stop()
{
flag=false;
}
public void run()
{
```

```
char ch;
while(flag)
{
ch=nm.charAt(0);
nm=nm.substring(1);
nm+=ch;
try{
Thread.sleep(400);
}catch(Exception e)
{}
repaint();
}
}
}
```

EVENT HANDLING

1. **Event** – Any user interaction on a graphical component is known as an event.
2. The user interaction may be
    a. Pressing of mouse
    b. Movement of mouse
    c. Pressing a key on Key Board
    d. Clicking on window
3. Graphical component is one visual object like
    a. Button
    b. Checkbox
    c. Button
    d. Radio Button
    e. Label
    f. Text Field
    g. Text Area
    h. List
    i. Checkbox
    j. Applet
    k. Frame
    l. Dialog Box
    m. Panel
    n. Window
4. Broadly events are of 4 types
    a. Mouse related
    b. Keyboard related
    c. Component related
    d. Window related
5. Event is a predefined class just like Applet and Thread.
6. **Handling** – Reacting to an event and executing a set of instructions.

Flow Chart of Event Handling



Registering Event

1. It is an indication to the JVM that the programmer wants to handle an event
2. Two information are supplied
    a. Which event?
    b. Address of handler object(Where the event has happened?)
3. How to register one event

addEvenTypeListener(address of handler object)

Events Chart

Sl. No.	Event Type	Event Class	Handler interface	Handler Methods	Generated by	Has adaptor?
1	Mouse	MouseEvent	MouseListener	mouseClicked(MouseEvent me)	Frame, Applet, Dialog, Panel	Yes
				mousePressed(MouseEvent me)		
				mouseReleased(MouseEvent me)		
				mouseEntered(MouseEvent me)		
				mouseExited(MouseEvent me)		
2	Mouse	MouseEvent	MouseMotionListener	mouseMoved((MouseEvent me)	Frame, Applet, Dialog, Panel	Yes
				mouseDregged((MouseEvent me)		
3	Action	ActionEvent	ActionListener	actionPerformed(ActionEvent ae)	Button, List(on Double Click)	No
4	Item	ItemEvent	ItemListener	itemStateChanged(ItemEvent ie)	List(On single click), Checkbox, Choice	No
5	Window	WindowEvent	WindowListener	windowOpened(WindowEvent we)	Frame, Dialog, Window	Yes
				windowClosed(WindowEvent we)		
				windowActivated(WindowEvent we)		
				windowDeactivated(WindowEvent we)		
				windowIconified(WindowEvent		

				we)		
				windowDeiconified(WindowEvent we)		
				windowClosing(WindowEvent we)		
6	Key	KeyEvent	KeyListener	keyPressed(KeyEvent ke)	Components	Yes
				keyReleased(KeyEvent ke)		
				keyTyped(KeyEvent ke)		

Program: 66

Wap in java to display Hello at every mouse click on one Applet.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//<applet code=hello width=500 height=500 > </applet>
public class hello extends Applet implements MouseListener
{
int x,y;
public void init()
{
addMouseListener(this);
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
g.drawString("HELLO",x,y);
}
public void mouseClicked(MouseEvent me)
{
x=me.getX();
y=me.getY();
repaint();
```

```
}
public void mousePressed(MouseEvent me)
{
}
public void mouseReleased(MouseEvent me)
{
}
public void mouseEntered(MouseEvent me)
{
}
public void mouseExited(MouseEvent me)
{
}
}
```

Program: 67

Wap in java to keep all the stars generated for each mouse click on an Applet.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//<applet code=cstar width=500 height=500> </applet>
public class cstar extends Applet implements MouseListener
{
int x,y;
public void init()
{
addMouseListener(this);
setBackground(Color.cyan);
}
public void update(Graphics g)
{
paint(g);
}
```

```
public void paint(Graphics g)
{
g.drawString("*",x,y);
}
public void mouseClicked(MouseEvent me)
{
x=me.getX();
y=me.getY();
repaint();
}
public void mousePressed(MouseEvent me)
{
}
public void mouseReleased(MouseEvent me)
{
}
public void mouseEntered(MouseEvent me)
{
}
public void mouseExited(MouseEvent me)
{
}
}
```

Program: 68

Wap in java draw a line between mouse press and release.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//<applet code=drline width=500 height=500> </applet>
public class drline extends Applet implements MouseListener
{
```

```
int x1,y1,x2,y2;
public void init()
{
addMouseListener(this);
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
g.drawLine(x1,y1,x2,y2);
}
public void mouseClicked(MouseEvent me)
{
}
public void mousePressed(MouseEvent me)
{
x1=me.getX();
y1=me.getY();
}
public void mouseReleased(MouseEvent me)
{
x2=me.getX();
y2=me.getY();
repaint();
}
public void mouseEntered(MouseEvent me)
{
}
public void mouseExited(MouseEvent me)
{
}
}
```

Program: 69

Wap in java to draw a rectangle between mouse press and mouse release.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//<applet code=crect width=500 height=500 > </applet>
public class crect extends Applet implements MouseListener
{
int x1,y1,x2,y2;
public void init()
{
addMouseListener(this);
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
int x,y;
if(x1<x2&&y1<y2)
g.drawRect(x1,y1,x2-x1,y2-y1);
else if(x1>x2&&y1>y2)
g.drawRect(x2,y2,x1-x2,y1-y2);
else if(x1>x2&&y1<y2)
g.drawRect(x2,y1,x1-x2,y2-y1);
else
g.drawRect(x1,y2,x2-x1,y1-y2);
}
public void mouseClicked(MouseEvent me)
{
}
public void mousePressed(MouseEvent me)
{
x1=me.getX();
```

```
y1=me.getY();
}
public void mouseReleased(MouseEvent me)
{
x2=me.getX();
y2=me.getY();
repaint();
}
public void mouseEntered(MouseEvent me)
{
}
public void mouseExited(MouseEvent me)
{
}
}
```

Program: 70

Wap in java to display background color change on mouse enter and mouse exit in one applet.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//<applet code=enex width=500 height=500 > </applet>
public class enex extends Applet implements MouseListener
{
int x,y;
public void init()
{
addMouseListener(this);
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
```

```
g.setColor(Color.blue);
}
public void mouseClicked(MouseEvent me)
{
}
public void mousePressed(MouseEvent me)
{
}
public void mouseReleased(MouseEvent me)
{
}
public void mouseEntered(MouseEvent me)
{
setBackground(Color.red);
}
public void mouseExited(MouseEvent me)
{
setBackground(Color.yellow);
}
}
```

Program: 71

Wap in java to draw a line during mouse move.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//<applet code=mmove width=500 height=500 > </applet>
public class mmove extends Applet implements MouseMotionListener
{
int x1,y1,x2,y2;
int v=0;
public void init()
{
```

```
addMouseMotionListener(this);
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
g.drawLine(x1,y1,x2,y2);
}
public void mouseMoved(MouseEvent me)
{
if(v==0)
{
x1=me.getX();
y1=me.getY();
v++;
}
else
{
x2=me.getX();
y2=me.getY();
repaint();
}
}
public void mouseDragged(MouseEvent me)
{
}
}
```

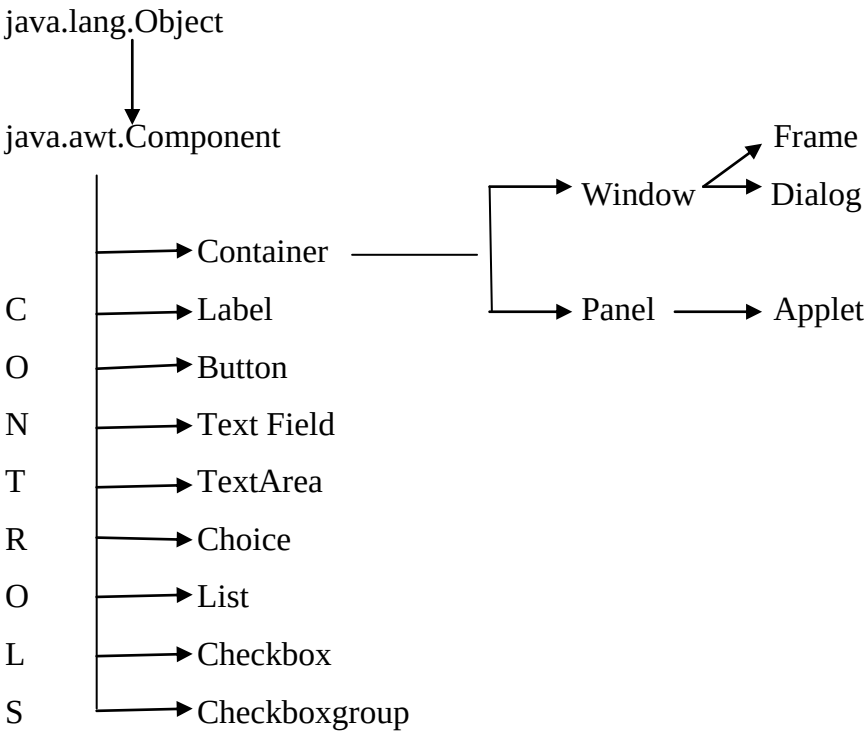
Program: 72

Wap in java to draw a line during mouse drag.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
//<applet code=mdrag width=500 height=500 > </applet>
```

```
public class mdrag extends Applet implements MouseMotionListener
{
int x1,y1,x2,y2;
int v=0;
public void init()
{
addMouseMotionListener(this);
setBackground(Color.cyan);
}
public void paint(Graphics g)
{
g.drawLine(x1,y1,x2,y2);
}
public void mouseMoved(MouseEvent me)
{
}
public void mouseDragged(MouseEvent me)
{
if(v==0)
{
x1=me.getX();
y1=me.getY();
v++;
}
else
{
x2=me.getX();
y2=me.getY();
repaint();
}
}
}
```

Component Class Hierarchy



Component Class Methods

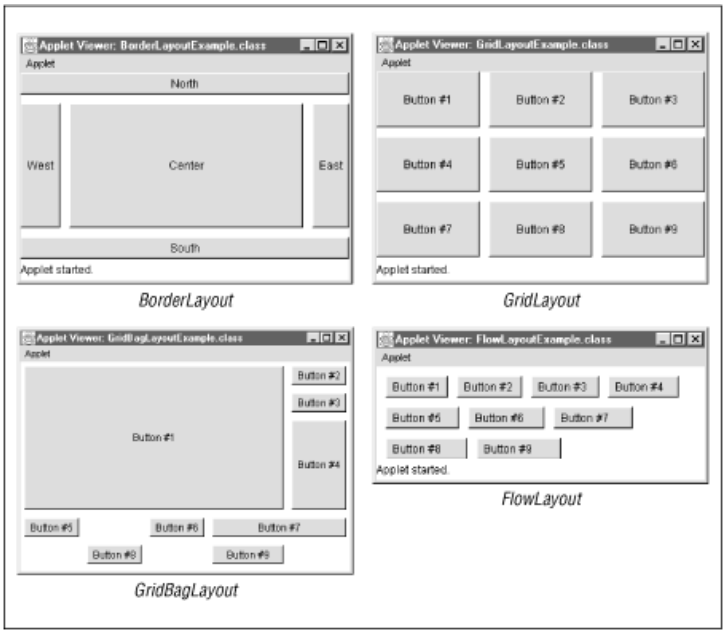
1. setForeground(Color c)
2. setBackground(Color c)
3. setFont(Font f)
4. setVisible(Boolean b)
5. setEnabled(Boolean b)
6. setSize(int w, int h)
7. setLocation(int x, int y)
8. setBounds(int x, int y, intw, int h)

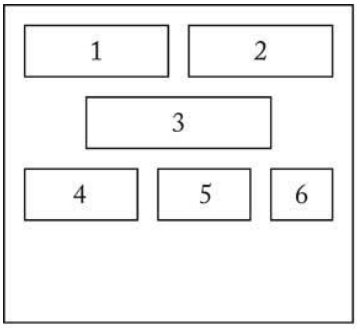
Container class methods

1. setLayout(LayoutManager mgr)
2. add(Component obj)
3. remove(Component obj)

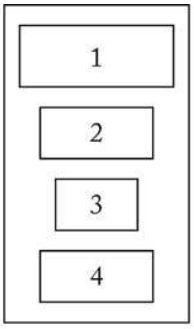
1. **setLayout** method is used to assign a layout to a container.
2. **LayoutManager** – It is a predefined class
3. It is associated with a container.
4. It controls the size and positioning of controls on a container.
5. An object of this class is created and applied to the container by using the setLayout method.
6. Different classes are
    a. FlowLayout
    b. BorderLayout
    c. GridLayout
    d. CardLayout
    e. BoxLayout
    f. GridBagLayout
    g. GroupLayout
    h. SpringLayout

Different examples of LayOuts

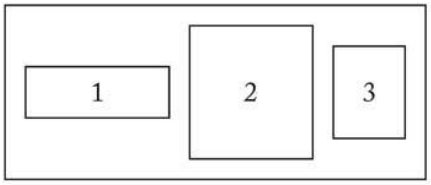




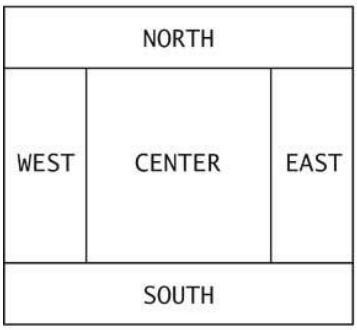
FlowLayout



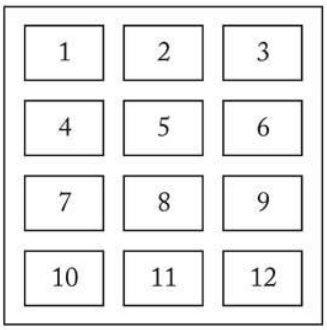
BoxLayout (vertical)



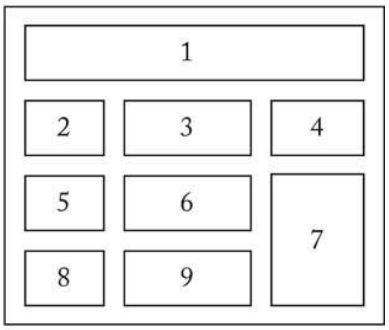
BoxLayout (horizontal)



BorderLayout



GridLayout

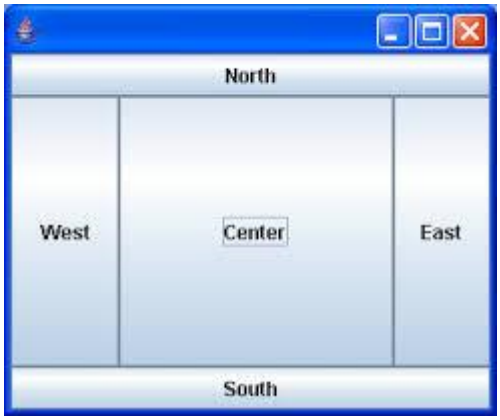


GridBagLayout

FlowLayout

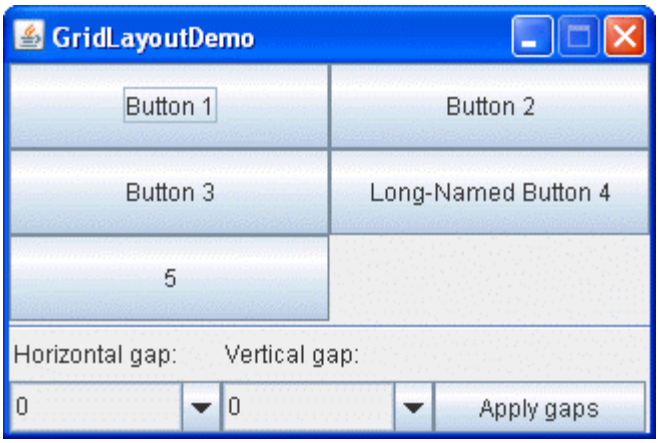
1. It is the default for Panel and Applet.
2. It adds components in a Left->Right and Top->Bottom order.
3. The default size of components is assumed.

BorderLayout



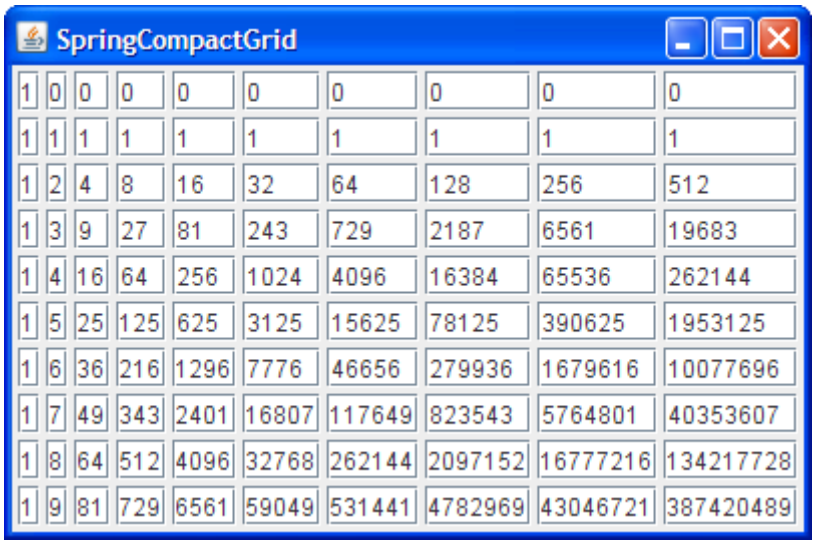
1. It is the default for Frame and Dialog.
2. It divides the container into 5 locations.
3. Each location can hold only one component.
4. Components stretch to fill the entire area.
5. Components are resized if container is resized.

GridLayout



1. It divides the container into a grid of rows and columns.
2. Each grid/location can hold one component.
3. All components are similar in size.
4. Components are added in a sequence.
5. Components are resized if the container is resized.

SpringLayout

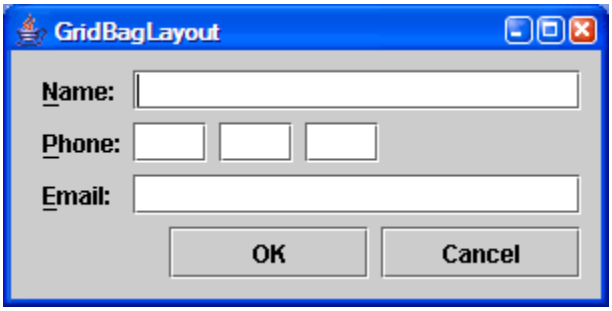


1. A SpringLayout arranges the children of its associated container according to a set of constraints.

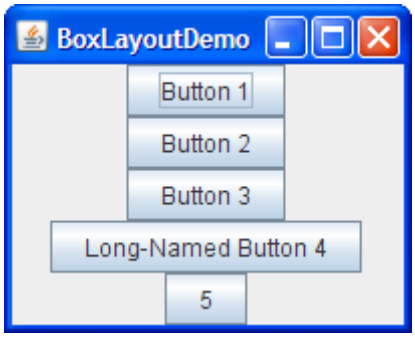
2. Constraints are nothing but horizontal and vertical distance between two component edges.
3. Every constrains are represented by a SpringLayout.Constraint object.
4. Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them.
5. Each edge position is dependent on the position of the other edge.
6. If a constraint is added to create new edge than the previous binding is discarded.
7. SpringLayout doesn't automatically set the location of the components it manages.

GridBagLayout

1. It is very fexible layout manager
2. It is an extension of GridLayout that provides some more features than the GridLayout
3. In GridLayout, all the cells have same height and width which is not necessary in the GridBagLayout.
4. Here, we can have cells of arbitrary width and height.
5. This can by specifying constraints.
6. To specify constraints, we have to create an object of FridBagConstraints.



BoxLayout



1. The BoxLayout is used to arrange the components either vertically or horizontally.
2. For this purpose, BoxLayout provides four constants. They are as follows:

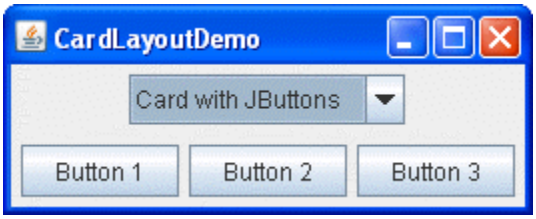
Fields of BoxLayout class

a. public static final int X_AXIS
b. public static final int Y_AXIS
c. public static final int LINE_AXIS
d. public static final int PAGE_AXIS

3. Constructor of BoxLayout class

   BoxLayout(Container c, int axis): creates a box layout that arranges the components with the given axis.

CardLayout



1. 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.
2. Constructors of CardLayout class
   a. CardLayout(): creates a card layout with zero horizontal and vertical gap.
   b. CardLayout(int hgap, int vgap): creates a card layout with the given horizontal and vertical gap.
3. Commonly used methods of CardLayout class
   a. public void next(Container parent): is used to flip to the next card of the given container.
   b. public void previous(Container parent): is used to flip to the previous card of the given container.
   c. public void first(Container parent): is used to flip to the first card of the given container.
   d. public void last(Container parent): is used to flip to the last card of the given container.
   e. public void show(Container parent, String name): is used to flip to the specified card with the given name.

GroupLayout

1. GroupLayout groups its components and places them in a Container hierarchically. The grouping is done by instances of the Group class.

2. Group is an abstract class and two concrete classes which implement this Group class are SequentialGroup and ParallelGroup.

3. SequentialGroup positions its child sequentially one after another where as ParallelGroup aligns its child on top of each other.

4. The GroupLayout class provides methods such as createParallelGroup() and createSequentialGroup() to create groups.

5. GroupLayout treats each axis independently. That is, there is a group representing the horizontal axis, and a group representing the vertical axis. Each component must exists in both a horizontal and vertical group, otherwise an IllegalStateException is thrown during layout, or when the minimum, preferred or maximum size is requested.

If the programmer wants to control the size and positions, then he/she must deactivate the layout managers by calling **setLayout(null)**.

Containers

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.

Frame

The methods of Frame class

1. setResizable(Boolean b)
2. setMenuBar(MenuBar m)
3. setTitle(String s)
4. string getTitle()

Label

1. The object of Label class is a component for placing text in a container.
2. It is used to display a single line of read only text.
3. The text can be changed by an application but a user cannot edit it directly.
4. It is a passive component
5. Constructors
    a. Label() Constructs an empty label.
    b. Label(String text) Constructs a new label with the specified string of text, left justified.
    c. Label(String text, int alignment) Constructs a new label that presents the specified string of text with the specified alignment.
6. Methods
    a. void addNotify() : Creates the peer for this label.
    b. AccessibleContext getAccessibleContext() : Gets the AccessibleContext associated with this Label.
    c. int getAlignment() : Gets the current alignment of this label.

d. String getText() : Gets the text of this label.
e. protected String paramString() : Returns a string representing the state of this Label.
f. void setAlignment(int alignment) : Sets the alignment for this label to the specified alignment.
g. void setText(String text) : Sets the text for this label to the specified text.

Button

1. Button is a control component that has a label and generates an event when pressed.
2. Constructors
    a. Button() Constructs a button with an empty string for its label.
    b. Button(String text) Constructs a new button with specified label.
3. Methods
    a. void addActionListener(ActionListener l) : Adds the specified action listener to receive action events from this button.
    b. void addNotify() : Creates the peer of the button.
    c. AccessibleContext getAccessibleContext() : Gets the AccessibleContext associated with this Button.
    d. String getActionCommand() : Returns the command name of the action event fired by this button.
    e. ActionListener[] getActionListeners() : Returns an array of all the action listeners registered on this button.
    f. String getLabel() : Gets the label of this button.
    g. protected void processActionEvent(ActionEvent e) : Processes action events occurring on this button by dispatching them to any registered ActionListener objects.
    h. void removeActionListener(ActionListener l) : Removes the specified action listener so that it no longer receives action events from this button.
    i. void setActionCommand(String command) : Sets the command name for the action event fired by this button.
    j. void setLabel(String label) : Sets the button's label to be the specified string.

TextField

1. The textField component allows the user to edit single line of text.
2. Constructors
    a. TextField() : Constructs a new text field.
    b. TextField(int columns) : Constructs a new empty text field with the specified number of columns.
    c. TextField(String text) : Constructs a new text field initialized with the specified text.
    d. TextField(String text, int columns) : Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.
3. Methods
    a. void addActionListener(ActionListener l) : Adds the specified action listener to receive action events from this text field.
    b. void addNotify() : Creates the TextField's peer.
    c. boolean echoCharIsSet() : Indicates whether or not this text field has a character set for echoing.
    d. AccessibleContext getAccessibleContext() : Gets the AccessibleContext associated with this TextField.
    e. ActionListener[] getActionListeners() : Returns an array of all the action listeners registered on this textfield.
    f. int getColumns() : Gets the number of columns in this text field.
    g. char getEchoChar() : Gets the character that is to be used for echoing.
    h. Dimension getMinimumSize() :Gets the minumum dimensions for this text field.
    i. Dimension getMinimumSize(int columns) : Gets the minumum dimensions for a text field with the specified number of columns.
    j. Dimension getPreferredSize() : Gets the preferred size of this text field.
    k. Dimension getPreferredSize(int columns) : Gets the preferred size of this text field with the specified number of columns.
    l. protected String paramString() : Returns a string representing the state of this TextField.

m. protected void processActionEvent(ActionEvent e) : Processes action events occurring on this text field by dispatching them to any registered ActionListener objects.
n. protected void processEvent(AWTEvent e) : Processes events on this text field.
o. void removeActionListener(ActionListener l) : Removes the specified action listener so that it no longer receives action events from this text field.
p. void setColumns(int columns) : Sets the number of columns in this text field.
q. void setEchoChar(char c) : Sets the echo character for this text field.
r. void setText(String t) : Sets the text that is presented by this text component to be the specified text.

TextArea

1. The TextArea control in AWT provide us multiline editor area.
2. Constructors
    a. TextArea() : Constructs a new text area with the empty string as text.
    b. TextArea(int rows, int columns) : Constructs a new text area with the specified number of rows and columns and the empty string as text.
    c. TextArea(String text) : Constructs a new text area with the specified text.
    d. TextArea(String text, int rows, int columns) : Constructs a new text area with the specified text, and with the specified number of rows and columns.
    e. TextArea(String text, int rows, int columns, int scrollbars) : Constructs a new text area with the specified text, and with the rows, columns, and scroll bar visibility as specified.
3. Methods
    a. void addNotify() : Creates the TextArea's peer.
    b. void append(String str) : Appends the given text to the text area's current text.
    c. AccessibleContext getAccessibleContext() : Returns the AccessibleContext associated with this TextArea.
    d. int getColumns() : Returns the number of columns in this text area.

e. Dimension getMinimumSize() : Determines the minimum size of this text area.
f. Dimension getMinimumSize(int rows, int columns) : Determines the minimum size of a text area with the specified number of rows and columns.
g. Dimension getPreferredSize() : Determines the preferred size of this text area.
h. Dimension getPreferredSize(int rows, int columns) : Determines the preferred size of a text area with the specified number of rows and columns.
i. int getRows() : Returns the number of rows in the text area.
j. int getScrollbarVisibility() : Returns an enumerated value that indicates which scroll bars the text area uses.
k. void insert(String str, int pos) : Inserts the specified text at the specified position in this text area.
l. protected String paramString() : Returns a string representing the state of this TextArea.
m. void replaceRange(String str, int start, int end) : Replaces text between the indicated start and end positions with the specified replacement text.
n. void setColumns(int columns) : Sets the number of columns for this text area.
o. void setRows(int rows) : Sets the number of rows for this text area.

Checkbox

1. Checkbox control is used to turn an option on(true) or off(false).
2. There is label for each checkbox representing what the checkbox does.
3. The state of a checkbox can be changed by clicking on it.
4. Constructors
    a. Checkbox() : Creates a check box with an empty string for its label.
    b. Checkbox(String label) : Creates a check box with the specified label.
    c. Checkbox(String label, boolean state) : Creates a check box with the specified label and sets the specified state.

d. Checkbox(String label, boolean state, CheckboxGroup group) : Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
e. Checkbox(String label, CheckboxGroup group, boolean state) : Creates a check box with the specified label, in the specified check box group, and set to the specified state.

5. Methods
   a. void addItemListener(ItemListener l) : Adds the specified item listener to receive item events from this check box.
   b. void addNotify() : Creates the peer of the Checkbox.
   c. AccessibleContext getAccessibleContext() : Gets the AccessibleContext associated with this Checkbox.
   d. CheckboxGroup getCheckboxGroup() : Determines this check box's group.
   e. ItemListener[] getItemListeners() : Returns an array of all the item listeners registered on this checkbox.
   f. String getLabel() : Gets the label of this check box.
   g. Object[] getSelectedObjects() : Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
   h. boolean getState() : Determines whether this check box is in the on or off state.
   i. protected String paramString() : Returns a string representing the state of this Checkbox.
   j. protected void processEvent(AWTEvent e) : Processes events on this check box.
   k. protected void processItemEvent(ItemEvent e) : Processes item events occurring on this check box by dispatching them to any registered ItemListener objects.
   l. void removeItemListener(ItemListener l) : Removes the specified item listener so that the item listener no longer receives item events from this check box.
   m. void setCheckboxGroup(CheckboxGroup g) : Sets this check box's group to the specified check box group.

n. void setLabel(String label) : Sets this check box's label to be the string argument.
o. void setState(boolean state) : Sets the state of this check box to the specified state.

Checkboxgroup

1. The CheckboxGroup class is used to group the set of checkbox.
2. Constructor
    a. CheckboxGroup() : Creates a new instance of CheckboxGroup.
3. Methods
    a. Checkbox getSelectedCheckbox() : Gets the current choice from this check box group.
    b. void setSelectedCheckbox(Checkbox box) : Sets the currently selected check box in this group to be the specified check box.
    c. String toString() : Returns a string representation of this check box group, including the value of its current selection.

Choice

1. Choice control is used to show pop up menu of choices.
2. Selected choice is shown on the top of the menu.
3. Constructors
    a. Choice() : Creates a new choice menu.
4. Methods
    a. void add(String item) : Adds an item to this Choice menu.
    b. void addItem(String item) : Obsolete as of Java 2 platform v1.1.
    c. void addItemListener(ItemListener l) : Adds the specified item listener to receive item events from this Choice menu.
    d. void addNotify() : Creates the Choice's peer.
    e. int countItems() : Deprecated. As of JDK version 1.1, replaced by getItemCount().
    f. AccessibleContext getAccessibleContext() : Gets the AccessibleContext associated with this Choice.
    g. String getItem(int index) : Gets the string at the specified index in this Choice menu.

h. int getItemCount() : Returns the number of items in this Choice menu.
i. ItemListener[] getItemListeners() : Returns an array of all the item listeners registered on this choice.
j. int getSelectedIndex() : Returns the index of the currently selected item.
k. String getSelectedItem() : Gets a representation of the current choice as a string.
l. Object[] getSelectedObjects() : Returns an array (length 1) containing the currently selected item.
m. void insert(String item, int index) : Inserts the item into this choice at the specified position.
n. protected String paramString() : Returns a string representing the state of this Choice menu.
o. protected void processEvent(AWTEvent e) : Processes events on this choice.
p. protected void processItemEvent(ItemEvent e) :
q. Processes item events occurring on this Choice menu by dispatching them to any registered ItemListener objects.
r. void remove(int position) : Removes an item from the choice menu at the specified position.
s. void remove(String item) : Removes the first occurrence of item from the Choice menu.
t. void removeAll() : Removes all items from the choice menu.
u. void removeItemListener(ItemListener l) : Removes the specified item listener so that it no longer receives item events from this Choice menu.
v. void select(int pos) : Sets the selected item in this Choice menu to be the item at the specified position.
w. void select(String str) : Sets the selected item in this Choice menu to be the item whose name is equal to the specified string.

List

1. The List represents a list of text items. The list can be configured that user can choose either one item or multiple items.

2. Constructors

a. List() : Creates a new scrolling list.
b. List(int rows) : Creates a new scrolling list initialized with the specified number of visible lines.
c. List(int rows, boolean multipleMode) : Creates a new scrolling list initialized to display the specified number of rows.

3. Methods

a. void add(String item) : Adds the specified item to the end of scrolling list.
b. void add(String item, int index) : Adds the specified item to the the scrolling list at the position indicated by the index.
c. void addActionListener(ActionListener l) : Adds the specified action listener to receive action events from this list.
d. void addItemListener(ItemListener l) : Adds the specified item listener to receive item events from this list.
e. void addNotify() : Creates the peer for the list.
f. void deselect(int index) : Deselects the item at the specified index.
g. AccessibleContext getAccessibleContext() : Gets the AccessibleContext associated with this List.
h. ActionListener[] getActionListeners() : Returns an array of all the action listeners registered on this list.
i. String getItem(int index) : Gets the item associated with the specified index.
j. int getItemCount() : Gets the number of items in the list.
k. ItemListener[] getItemListeners() : Returns an array of all the item listeners registered on this list.
l. String[] getItems() : Gets the items in the list.
m. Dimension getMinimumSize() : Determines the minimum size of this scrolling list.
n. Dimension getMinimumSize(int rows) : Gets the minumum dimensions for a list with the specified number of rows.
o. Dimension getPreferredSize() : Gets the preferred size of this scrolling list.
p. Dimension getPreferredSize(int rows) : Gets the preferred dimensions for a list with the specified number of rows.

q. int getRows() : Gets the number of visible lines in this list.
r. int getSelectedIndex() : Gets the index of the selected item on the list,
s. int[] getSelectedIndexes() : Gets the selected indexes on the list.
t. String getSelectedItem() : Gets the selected item on this scrolling list.
u. String[] getSelectedItems() : Gets the selected items on this scrolling list.
v. Object[] getSelectedObjects() : Gets the selected items on this scrolling list in an array of Objects.
w. int getVisibleIndex() : Gets the index of the item that was last made visible by the method makeVisible.
x. boolean isIndexSelected(int index) : Determines if the specified item in this scrolling list is selected.
y. boolean isMultipleMode() : Determines whether this list allows multiple selections.
z. void makeVisible(int index) : Makes the item at the specified index visible.
aa. protected String paramString() : Returns the parameter string representing the state of this scrolling list.
bb. protected void processActionEvent(ActionEvent e) : Processes action events occurring on this component by dispatching them to any registered ActionListener objects.
cc. protected void processEvent(AWTEvent e) : Processes events on this scrolling list.
dd. protected void processItemEvent(ItemEvent e) : Processes item events occurring on this list by dispatching them to any registered ItemListener objects.
ee. void remove(int position) : Removes the item at the specified position from this scrolling list.
ff. void remove(String item) : Removes the first occurrence of an item from the list.
gg. void removeActionListener(ActionListener l) : Removes the specified action listener so that it no longer receives action events from this list.
hh. void removeAll() : Removes all items from this list.

ii. void removeItemListener(ItemListener l) : Removes the specified item listener so that it no longer receives item events from this list.

jj. void removeNotify() : Removes the peer for this list.

kk. void replaceItem(String newValue, int index) : Replaces the item at the specified index in the scrolling list with the new string.

ll. void select(int index) : Selects the item at the specified index in the scrolling list.

mm. void setMultipleMode(boolean b) : Sets the flag that determines whether this list allows multiple selections.

Program: 73

Wap to use of Frame.

```
import java.awt.*;
import java.awt.event.*;
public class mywin extends Frame implements ActionListener
{
Button ext;
Label lbl;
mywin()
{
ext=new Button("Exit");
lbl=new Label("Click Here To Close ");
setLayout(null);
lbl.setBounds(160,150,140,20);
ext.setBounds(200,180,60,20);
add(ext);
add(lbl);
ext.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
System.exit(0);
}
```

```
public static void main(String args[])
{
mywin ob=new mywin();
ob.setSize(800,800);
ob.setTitle("My Frame");
ob.setVisible(true);
ob.setBackground(Color.cyan);
}
}
```

Program: 74

Wap in java to change the Frame's background color red, green, blue depending on the button clicked.

```
import java.awt.*;
import java.awt.event.*;
public class rgb extends Frame implements ActionListener
{
Button b1,b2,b3,b4;
rgb()
{
b1=new Button("RED");
b2=new Button("GREEN");
b3=new Button("BLUE");
b4=new Button("Exit");
setLayout(null);
b1.setBounds(100,300,40,30);
b2.setBounds(160,300,40,30);
b3.setBounds(220,300,40,30);
b4.setBounds(160,340,30,20);
add(b1);add(b2);add(b3);add(b4);
b1.addActionListener(this);b2.addActionListener(this);
b3.addActionListener(this);b4.addActionListener(this);
}
```

```
public void actionPerformed(ActionEvent ae)
{
try{
String s=ae.getActionCommand();
if(s.equals("RED"))
setBackground(Color.red);
else if(s.equals("GREEN"))
setBackground(Color.green);
else if(s.equals("BLUE"))
setBackground(Color.blue);
else if(s.equals("Exit"))
System.exit(0);
}catch(Exception e)
{}
}
public static void main(String args[])
{
rgb oa=new rgb();
oa.setSize(500,500);
oa.setVisible(true);
}
}
```

Program: 75

Wap in java , on one frame perform addition 2 numbers by using label, textfield and button.

```
import java.awt.*;
import java.awt.event.*;
public class add2f extends Frame implements ActionListener
{
String str1=new String("");
String str2=new String("");
```

```
String str3=new String("");
TextField t;
Label l;
Button b1,b2,b3,b4,b5;
int b=0;
add2f()
{
l=new Label("Addition 2 number :");
t=new TextField();
b1=new Button("5");
b2=new Button("10");
b3=new Button("+");
b4=new Button("=");
b5=new Button("Exit");
add(t);add(l);add(b1);
add(b2);add(b3);add(b4);add(b5);
setLayout(null);
l.setBounds(10,30,200,30);
t.setBounds(10,70,150,30);
b1.setBounds(30,120,35,28);
b2.setBounds(30,170,35,28);
b3.setBounds(80,120,35,28);
b4.setBounds(80,170,35,28);
b5.setBounds(47,220,35,28);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
```

```
if(ae.getActionCommand()=="+")
{
str2=ae.getActionCommand();
t.setText(str2);
b=1;
}
else if(ae.getActionCommand()=="=")
{
int n1=Integer.parseInt(str1);
int n2=Integer.parseInt(str3);
int ans=0;
if(str2=="+")
ans=n1+n2;
t.setText(""+ans);
b=0;
str1="";
str3="";
}
else if(ae.getActionCommand()=="Exit")
{
System.exit(0);
}
else if(b==0)
{
str1=str1+ae.getActionCommand();
t.setText(str1);
}
else if(b==1)
{
str3=str3+ae.getActionCommand();
t.setText(str3);
}
```

```
}
public static void main(String args[])
{
add2f oc=new add2f();
oc.setBackground(Color.blue);
oc.setTitle("Addition");
oc.setBounds(140,140,180,280);
oc.setVisible(true);
}
}
```

Program: 76

Wap in java to Design of Calculator.

```
import java.awt.*;
import java.awt.event.*;
public class calcu extends Frame implements ActionListener
{
String str1=new String("");
String str2=new String("");
String str3=new String("");
TextField t;
int b=0;
Button b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15,b16,b17,b18;
calcu()
{
t=new TextField();
b1=new Button("1");
b2=new Button("2");
b3=new Button("3");
b4=new Button("4");
b5=new Button("5");
b6=new Button("6");
b7=new Button("7");
```

```
b8=new Button("8");
b9=new Button("9");
b10=new Button("0");
b11=new Button(".");
b12=new Button("+");
b13=new Button("-");
b14=new Button("*");
b15=new Button("/");
b16=new Button("=");
b17=new Button("Clear");
b18=new Button("Close");
add(t);add(b1);add(b2);add(b3);add(b4);add(b5);add(b6);add(b7);add(b8);add(b9);
add(b10);add(b11);add(b12);add(b13);add(b14);add(b15);add(b16);add(b17);add(b18);
setLayout(null);
t.setBounds(10,30,190,30);
b1.setBounds(10,120,35,28);
b2.setBounds(50,120,35,28);
b3.setBounds(90,120,35,28);
b4.setBounds(130,120,35,28);
b5.setBounds(10,153,35,28);
b6.setBounds(50,153,35,28);
b7.setBounds(90,153,35,28);
b8.setBounds(130,153,35,28);
b9.setBounds(10,186,35,28);
b10.setBounds(50,186,35,28);
b11.setBounds(90,186,35,28);
b12.setBounds(130,186,35,28);
b13.setBounds(10,219,35,28);
b14.setBounds(50,219,35,28);
b15.setBounds(90,219,35,28);
b16.setBounds(130,219,35,28);
b17.setBounds(165,70,35,28);
```

```
b18.setBounds(165,265,35,28);
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
b6.addActionListener(this);
b7.addActionListener(this);
b8.addActionListener(this);
b9.addActionListener(this);
b10.addActionListener(this);
b11.addActionListener(this);
b12.addActionListener(this);
b13.addActionListener(this);
b14.addActionListener(this);
b15.addActionListener(this);
b16.addActionListener(this);
b17.addActionListener(this);
b18.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
if(ae.getActionCommand()=="+")
{
str2=ae.getActionCommand();
t.setText(str2);
b=1;
}
else if(ae.getActionCommand()=="-")
{
str2=ae.getActionCommand();
t.setText(str2);
```

```
b=1;
}
else if(ae.getActionCommand()=="*")
{
str2=ae.getActionCommand();
t.setText(str2);
b=1;
}
else if(ae.getActionCommand()=="/")
{
str2=ae.getActionCommand();
t.setText(str2);
b=1;
}
else if(ae.getActionCommand()=="=")
{
int n1=Integer.parseInt(str1);
int n2=Integer.parseInt(str3);
int ans=0;
if(str2=="+")
ans=n1+n2;
else if(str2=="-")
ans=n1-n2;
else if(str2=="*")
ans=n1*n2;
else if(str2=="/")
ans=n1/n2;
t.setText(""+ans);
b=0;
str1="";
str3="";
}
```

```
else if(ae.getActionCommand()=="Clear")
{
t.setText(null);
b=0;
str1="";
str3="";
}
else if(ae.getActionCommand()=="Close")
{
System.exit(0);
}
else if(b==0)
{
str1=str1+ae.getActionCommand();
t.setText(str1);
}
else if(b==1)
{
str3=str3+ae.getActionCommand();
t.setText(str3);
}
}
public static void main(String args[])
{
calcu oc=new calcu();
oc.setBackground(Color.pink);
oc.setTitle("MyCalculator");
oc.setBounds(140,140,210,300);
oc.setVisible(true);
}
}
```

Program: 77

Write a program **in java to** use of checkbox.

Program-

```
import java.awt.*;

public class checkb extends Frame

{

 Label l;

 Checkbox
 c1,c2,c3,c4;
 checkb()
{

 l=new Label("Skills");
 c1=new
 Checkbox("C",true);
 c2=new
Checkbox("C++",true); c3=new
Checkbox("Java",true); c4=new
Checkbox("Oracle",false);
setLayout(null);
l.setBounds(150,30,80,60);
c1.setBounds(40,80,40,30);
c2.setBounds(90,80,40,30);

 c3.setBounds(140,80,40,30);
c4.setBounds(190,80,50,30);
add(l);add(c1);add(c2);add(c3);

add(c4);
}

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

```
{
 checkb oa=new checkb();
 oa.setSize(400,200);
 oa.setBackground(Color.yel
 low); oa.setVisible(true);
}
}
```

Program: 78

Write a program in java to use of RadioButton.

Program-

```
import java.awt.*;
public class radiob extends Frame
{
 Label l;
 Checkbo
 x c1,c2;
 CheckboxGr
 oup cg;
 radiob()
{
 cg=new CheckboxGroup();

 l=new Label("Gender:");
 c1=new Checkbox("Male",cg,true);
c2=new
Checkbox("Female",cg,false);
setLayout(null);
```

```
l.setBounds(40,30,80,60);
c1.setBounds(40,80,60,30);
c2.setBounds(100,80,60,30);
add(l);add(c1);add(c2);
}

public static void main(String args[])

{

 radiob oa=new radiob();
 oa.setSize(400,200);
 oa.setBackground(Color.yel
 low); oa.setVisible(true);
}

}
```

Program: 79

Write a Program in java to use of ChoiceBox

```
import java.awt.*;

public class choice extends Frame

{

Label l;

Choice c;

Choice()

{

 l=new
Label("Employee");
c=new Choice();
c.add("Manager");
```

```
 c.add("Dy Manager");
c.add("Asst Manager");
c.add("Cleark");

c.add("Sweper");

c.add("Gardener");
setLayout(null);
l.setBounds(40,30,80,40);
c.setBounds(40,80,80,80);

add(l);add(c);

}

public static void main(String args[])

{

 choice oa=new choice();
 oa.setSize(400,200);
 oa.setBackground(Color.yellow);
 oa.setVisible(true);

}

}
```

Program: 80

Wap in java to rotate a line about own of its end point.

```
import java.awt.*;
import java.awt.event.*;
public class roline extends Frame implements ActionListener
{
Button ext;
roline()
{
ext=new Button("close");
```

```
setLayout(null);
ext.setBounds(100,500,30,20);
add(ext);
ext.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
System.exit(0);
}
public void paint(Graphics g)
{
for(int i=0;i<=800;i++)
{
double j=i*Math.PI/360;
int x=(int)(150+100*Math.sin(j));
int y=(int)(200-100*Math.cos(j));
g.setColor(Color.red);
g.drawLine(150,200,x,y);
try{
Thread.sleep(50);
}catch(Exception e)
{}
g.setColor(Color.cyan);
g.drawLine(150,200,x,y);
}
}
public static void main(String args[])
{
roline ob=new roline();
ob.setSize(600,600);
ob.setTitle("My Frame");
ob.setVisible(true);
```

```
ob.setBackground(Color.cyan);
}
}
```

Delegation Event Model

The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a source generates an event and sends it to one or more listeners. In this scheme, the listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns. The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to "delegate" the processing of an event to a separate piece of code.

The event model is based on the Event Source and Event Listeners. Event Listener is an object that receives the messages / events. The Event Source is any object which creates the message / event. The Event Delegation model is based on – The Event Classes, The Event Listeners, Event Objects.

There are three participants in event delegation model in Java;

- Event Source – the class which broadcasts the events
- Event Listeners – the classes which receive notifications of events
- Event Object – the class object which describes the event.

An event occurs (like mouse click, key press, etc) which is followed by the event is broadcasted by the event source by invoking an agreed method on all event listeners. The event object is passed as argument to the agreed-upon method. Later the event listeners respond as they fit, like submit a form, displaying a message / alert etc.

Adapator Class

It is used in event handling

It contains the handler methods just like a handler interface.

A class which wants to act as a handler class can implement the corresponding handler interface or can extend the corresponding adaptor class.

Unlike handler interface, all the methods here in an adaptor class are complete(blank body).

Examples

MouseAdaptor

KeyAdaptor

WindowAdaptor

ComponenetAdaptor

ContainerAdaptor

MouseMotionAdaptor

Program: 81

A program to explain Adapater class

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class AdapterExample extends JFrame
{
    AdapterExample()
    {
      this.addWindowListener( new WindowAdapter() {
      public void windowClosing(WindowEvent e)
                {
                  System.exit(0);
                }
```

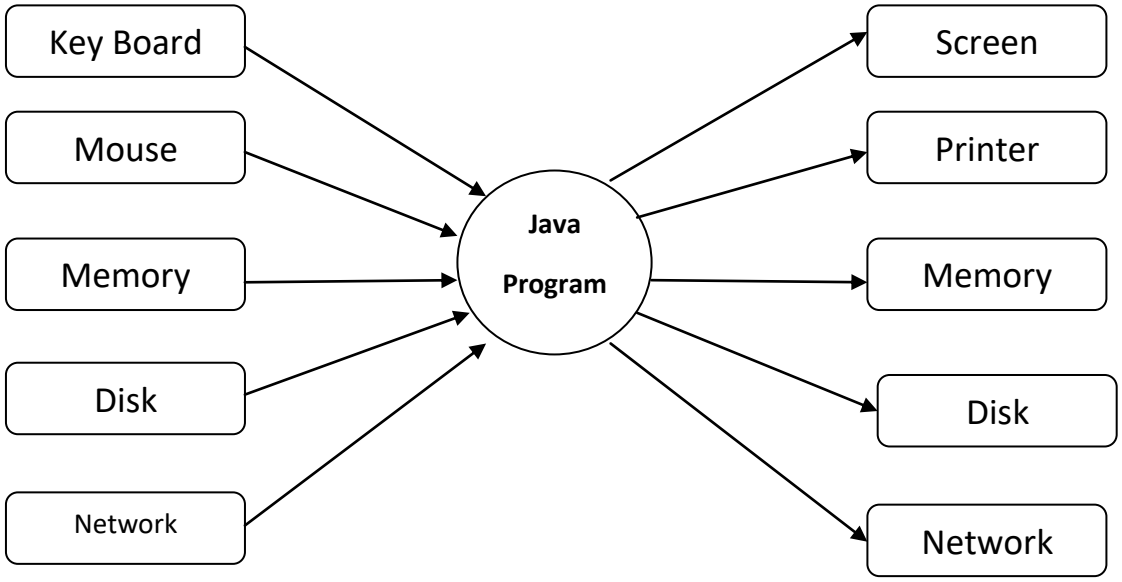
```
                    });
        }
}
   class AdapterClassJavaExample
{
    public static void main(String [] args)
    {
      AdapterExample frame = new AdapterExample();
      frame.setTitle("Adapter Class Java Example");
      frame.setBounds(100,200,200,200);
      frame.setVisible(true);
    }
}
```

Stream in Java

1. A stream is a logical entity that either produces or consumes information.
2. A stream is a java path along which data flows.
3. A stream is linked to a physical device by the Java I/O system.
4. All streams behave in the same manner, even if the actual physical devices to which they are linked differ.
5. An input stream can abstract many different kinds of input from a disk file, a keyboard or a network socket.
6. An output stream may refer to the console, a disk file or a network connection.
7. Java implements streams within class hierarchies defined in the java.io package.
8. A stream presents a uniform, easy to use, object oriented interface between the program and input/output devices.

SOURCES

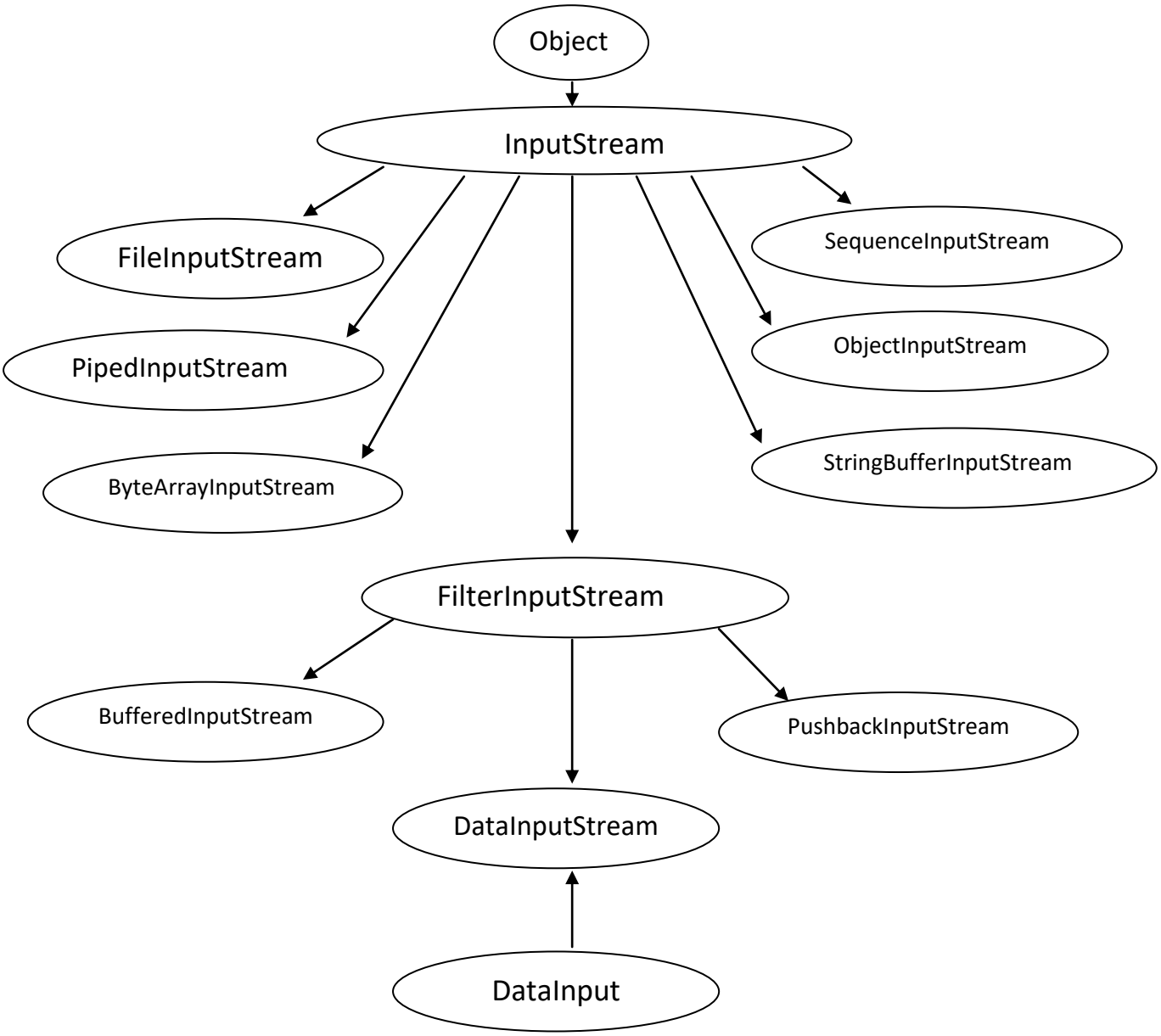
DESTINATIONS



9. Java defines 2 types of streams.
    a. Byte Stream
    b. Character Stream
10. Byte Stream – It provides a convenient means for handling input and output of bytes.

11. Byte streams are defined by using 2 class hierarchies. At the top, the 2 abstract classes : InputStream and OutputStream .
12. ByteStream classes are used for reading and writing binary data.
13. Each of these abstract classes has several concrete subclasses, that handle the different devices such as disk files, network connections and memory files.

Input Stream classes



Methods of InputStream class

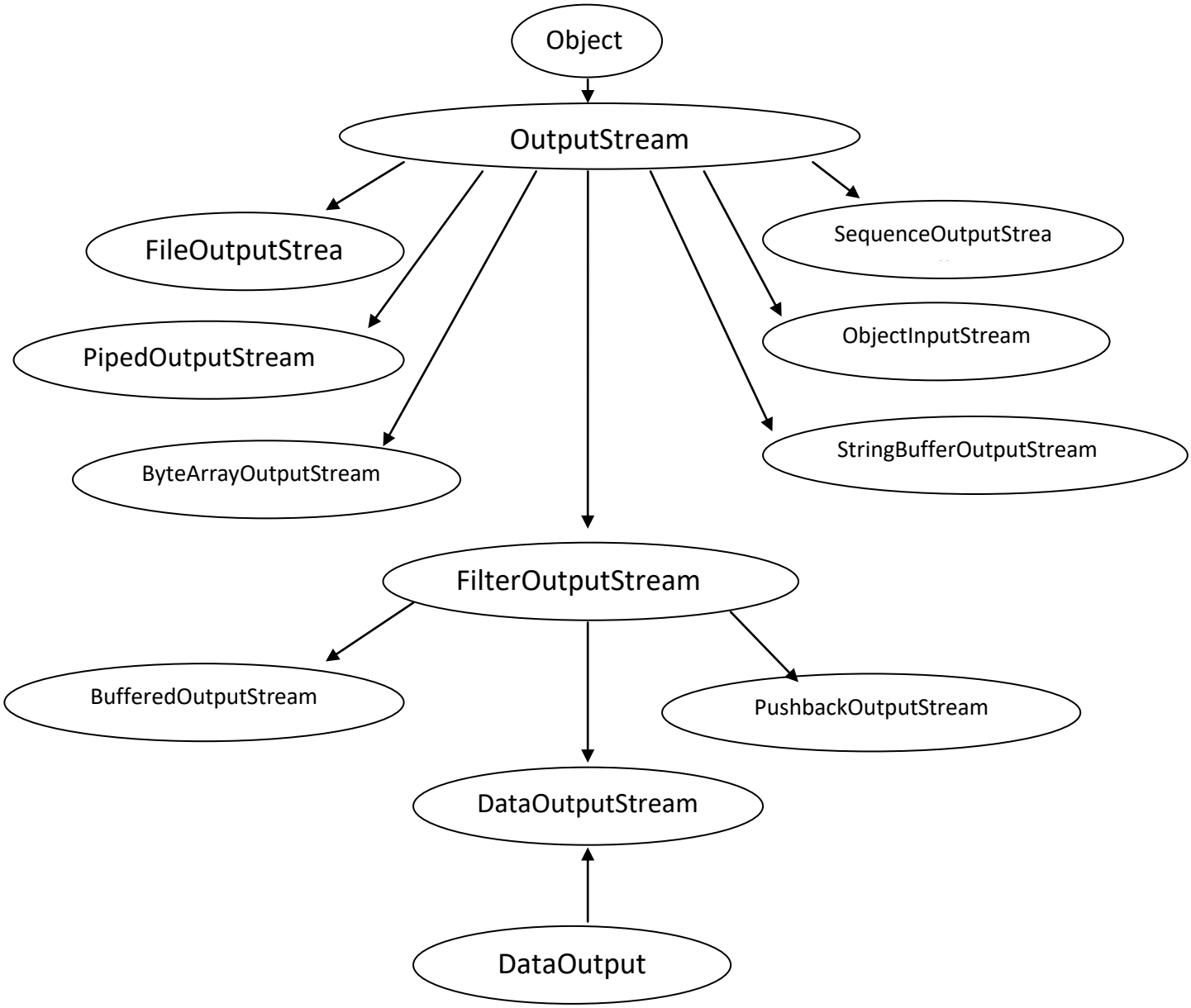
1. read() : Reads a byte from the input stream
2. read(byte b[]) : Reads an array of bytes into b.
3. read(byte b[], int n, int m) : Reads m bytes into b starting from nth byte.

4. available() : Gives number of bytes available in the input.
5. skip(n) : Skips over n bytes from the input stream.
6. reset(): Goes back to the beginning of the stream.
7. close() : Closes the input stream.

Methods of DataInput interface

1. readInt()
2. readShort()
3. readLong()
4. readFloat()
5. readUTF()
6. readDouble()
7. readLine()
8. readChar()
9. readBoolean()

Output Stream classes



Methods of OutputStream class

1. write() : Writes a byte to the Output stream.
2. Write(byte[] b) : Writes all bytes in the array to the output stream.
3. Write(byte b[], int n, int m) : Writes m bytes from array b starting from n^{th} byte.
4. Close() : Closes the output stream.
5. Flush() : Flushes the output stream.

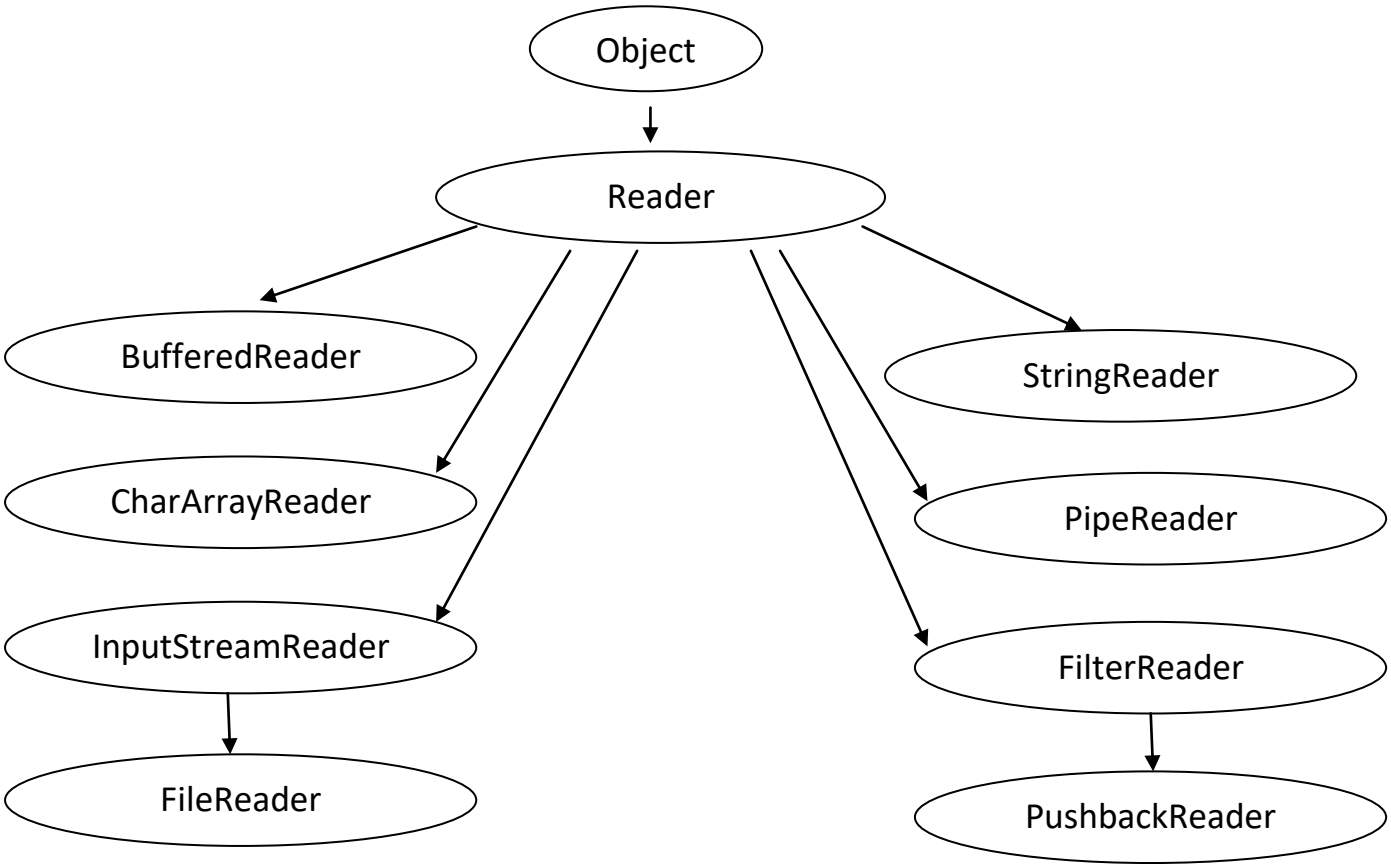
Methods of DataOutput interface

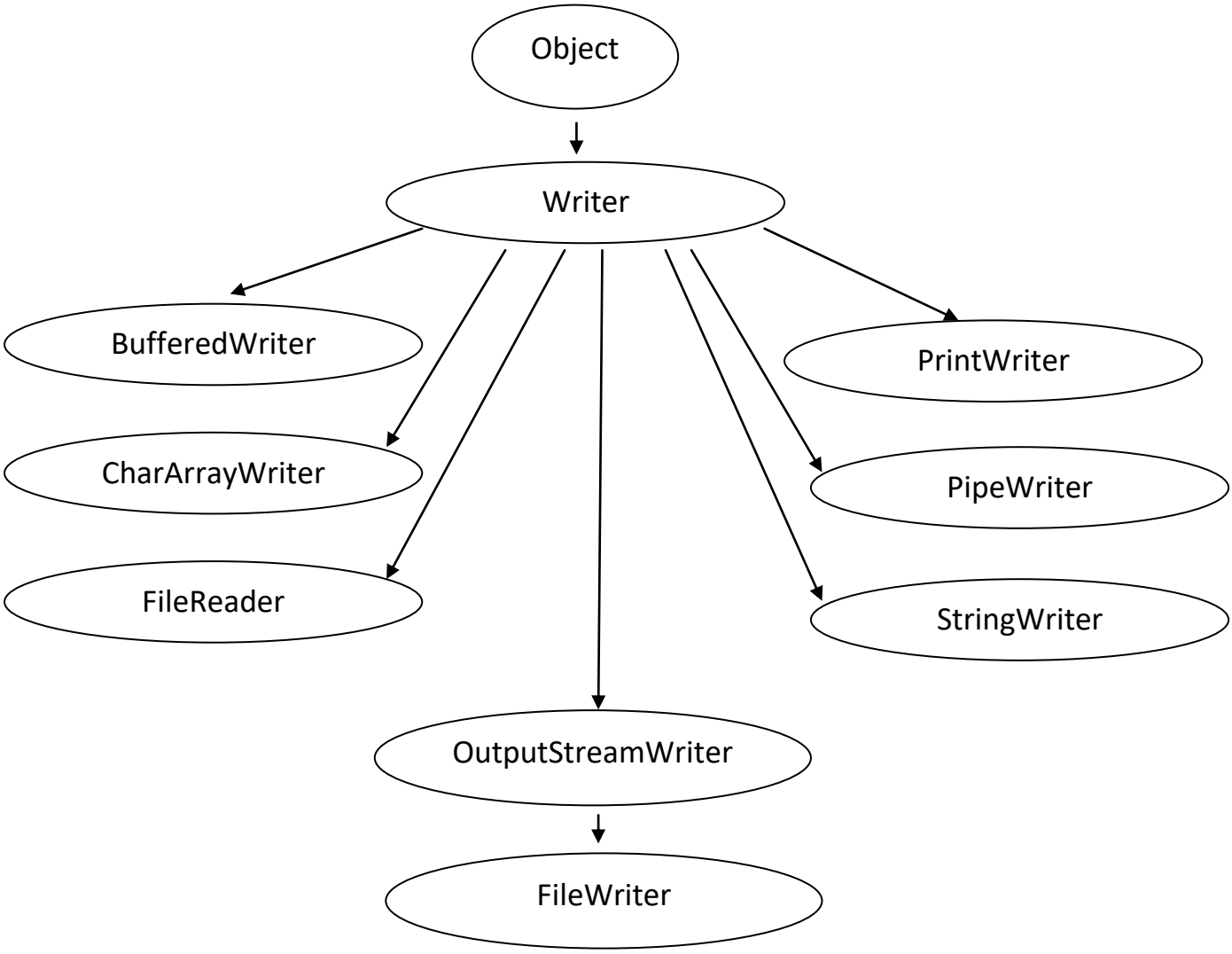
1. wruteShort()

2. writeInt()
3. writeLong()
4. writeFloat()
5. writeUTF()
6. writeDouble()
7. writeByte()
8. writeChar()
9. writeBoolean()

Character Stream classes

1. Character stream classes are used to read and write 16 bit Unicode character.
2. At the top, there are 2 abstract classes : Reader and Writer





The Predefines Streams

1. System class of java.lang package contains 3 predefined stream variables, in, out and err.
2. These are public and static in System.
3. System.out refers to the standard output by default this is console.
4. System.in refers to standard input, which is the keyboard by default.

Program: 82

Reading characters

```
Import java.io.*;
class br_read
{
public static void main (String args[]) throws IOException
{
char c;
BufferedReader br=new BufferedReader(new InputStreamReader (System.in);
System.out.println("Enter character 'q' to quit");
do
{
c=(char) br.read();
System.out.println(c);
}while(c != q);
}
}
```

File

1. Although most of the classes defined by java.io operate on streams, the file class deals directly with files and the file system.
2. A File object is used to obtain or manipulate the information associated with a disk file such as the permissions, time, date and directory path and to navigate subdirectory hierarchy.
3. A directory in java is treated simply as a file with one additional property – a list of filenames can be examined by list() method.
4. Constructors:
    a. File(String directory_path)
    b. File(String directory_path, String file_name)
    c. File(File File directory_obj, String file_name)
    d. File(URI uriobj)
5. Methods:
    a. getName()
    b. getParent()
    c. getPath()
    d. getAbsolutePath()
    e. exists()
    f. canWrite()
    g. canRead()
    h. isDirectory()
    i. isFile()
    j. isAbsolute()
    k. lastModified()
    l. length()
    m. renameTo(File newname)
    n. Boolean delete()
    o. deleteOnExit()
    p. isHidden()
    q. setLatModified(long millisecond)
    r. boolean setReadOnly()
    s. compaTo()

Directories

1. A directory is a File that contains a list of other files and directories.
2. When we create a file object and it is a directory, the isDirectory() method will return true. We can call list() on that object to extract list of the files and directories inside.

Program : 83

Program do display files and directories inside a folder

```
import java.io.File;
class dirlist
{
Public static void main(String args[])
{
String dirname=”C:/imit”);
File f1=new file(dirname);
if(f1.isDirectory())
{
System.out.println(“Directory of  “+dirname);
String s[]=f1.list();
for(int i=0;i<s.length; i++)
{
File f=new File(dirname+”/”+s[i]);
if(f.isDirectory())
{
System.out.println(s[i] + “is a directory”);
}
else
{
System.out.println(s[i]+”is a file”);
}
}
}
else
{
System.out.println(dirname+” is not a directory”);
}
}
}
```

Reading and writing characters in a file

1. The subclasses of Reader and Writer implement streams that can handle characters.
2. The two subclasses used for handling characters in files are FileReader and FileWriter.

Program: 84

Copying characters from one file into another file

```
import java.io.*;
class copychar
{
Public static void main(String args[])
{
File infile = new File(“input.dat”);
File outfile=new File(“output.dat”);
FileReader ins=null;
FileWriter outs=null;
try
{
ins=new FileReader(infile);
outs=new FileWriter(outfile);
int ch;
while((ch=ins.read())!=-1)
{
outs.write(ch);
}
}catch(IOException e)
{
System.out.println(e);
System.exit(-1);
}
finally
{
try
{
ins.close();
outs.close();
}catch(IOException e)
{
}
}
}
}
```

Program: 85

Copying bytes from one file to another

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

```
FileInputStream ins=null;
FileOutputStream outs=null;
byte byteread;
try
{
ins=new FileInputStream(“in.dat”);
outs=new FileOutputStream(“out.dat”);
do
{
byteread=(byte) ins.read();
Outs.write(byteread);
}while(byteread!=-1)
}catch(FileNotFoundException e)
{
System.out.println(“File Not Found”);
}
Catch(IOException e)
{
System.out.println(e.getMessage());
}
finally
{
try
{
ins.close();
outs.close();
}catch(IOException e)
{
}
}
}
}
```

Program:86

Reading and Writing Primitive data

```
import java.io.*;
class readwriteprimitive
{
Public static void main(String args[]) throws IOException
{
File primitive= new File(“prim.dat”);
FileOutputStream fos=new FileOutputStream(primitive);
DataOutputStream dos=new DataOutPutStream(fos);
```

```
dos.writeInt(1999);
dos.writeDouble(1999.4);
dos.writeBoolean(false);
dos.writeChar('f');
dos.close();
fos.close();
FileInputStream fis=new FileInputStream(primitive);
DataInputStream dis=new DataInputStream(fis);
System.out.println(dis.readInt());
System.out.println(dis.readDouble ());
System.out.println(dis.readBoolean());
System.out.println(dis.readChar());
dis.close();
fis.close();
}
}
```

Program: 87

Wap in Java for create your own 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...");
    }
}
```

Program: 88

Wap in Java for synchronization of Threads.

```
class Table{
   void printTable(int n){
     for(int i=1;i<=5;i++){
       System.out.println(n*i);
       try{
        Thread.sleep(400);
```

```
    }catch(Exception e){System.out.println(e);}
  }


 }
}


class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}


}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
```

```
}
public void run(){
t.printTable(100);
}
}

class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

SWING

Unlike AWT, Java Swing provides lightweight components. The javax.swing package provides classes for java swing API such as

1. JApplet
2. JFrame
3. JButton
4. JLabel
5. JTextField
6. JTextArea
7. JRadioButton
8. JCheckbox
9. JCombobox
10. JScrollpane
11. JToolbar
12. JTable
13. JSlider
14. JTree
15. JMenu
16. JColorChooser

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

Sl. No.	AWT	Swing
1	AWT components are heavyweight	Swing components are lightweight
2	AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
3	AWT provides less components than Swing	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
4	AWT doesn't follows MVC.	Swing follows MVC (Model View

		Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.

Java Swing Examples

There are two ways to create a frame:

By creating the object of Frame class (association)

By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

Program:89

File: FirstSwingExample.java

```
import javax.swing.*;

public class FirstSwingExample

{

public static void main(String[] args)

 {

JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton

b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame
```

```
f.setSize(400,500);//400 width and 500 height

f.setLayout(null);//using no layout managers

f.setVisible(true);//making the frame visible

}

}
```

Program: 90

Ssimple example - 2 of java swing

Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
import javax.swing.*;

public class Simple

{

JFrame f;

Simple()

{

f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton

b.setBounds(130,100,100, 40);

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height

f.setLayout(null);//using no layout managers

f.setVisible(true);//making the frame visible
```

```
}

public static void main(String[] args) {

new Simple();

}

}
```

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

Program: 91

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
import javax.swing.*;

public class Simple2 extends JFrame{//inheriting JFrame

JFrame f;

Simple2(){

JButton b=new JButton("click");//create button

b.setBounds(130,100,100, 40);

add(b);//adding button on frame

setSize(400,500);

setLayout(null);

setVisible(true);

}

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

```
new Simple2();
}
}
```

Program:92

Write a program in java swing to show the names of countries on clicking the corresponding flag.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
//<applet code=JButtonDemo" width=250 height=300></applet>
public class JButtonDemo extends JApplet implements ActionListener
{
JTextField jtf;
public void int()
{
Container cp=getContentPane();
cp=setLayout(new FlowLayout);
ImageIcon India=new ImageIcon("India.gif");
JButton jb1=new JButton(India);
jb1.setActionCommand("India");
jb1.addActionListener(this);
cp.add(jb1);

ImageIcon USA=new ImageIcon("USA.gif");
```

```
JButton jb2=new JButton(USA);
Jb2.setActionCommand(“USA”);
Jb2.addActionListener(this);
cp.add(jb2);
ImageIcon France =new ImageIcon(“France.gif”);
JButton jb3=new JButton(France);
Jb3.setActionCommand(“France”);
Jb3.addActionListener(this);
cp.add(jb3);

ImageIcon Germany =new ImageIcon(“Germany.gif”);
JButton jb4=new JButton (Germany);
Jb4.setActionCommand(“Germany”);
Jb4.addActionListener(this);
cp.add(jb4);

jtf= new JTextField(15);
cp.add(jtf);
}
Public void actionPerformed(ActionEvent ae)
{
Jtf.setText(ae.getActionCommand());
}
}
```

Program: 93

Write a sample program of swing in Java.

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
public class exp1 extends JFrame
{
	public exp1()
	{
		setTitle("Simple Example");
		setSize(300,200);
		setLocationRelativeTo(null);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
	}
	public static void main(String s[])
	{
		exp1 e=new exp1();
		e.setVisible(true);
	}
}
```

String in java

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

a) By string literal
b) By new keyword

1**) String Literal**

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

String s="welcome";

Each time we 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 the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

String s1="Welcome";

String s2="Welcome";//It doesn't create a new instance

2) By new keyword

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 a heap (non-pool).

Program: 94

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);

}}
```

Output:

```
java

strings

example
```

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

Sl. No. Method Description

1. char charAt(int index) - returns char value for the particular index

2. int length() - returns string length

3. static String format(String format, Object... args) - returns a formatted string.

4. static String format(Locale l, String format, Object... args) - returns formatted string with given locale.

5. String substring(int beginIndex) - returns substring for given begin index.

6. String substring(int beginIndex, int endIndex) - returns substring for given begin index and end index.

7. boolean contains(CharSequence s) - returns true or false after matching the sequence of char value.

8. static String join(CharSequence delimiter, CharSequence... elements) - returns a joined string.

9. static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) - returns a joined string.

10. boolean equals(Object another) - checks the equality of string with the given object.

11. boolean isEmpty() - checks if string is empty.

12. String concat(String str) - concatenates the specified string.

13. String replace(char old, char new) - replaces all occurrences of the specified char value.

14. String replace(CharSequence old, CharSequence new) - replaces all occurrences of the specified CharSequence.

15. static String equalsIgnoreCase(String another) - compares another string. It doesn't check case.

16. String[] split(String regex) - returns a split string matching regex.

17. String[] split(String regex, int limit) - returns a split string matching regex and limit.

18. String intern() - returns an interned string.

19. int indexOf(int ch) - returns the specified char value index.

20. int indexOf(int ch, int fromIndex) - returns the specified char value index starting with given index.

21. int indexOf(String substring) - returns the specified substring index.

22. int indexOf(String substring, int fromIndex) - returns the specified substring index starting with given index.

23. String toLowerCase() - returns a string in lowercase.

24. String toLowerCase(Locale l) - returns a string in lowercase using specified locale.

25. String toUpperCase() - returns a string in uppercase.

26. String toUpperCase(Locale l) - returns a string in uppercase using specified locale.

27. String trim() - removes beginning and ending spaces of this string.

28. static String valueOf(int value) - converts given type into string. It is an overloaded method.

StringBuffer class

JStringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed. Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

Important Constructors of StringBuffer class

a) StringBuffer() - creates an empty string buffer with the initial capacity of 16.
b) StringBuffer(String str) - creates a string buffer with the specified string.
c) StringBuffer(int capacity) - creates an empty string buffer with the specified capacity as length.

Important methods of StringBuffer class

1. public synchronized StringBuffer append(String s) - is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
2. public synchronized StringBuffer insert(int offset, String s) - is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
3. public synchronized StringBuffer replace(int startIndex, int endIndex, String str) - is used to replace the string from specified startIndex and endIndex.
4. public synchronized StringBuffer delete(int startIndex, int endIndex) - is used to delete the string from specified startIndex and endIndex.
5. public synchronized StringBuffer reverse() - is used to reverse the string.
6. public int capacity() - is used to return the current capacity.
7. public void ensureCapacity(int minimumCapacity) - is used to ensure the capacity at least equal to the given minimum.
8. public char charAt(int index) - is used to return the character at the specified position.

9. public int length() - is used to return the length of the string i.e. total number of characters.
10. public String substring(int beginIndex) - is used to return the substring from the specified beginIndex.
11. public String substring(int beginIndex, int endIndex) - is used to return the substring from the specified beginIndex and endIndex.

What is mutable string?

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

Program:95

```
class StringBufferExample{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}
```

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

Program:96

```
class StringBufferExample2{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}

}
```

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

Program:97

```
class StringBufferExample3{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo

}

}
```

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

Program:98

```
class StringBufferExample4{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}
```

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

Program:99

```
class StringBufferExample5{

public static void main(String args[]){

StringBuffer sb=new StringBuffer("Hello");

sb.reverse();

System.out.println(sb);//prints olleH

}

}
```

6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

Program:100

```
class StringBufferExample6{

public static void main(String args[]){

StringBuffer sb=new StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2

}

}
```

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

Program:101

```
class StringBufferExample7{
```

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

StringBuffer sb=new StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16*2)+2=34 i.e (oldcapacity*2)+2

sb.ensureCapacity(10);//now no change

System.out.println(sb.capacity());//now 34

sb.ensureCapacity(50);//now (34*2)+2

System.out.println(sb.capacity());//now 70

}
```

JavaBean

A JavaBean is a Java class that should follow the following conventions:

a) It should have a no-arg constructor.
b) It should be Serializable.
c) It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

It is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

Example

```
//Employee.java

 package mypack;

public class Employee implements java.io.Serializable{

private int id;

private String name;

public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}

public void setName(String name){this.name=name;}

public String getName(){return name;}

}
```

How to access the JavaBean class?

To access the JavaBean class, we should use getter and setter methods.

Program:102

```
package mypack;

public class Test{

public static void main(String args[]){

Employee e=new Employee();//object is created

e.setName("Arjun");//setting value to the object

System.out.println(e.getName());

}}
```

JavaBean Properties

A JavaBean property is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

1. getPropertyName ()

For example, if the property name is firstName, the method name would be getFirstName() to read that property. This method is called the accessor.

2. setPropertyName ()

For example, if the property name is firstName, the method name would be setFirstName() to write that property. This method is called the mutator.

Advantages of JavaBean

The following are the advantages of JavaBean.

a) The JavaBean properties and methods can be exposed to another application.
b) It provides an easiness to reuse the software components.

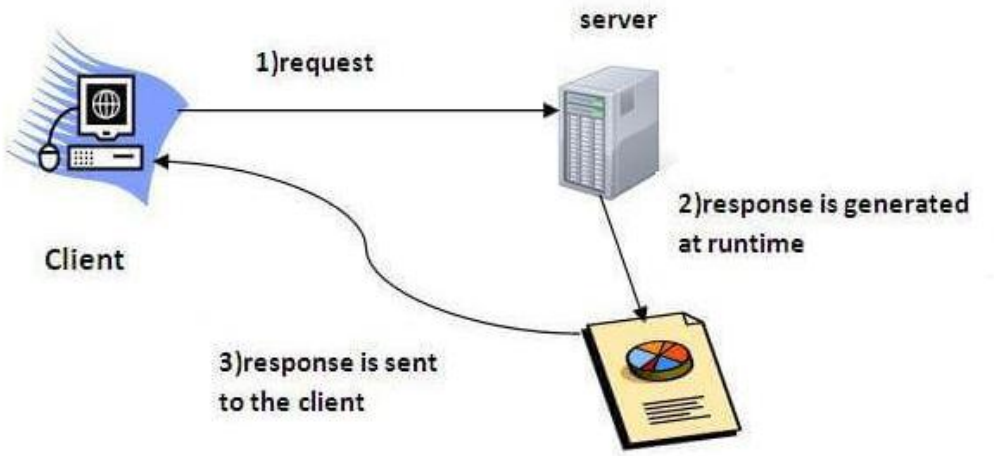
Disadvantages of JavaBean

The following are the disadvantages of JavaBean:

a) JavaBeans are mutable. So, it can't take advantages of immutable objects.
b) Creating the setter and getter method for each property separately may lead to the boilerplate code.

Servlet

1. Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).
2. Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
3. There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.
4. Servlet is a technology which is used to create a web application.
5. Servlet is an API that provides many interfaces and classes including documentation.
6. Servlet is an interface that must be implemented for creating any Servlet.
7. Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
8. Servlet is a web component that is deployed on the server to create a dynamic web page.

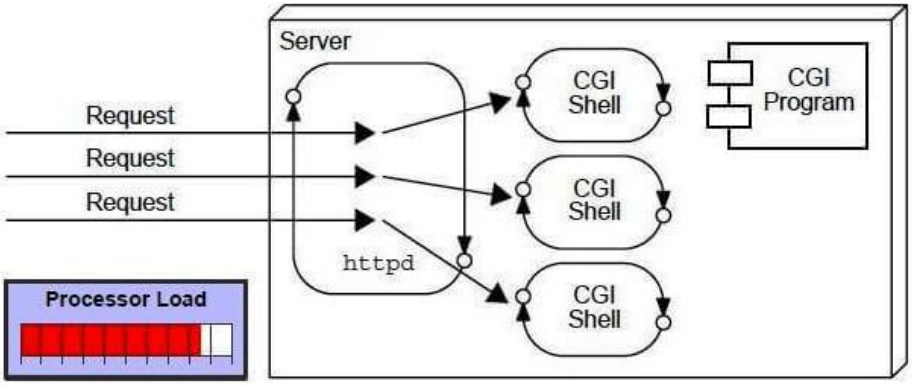


Web application

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

CGI (Common Gateway Interface)

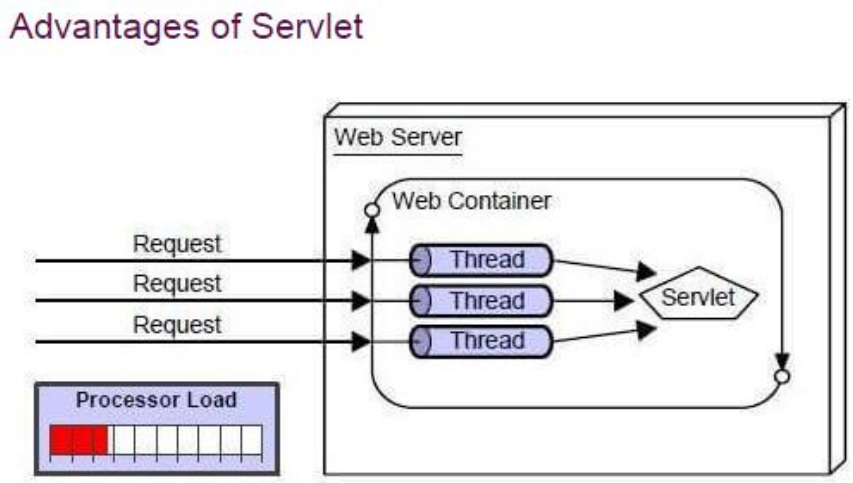
CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.



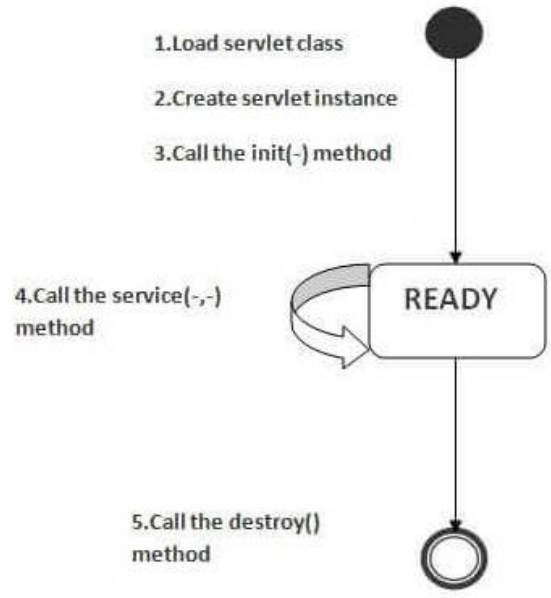
There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. Better performance: because it creates a thread for each request, not process.
2. Portability: because it uses Java language.
3. Robust: JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. Secure: because it uses java language.

Life Cycle of a Servlet (Servlet Life Cycle)

1. Life Cycle of a Servlet
2. Servlet class is loaded
3. Servlet instance is created
4. init method is invoked
5. service method is invoked
6. destroy method is invoked

Life cycle of a servlet



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1**) Servlet class is loaded**

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

public void init(ServletConfig config) throws ServletException

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

Program: 103

Wap in Java for servlet.

```
import javax.servlet.*;

import javax.servlet.http.*;

public class sdemo extends HttpServlet

{

    int c=0;

    public void service(HttpServlet req, HttpServlet res) throws ServletException, IOException

    {

        c++;

        PrintWriter ob=res.getWriter();

        ob.println("<HTML><HEAD><TITLE>MY FIRSTDYNAMIC");

        ob.println("PAGE</TITLE></HEAD><BODY><BR><HR>");

        ob.println("<H1><CENTER>WELCOME GUEST<BR>");

        ob.println("UR VISITOR NO<FONT COLOR=RED>");

        ob.println("</FONT></CENTER></H1><BR><HR>");

        ob.println("</BODY></HTML>");
```

```
        }
}
```

Servlet Program

Servlets are Java classes which service HTTP requests and implement the javax.servlet.Servlet interface. Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Program:104

Servlet Sample Code

Following is the sample source code structure of a servlet example to show Hello World –

```
// Import required java libraries

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

// Extend HttpServlet class

public class HelloWorld extends HttpServlet {

private String message;

public void init() throws ServletException {

    // Do required initialization

    message = "Hello World";
```

```
  }

 public void doGet(HttpServletRequest request, HttpServletResponse response)

     throws ServletException, IOException {

     // Set response content type

     response.setContentType("text/html");

     // Actual logic goes here.

     PrintWriter out = response.getWriter();

     out.println("<h1>" + message + "</h1>");

  }

 public void destroy() {

     // do nothing.

  }

}
```

Compiling a Servlet

Let us create a file with name HelloWorld.java with the code shown above. Place this file at C:\ServletDevel (in Windows) or at /usr/ServletDevel (in Unix). This path location must be added to CLASSPATH before proceeding further.

Assuming your environment is setup properly, go in ServletDevel directory and compile HelloWorld.java as follows −

```
$ javac HelloWorld.java
```

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. I have included only servlet-api.jar JAR file because I'm not using any other library in Hello World program.

This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, you have to include the location of the Java SDK that you are using in the PATH environment variable.

If everything goes fine, above compilation would produce HelloWorld.class file in the same directory. Next section would explain how a compiled servlet would be deployed in production.

Servlet Deployment

By default, a servlet application is located at the path <Tomcat-installationdirectory>/webapps/ROOT and the class file would reside in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of com.myorg.MyServlet, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in web.xml file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/.

```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
```

<servlet-name>HelloWorld</servlet-name>

<url-pattern>/HelloWorld</url-pattern>

</servlet-mapping>

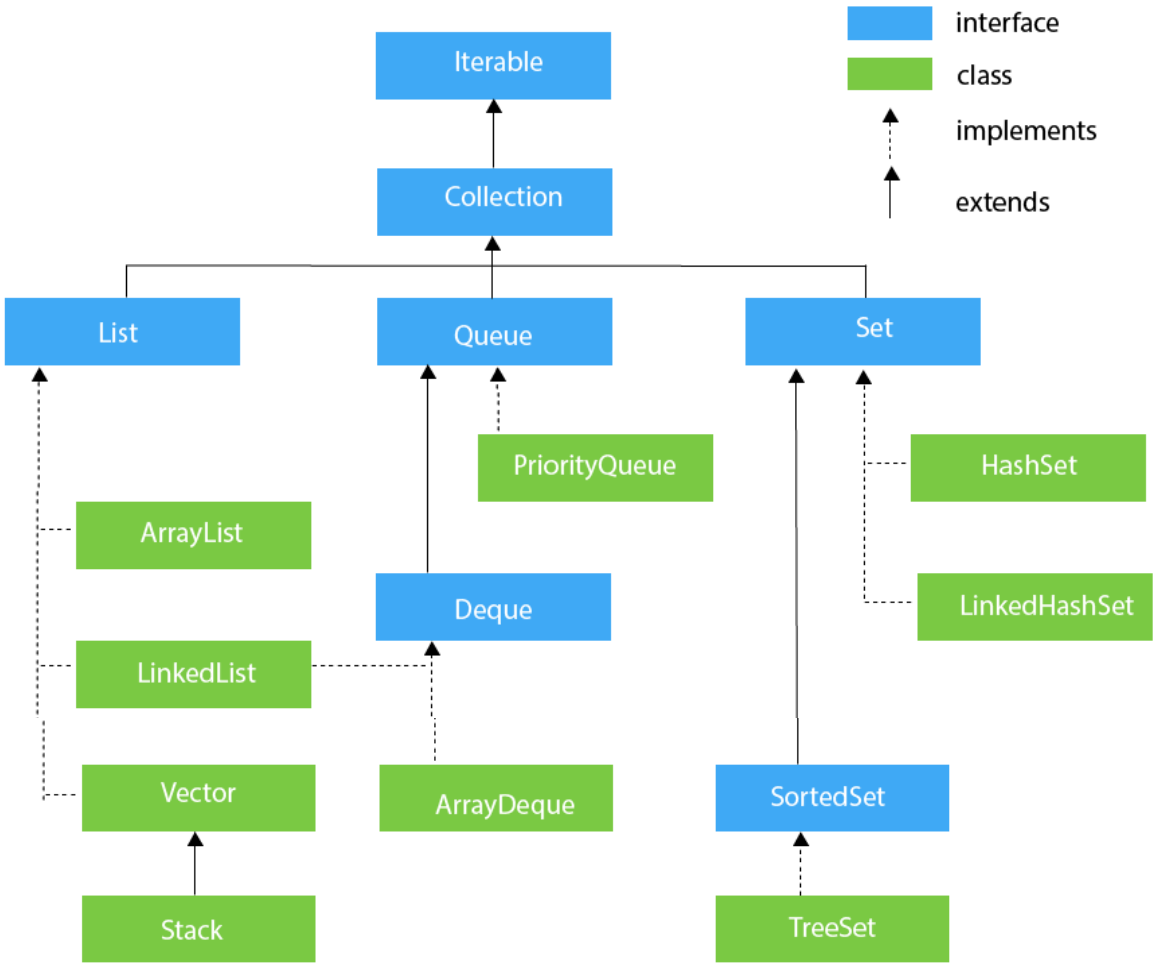
Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installationdirectory>\bin\startup.bat (on Windows) or <Tomcat-installationdirectory>/bin/startup.sh (on Linux/Solaris etc.) and finally type http://localhost:8080/HelloWorld in the browser's address box. If everything goes fine, you would get the following result

Collection

1. The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
2. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
3. Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Hierarchy of Collection Framework



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description

1. public boolean add(E e) - It is used to insert an element in this collection.

2. public boolean addAll(Collection<? extends E> c) - It is used to insert the specified collection elements in the invoking collection.

3. public boolean remove(Object element) - It is used to delete an element from the collection.

4. public boolean removeAll(Collection<?> c) - It is used to delete all the elements of the specified collection from the invoking collection.

5. default boolean removeIf(Predicate<? super E> filter) - It is used to delete all the elements of the collection that satisfy the specified predicate.

6. public boolean retainAll(Collection<?> c) - It is used to delete all the elements of invoking collection except the specified collection.

7. public int size() - It returns the total number of elements in the collection.

8. public void clear() - It removes the total number of elements from the collection.

9. public boolean contains(Object element) - It is used to search an element.

10. public boolean containsAll(Collection<?> c) - It is used to search the specified collection in the collection.

11. public Iterator iterator() - It returns an iterator.

12. public Object[] toArray() - It converts collection into array.

13. public <T> T[] toArray(T[] a) - It converts collection into array. Here, the runtime type of the returned array is that of the specified array.

14. public boolean isEmpty() - It checks if collection is empty.

15. default Stream<E> parallelStream() -It returns a possibly parallel Stream with the collection as its source.

16. default Stream<E> stream() - It returns a sequential Stream with the collection as its source.

17. default Spliterator<E> spliterator() - It generates a Spliterator over the specified elements in the collection.

18. public boolean equals(Object element) - It matches two collections.

19. public int hashCode() - It returns the hash code number of the collection.

Iterator interface

Iterator interface provides the facility of iterating the elements in a forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

No. Method Description

1. public boolean hasNext() - It returns true if the iterator has more elements otherwise it returns false.

2. public Object next() -It returns the element and moves the cursor pointer to the next element.

3. public void remove() -It removes the last elements returned by the iterator. It is less used.

Iterable Interface

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

Iterator<T> iterator() - It returns the iterator over the elements of type T.

Collection Interface

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Some of the methods of Collection interface are Boolean add (Object obj), Boolean addAll (Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

List Interface

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

List <data-type> list1= new ArrayList();

List <data-type> list2 = new LinkedList();

List <data-type> list3 = new Vector();

List <data-type> list4 = new Stack();

There are various methods in List interface that can be used to insert, delete, and access the elements from the list.

The classes that implement the List interface are given below.

Program:105

ArrayList

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed. Consider the following example.

```
import java.util.*;

class TestJavaCollection1{
```

```
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());
}
}
}
```

Output:

Ravi

Vijay

Ravi

Ajay

Program:106

LinkedList

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

Consider the following example.

```
import java.util.*;
public class TestJavaCollection2{
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
```

Program:107

Vector

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

Consider the following example.

```
import java.util.*;

public class TestJavaCollection3{

public static void main(String args[]){

Vector<String> v=new Vector<String>();

v.add("Ayush");

v.add("Amit");

v.add("Ashish");

v.add("Garima");

Iterator<String> itr=v.iterator();

while(itr.hasNext()){

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

}

}

}
```

Output:

```
Ayush

Amit

Ashish
```

Garima

Program:108

Stack

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

Consider the following example.

```
import java.util.*;

public class TestJavaCollection4{

public static void main(String args[]){

Stack<String> stack = new Stack<String>();

stack.push("Ayush");

stack.push("Garvit");

stack.push("Amit");

stack.push("Ashish");

stack.push("Garima");

stack.pop();

Iterator<String> itr=stack.iterator();

while(itr.hasNext()){

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

}

}

}
```

Output:

Ayush

Garvit

Amit

Ashish

Queue Interface

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

Queue interface can be instantiated as:

Queue<String> q1 = new PriorityQueue();

Queue<String> q2 = new ArrayDeque();

There are various classes that implement the Queue interface, some of them are given below.

Program:109

PriorityQueue

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Consider the following example.

```
import java.util.*;

public class TestJavaCollection5{

public static void main(String args[]){

PriorityQueue<String> queue=new PriorityQueue<String>();

queue.add("Amit Sharma");
```

```
queue.add("Vijay Raj");
queue.add("JaiShankar");
queue.add("Raj");
System.out.println("head:"+queue.element());
System.out.println("head:"+queue.peek());
System.out.println("iterating the queue elements:");
Iterator itr=queue.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
queue.remove();
queue.poll();
System.out.println("after removing two elements:");
Iterator<String> itr2=queue.iterator();
while(itr2.hasNext()){
System.out.println(itr2.next());
}
}
}
```

Output:

```
head:Amit Sharma
head:Amit Sharma
iterating the queue elements:
Amit Sharma
```

Raj

JaiShankar

Vijay Raj

after removing two elements:

Raj

Vijay Raj

Deque Interface

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

Deque can be instantiated as:

Deque d = new ArrayDeque();

Program:110

ArrayDeque

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

Consider the following example.

```
import java.util.*;

public class TestJavaCollection6{

public static void main(String[] args) {

//Creating Deque and adding elements

Deque<String> deque = new ArrayDeque<String>();

deque.add("Gautam");
```

```
deque.add("Karan");

deque.add("Ajay");

//Traversing elements

for (String str : deque) {

System.out.println(str);

}

}

}
```

Output:

Gautam

Karan

Ajay

Set Interface

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set can be instantiated as:

```
Set<data-type> s1 = new HashSet<data-type>();

Set<data-type> s2 = new LinkedHashSet<data-type>();

Set<data-type> s3 = new TreeSet<data-type>();
```

Program:111

HashSet

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

Consider the following example.

```
import java.util.*;
public class TestJavaCollection7{
public static void main(String args[]){
//Creating HashSet and adding elements
HashSet<String> set=new HashSet<String>();
set.add("Ravi");
set.add("Vijay");
set.add("Ravi");
set.add("Ajay");
//Traversing elements
Iterator<String> itr=set.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Output:

```
Vijay
Ravi
```

Ajay

Program:112

LinkedHashSet

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

Consider the following example.

```
import java.util.*;
public class TestJavaCollection8{
public static void main(String args[]){
LinkedHashSet<String> set=new LinkedHashSet<String>();
set.add("Ravi");
set.add("Vijay");
set.add("Ravi");
set.add("Ajay");
Iterator<String> itr=set.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Output:

```
Ravi
Vijay
```

Ajay

SortedSet Interface

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order. The SortedSet provides the additional methods that inhibit the natural ordering of the elements.

The SortedSet can be instantiated as:

SortedSet<data-type> set = new TreeSet();

Program:113

TreeSet

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

Consider the following example:

```
import java.util.*;

public class TestJavaCollection9{

public static void main(String args[]){

//Creating and adding elements

TreeSet<String> set=new TreeSet<String>();

set.add("Ravi");

set.add("Vijay");

set.add("Ravi");

set.add("Ajay");

//traversing elements

Iterator<String> itr=set.iterator();
```

```
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Output:

Ajay

Ravi

Vijay

Program: 114

Program to explain the use of Menu and MenuBar in Frame

```
import java.awt.*;
class MenuExample
{
    MenuExample(){
        Frame f= new Frame("Menu and MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("Menu");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Item 1");
        MenuItem i2=new MenuItem("Item 2");
        MenuItem i3=new MenuItem("Item 3");
        MenuItem i4=new MenuItem("Item 4");
        MenuItem i5=new MenuItem("Item 5");
```

```
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}
```