***Introduction to Java***

In software industry, one of the most important characteristic is called change. Now a days, new technologies or programming approaches are added to existing software day by day.

In the early days of computer, there are different programming techniques like modular programming, top-down, bottom-up, procedural and structured programming etc. Among them, structured programming like ‘C’ becomes very popular. But it also has some drawbacks like some functions were not properly worked because of its limited storage capacity.

To overcome these problems another programming approach called ‘Object-oriented programming’ was developed. Bjarne Stroustrup was developed C++ in 1980s.

***Features of OOPs***

The developers can define the following features for OOPs.

***Differences between C and Java***

1. Java does not support datatypes like struct and union.
2. Java does not support keywords like auto, sizeof, register, extern,…
3. Java does not support modifiers like signed and unsigned
4. It does not support header files
5. It does not support #include, #define and #if
6. It does not support pointers
7. It does not support goto statement
8. Java can support object-oriented programming.

***Differences between C++ and Java***

Java does not support

1. pointers
2. header files
3. multiple inheritance
4. operator overloading
5. templates….

**Java Environment:-**

Java provides a large no.of development tools and hundreds of classes and their methods. In java, the development tools are grouped into JDK and hundreds of classes are grouped into JSL or API.

**JDK – Java Development Kit**

1. javac – java compiler, converts source code into byte code
2. java – java interpreter, runs java program
3. javah – java headerfile, used to support native methods in C/C++
4. javadoc – converts source code into HTML code
5. javap – java disassembler, converts byte code into source code
6. jdb – java debugger, detects/finds the errors in a java program
7. appletviewer – to run applets

**JSL (JAVA STANDARD LIBRARY) OR**

**API (APPLICATION POGRAMMING INTERFACE**

In java, hundreds of classes are Stored in JSL. In Java library, the classes are grouped into packages. A package is a collection of classes and interfaces. Some of the important packages are:

1. ***java.lang*** – provide the classes for supporting the basic requirements of java programming like String, System, Thread, Exception etc.
2. ***java.io*** – provide the classes for Input/Output operations in a java program.
3. ***java.util*** – provide the classes for utilities like system date ,time, stacks, linked list, arrayList etc.
4. ***java.net*** – provide the classes required for internet applications.
5. ***java.applet*** – provide the classes for developing applet programs (window based applications (GUI)).
6. ***java.awt*** – provide the classes for developing GUI applications like Button, Label, Checkbox etc.

***Features of Java***

1. Interpreted and Compiled
2. Object-oriented
3. platform Independent
4. Robust
5. Small and Simple
6. Portable
7. Dynamic
8. Distributed
9. Multithreading
10. High performance

Object-oriented

***Platform Independent***

The programs developed in one platform (OS) can be executed in another platform without any modifications.

***Robust***

* Java language is strictly about the datatypes at both compile time and run time.
* Java provides a garbage collector which can automatically removes the unused objects from memory.
* Java provides an Exception handling mechanism which is used to handle the runtime errors called exceptions.

***Small and simple***

To make java language simple and small some of the features are removed from C and C++ languages. The Java language was derived from both C and C++. It takes syntaxes from ‘C’ and object-oriented features from C++.

***Portable***

The Java programs are easily transferred from one computer to another. The java programs are write once and read anywhere (WORA).

***Dynamic***

The new classes or methods can be added to an existing project without any disturbance at runtime.

***Distributed***

Java was mainly developed for internet based applications. So, the java programs are distributed to access from anywhere.

***Multithreading***

A thread is a process/program. In java, we can run (execute) more than one programs simultaneously. It is similar to multitasking in modern operating systems.

***High performance***

The java programs are executed faster than C and C++. For this, it uses a JIT compiler (Just-In-time) internally.

**Java character set**

A character set is a collection of characters that can build words, constants, expressions etc. java can support a 32-bit Unicode character system. It contain 34000+ characters derived from 24 countries. Among them, Java also uses an ASCII character system (256 characters). They are:

Ex.

Alphabets - A to Z and a to z

Digits - 0

Symbols - +, -, \*, /,\,@,#,$,%,^ &,\* ,( ,),[,],{,},<,>,? etc

**Java Tokens**

Small units used in a program are called ‘tokens’.

They are:

1. Keywords
2. Identifiers
3. Literals (constants)
4. Operators
5. Separators

***Keywords***

It means pre-defined words ie the purpose of those words is already defined in Java. They are worked for their intended purpose only. They are also known as ‘reserved words’.

Ex.

char if class abstract strictfp

boolean else private package

byte switch public import

short case protected try

int break interface catch

long default extends throw

float continue implements throws

double while super finally

enum do final synchronize

void for this volatile

Class name – initial capital, remaining are in lower case

Multiple words then each word initial capital

Method name - in lower case, if it has mutlitple words 2nd word onwards initial capital

Package name – lower case

Integer datatypes:

|  |  |  |
| --- | --- | --- |
| Datatype | Size(bytes) | Range |
| byte | 1 | -128 to 127 |
| short | 2 | -32768 to 32767 |
| int | 4 | -2,147,483,648 to 2,147,483,647 |
| long | 8 |  |

Floating point datatypes

|  |  |  |
| --- | --- | --- |
| Datatype | Size(bytes) | Range |
| float | 4 | -3.4e-38 to 3.4e+38 |
| double | 8 | -1.7e-308 to 3.4e+308 |

b) Non-numeric datatypes –

i) character datatypes – represent the character data

- identified by the keyword **char**

- required **2 bytes** of memory

ii) Boolean datatypes - represents a Boolean value either true/false

* Identified by the keyword Boolean
* Required **1 bit**, **0 – false, 1 – true**

Variable declaration:-

Syntax: datatype varname, varname, … ;

Ex. char a;

import java.io.\*;

DataInputStream class – used to accept input from the user.

Syntax: DataInputStream object=new DataInputStream(System.in);

Ex. DataInputStream d = new DataInputStream(System.in);

Methods:

1. read() – reads the ascii value of a character

sym: char = obj.read();

char a = d.read(); - ascii value

char a =(char) d.read();

1. readLine() – reads multiple characters as input

* returns a string value

syn: string = obj.readLine();

String s=d.readLine();

To read an integer

int a = Interger.parseInt(d.readLine());

to read a float value

float b = Float.parseFloat(d.readLine());

to read a double value

double c=Double.parseDouble(d.readLine());

1. print() – to print/display

syn: System.out.print(message/variable);

ex. System.out.print(“welcome”);

int a=100;

System.out.print(a);

1. println() – can print in a single line

syn: System.out.println(message/variable);

ex. System.out.println(“welcome to Java”);

int x=100;

System.out.println(x); o/p - 100

+(concatenation operator):- it concatenate/add a string and a variable

System.out.println(“X = “+x); o/p - X=100

structure of java program

1. documentation section – comment lines
2. single line comment - //
3. multi line comment - /\* ---

---- \*/

1. java documentation - /\*\* ---

---- \*/

1. package statement – collection of related classes

used to create user defined packages

syn: package packagename;

public class classname

{

Variables(attributes)

Methods(opearations)

}

1. import statement

To work with either a class in a predefined or user defined package then it must be linked to the program. For this import statement is used. It is similar to #include.

syn:- import packagename[.subpackagename].\*;

ex. import java.io.\*;

import java.util.Scanner;

1. class definition

syn class <classname>

{

Variables

Methods

}

1. interface definition –

Java does not support multiple inheritance. To ensure that interfaces were used. An interface is also one kind of class. It is a collection of final variables and abstract methods.

syn: interface <interfacename>

{

final variables;

abstract methods;

}

1. Every C/C++ program start with the function main. Similarly, the java developers also wants to start a java program with main. So, they define main method, where a method is a member of a class.

Syn: class mainmethodclassname

{

public static void main(String args[])

{

Statements;

}

}

Ex. class Sample

{

public static void main(String args[])

{

System.out.println(“welcome to Java”);

}

}

***To save a java program***:

Programs are saved with classname.java (main method class name)

Ex. Sample.java

***To compile a java program:***

Syn: javac filename.java

Ex. javac Sample.java

It generates the Byte code file – ***Sample.class***

***To Run a java program***

Syn: java filename

Ex. java Sample

***To find total and average of 3 no.s***

import java.io.\*;

class prg2

{

public static void main(String[] args) throws Exception

{

DataInputStream d=new DataInputStream(System.in);

System.out.println("Enter 3 no.s : ");

int a,b,c;

a=Integer.parseInt(d.readLine());

b=Integer.parseInt(d.readLine());

c=Integer.parseInt(d.readLine());

int tot=a+b+c;

double avg=tot/3.0;

System.out.println("Total : "+tot);

System.out.println("Average : "+avg);

}

}

**Control statements**

In general, the statements in a program are executed one after another sequentially. To control the execution of a program control statements are used. The control statements are classified into various types as follows:

1. If statement:-

It can execute a statement/set of statements based on a condition. It is applied in 4 ways.

1. Simple if b) if..else c) else if .. ladder d) nested if

***To find the biggest of two no.s***

import java.io.\*;

class sif

{

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

{

DataInputStream d=new DataInputStream(System.in);

System.out.println("Enter 2 no.s: ");

int a=Integer.parseInt(d.readLine());

int b=Integer.parseInt(d.readLine());

int max=a;

if(max<b)

max=b;

System.out.println("Big = "+max);

}

}

else if .. ladder:

Syn:

if(condition-1)

{

statements;

}

else if(condition-2)

{

statements;

}

----

---

else

{

statements;

}

Nested if:

Syn:

if(condition-1)

{

if(condition-2)

{

statements;

}

else

{

statements;

}

}

else

{

if(cond-3)

{

}

else

{

}

}

1. Switch statement:

Syntax:

switch(expression)

{

case value1:

statements;

break;

case value2:

statements;

break;

-----

case valuen:

statements;

break;

default:

statements;

break;

}

import java.io.\*;

class switchex1

{

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

{

DataInputStream d = new DataInputStream(System.in);

char a;

System.out.println("Enter an alphabet: ");

a=(char)d.read();

switch(a)

{

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

System.out.println("-- is lower case vowel");

break;

case 'A': case 'E': case 'I':

case 'O': case 'U':

System.out.println("-- is an upper case vowel");

break;

default:

System.out.println("--- is a consonant");

break;

}

}

}

Conditional operator statement:

The combination of the symbols ? and : is called condtional operator. The control statement applied with the conditional operator is called conditional operator statement. It is used to evaluate a condition like if statement.

Syn: **variable = condition ? expression-1 : expression-2;**

**Ex. big = a>b ?a :b;**

Looping statements:

Repeated execution of statements is called a loop.

The control statements used to terminate or stop a loop are called ‘looping statements’.

They are:

1. While
2. Do..whil
3. For

While :

Entry-controlled or pre-test loop

Syntax:

While(condition)

{

Statements;

Incr/decr;

}

Do..while: exit-controlled or post-test loop

Syntax:

do

{

Statements;

Incr/decr;

}while(condition);

***for loop***

Syn: for(initialization; condition ; incr/decr)

{

---

Statements;

}

import java.io.\*;

class for1

{

public static void main(String[] args) throws Exception

{

DataInputStream dis=new DataInputStream(System.in);

int n,i;

System.out.println("Enter n : ");

n=Integer.parseInt(dis.readLine());

System.out.print("\nForward : ");

for(i=1;i<=n;i++)

System.out.print(i+" ");

System.out.print("\nBackward : ");

for(i=n;i>0;i--)

System.out.print(i+" ");

}

}

**Scanner class:-**

It is used to access user input ie. To take input from the user. It will be provided in ***java.util*** package.

syn: Scanner objname = new Scanner(System.in);

Ex. Scanner sc=new Scanner(System.in);

**Methods:**

1. next() – to read string data

Ex. String name = sc.next();

1. nextInt() – to read an integer value

Ex. int n= sc.nextInt();

1. nextLong() – to read a long datatype value

Ex. long l=sc.nextLong();

1. nextFloat() – to read a float value

Ex. float s=sc.nextFloat();

1. nextDouble() – to read a double type value

Ex. double sal = sc.nextDouble();

1. nextChar() – used to read a single character

eg. char a=sc.nextChar();

**ARRAYS**

It is a homogeneous collection of elements ie collection of similar elements (data). It can store set of similar elements in sequential memory locations underlying a single variable name. Each array element is identified by an index value starts from ‘0’.

**Types of Arrays**

The arrays are classified into 2 types. They are

1. Single dimensional arrays
2. Multi dimensional arrays

***Single Dimensional arrays***

These arrays can store set of similar elements in only one dimension. A dimension is nothing but either a row/ a col.

1. **Declaration:** to allocate memory

syn1: datatype arranyname[];

arrayname=new datatype[size];

ex. int a[]; or int[] a;

a=new int[10];

syn2: datatype arranyname[]=new datatype[size];

ex. int a[] = new int[10];

int[] a=new int[10];

1. **input/output:**

These arrays required a loop statement for representing the index values.

Input:

for(i=0;i<10;i++)

a[i]=sc.nextInt();

output:

for(i=0;i<10;i++0

System.out.println(a[i]);

1. **initialization:** to store fixed values into an array

Ex. int a[] = {1,2,3,4,5,6,7};

To find the length of the array use ***arrayname.length ie a.length***

import java.util.\*;

class Arr2

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int a[],n,i;

System.out.println("Enter no.of elements :");

n=sc.nextInt();

a=new int[n];

System.out.println("Enter "+n+" elements : ");

for(i=0;i<n;i++)

a[i]=sc.nextInt();

System.out.print("Array elements : ");

for(i=0;i<a.length;i++)

System.out.print(a[i]+" ");

int s=0;

for(i=0;i<n;i++)

s=s+a[i];

System.out.println("\nSum : "+s);

int big=a[0];

for(i=0;i<n;i++)

{

if(a[i]>big)

big=a[i];

}

System.out.println("Big = "+big);

}

}

***Multi Demensional Arrays***

These arrays can store set of similar elements in more than one dimensions ie in no.of rows and cols. It is nothing but a matrix.

1. **declaration**: to allocate memory

**syn1**: datatype arranyname[][];

arrayname=new datatype[R][C];

where R – represents no.of rows

C – represents no.of cols

ex. int a[][];

a=new int[3][4];

1st row – a[0][0], a[0][1], a[0][2], a[0][3]

2nd row – a[1][0], a[1][1], …

3rd row – a[2][0], a[2][1], …

**syn2:** datatype arranyname[][]=new datatype[R][C];

ex. int a[][] = new int[3][4];

1. **input/output**

These arrays required 2 loop statements – one represents the row index and the other one represents the col index.

input:

eg. for(i=0;i<3;i++) - row

for(j=0;j<4;j++) - col

a[i][j]=sc.nextInt();

output:

eg. for(i=0;i<3;i++)

{

for(j=0; j<4;j++)

s.o.print(a[i][j]+” “);

s.o.println(“”);

}

1. **Initialization:**

To store fixed values in a matrix.

eg.

int a[][]={{1,2,3,4 },{2,3,4,5 },{3,4,5,6}};

import java.util.\*;

class Marr2

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int a[][],i,j,r,c;

System.out.println("Enter no.of rows and cols of Matrix-A : ");

r=sc.nextInt();

c=sc.nextInt();

a=new int[r][c];

System.out.println("Enter "+(r\*c)+" elements : ");

for(i=0;i<r;i++)

for(j=0;j<c;j++)

a[i][j]=sc.nextInt();

System.out.println("Matrix-A...");

for(i=0;i<r;i++)

{

for(j=0;j<c;j++)

System.out.print(a[i][j]+" ");

System.out.println("");

}

System.out.println("Transpose of Matrix-A...");

for(i=0;i<c;i++)

{

for(j=0;j<r;j++)

System.out.print(a[j][i]+" ");

System.out.println("");

}

}

}

**OOPS**

Struct emp

{

Int eno;

Char ena[20]; data

Float sal;

}e;

Void input()

{

}

Void output() code

{

}

The programming which was developed using classes and objects is called oops. The developers of oops can define some of the features for oops.

**Features of OOPs**

1. Object
2. Class
3. Data Encapsulation
4. Data Abstraction
5. Inheritance
6. Polymorphism
7. Dynamic Binding
8. Message Passing
9. ***Object*** – is a real world entity. Each entity can have some of the properties(attributes) and behavior(operations).

***Attributes (data)***

***Operations(code)/ methods***

ex.

Attribute1

Atribute2

-------

Operation1

Operation2

-----

1. ***class*** – is a blue print or model of an object.

Syn: class classname

{

Attributes (data)

Operations (code)

}

1. **Data Encapsulation** – means to combine both the data and code together. A class can exhibit this encapsulation.
2. ***Data Abstractio***n – means to access the data of a class without knowing the implementation details ie. It hides the unnecessary data from users. This is called ‘data hiding’.

A class can construct a new datatype called Abstract Datatype (ADT).

1. ***Inheritance*** – is a mechanism used to create new classes from existing classes. The main advantage is the re-usability of code ie. The code written in one class can be accessed from another class.

Existing class - Base or Super class (parent)

New class - Derived or Sub class (child)

1. ***Polymorphism*** – poly means many and morphism means forms ie one function is applied in more than one forms for different purposes.

Polymorphism

Compile time Runtime

Overloading overriding

Static/Early Dynamic/Late

Binding Binding

Ex. Function overloading ex. Virtual Functions

Operator overloading

Constructor overloading

1. **Dynamic Bindin**g – binding means the method to be executed to a method call.

Static binding – if the compiler identifies the method to be executed to a method call at **compile time** is called **static binding**.

Dynamic binding - if the compiler identifies the method to be executed to a method call at **run time** is called **dynamic binding**.

1. **Message passing** – It is useful to send information from one object to another ie establishes the communication between the objects. For this arguments are used.

Class:

A class is a collection of variables (data) and methods (functions).

Syntax:

class classname

{

datatype var1;

datatype var2;

----

returntype methodname()

{

---

statements;

}

----

}

Ex. class Emp

{

int eno;

string name;

double sal;

void input()

{

Statements;

}

void output()

{

Statements;

}

}

Object:- It is an instance of a class. For a class the memory will be allocated through that class object only because a class can create new datatype with the classname.

Syntax: classname objectname = new classname();

Ex Emp e=new Emp();

.(dot) operator:- It is used to access both the variables and methods of aclass.

To acess a variable:

Syntax: object.variable

Ex. e.eno

e.ena

e.sal

To access a method:

Syntax: object.methodname();

Ex. e.input();

e.output();

**Visibility or access control:**

They are used to control the accessing of data of a class in a program. They are in 3 types.

1. private - It is used to declare both the variables and methods of a class. They are worked like as local variables. The private members are accessed inside of that class only. They are not accessed in outside of a class or another class.

ex . private int a;

private void input()

{

}

1. public - It is used to declare both the variables and methods of a class. They are worked like as global variables. The public members are accessed inside of a class or outside of a class or another class.

ex. public int b;

public void input()

{

}

1. protected – It is also similar to private. But the protected members are accessed by the derived classes (sub classes).

ex. protected int x;

protected void input()

{

}

import java.util.\*;

class Sample

{

Scanner sc=new Scanner(System.in);

private int a;

void input()

{

System.out.println("enter a no: ");

a=sc.nextInt();

}

void output()

{

System.out.println("A =" +a);

}

}

class prg2

{

public static void main(String[] args)

{

Sample s=new Sample();

s.input();

s.output();

}

}

***Passing arguments to a method:***

In java, a class method is also defined with arguments. With this, we can send values class variables from outside.

Syn: returntype methodname(argslist)

{

----

Statements;

}

import java.util.\*;

class Sample

{

int n;

void input(int k)

{

n=k;

}

void disp(String msg)

{

System.out.println(msg+n);

}

}

class marg1

{ public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

Sample s1 = new Sample();

System.out.println("Enter a no: ");

int a=sc.nextInt();

s1.input(a);

s1.disp("S1 : ");

Sample s2=new Sample();

s2.input(123);

s2.disp("S2 : ");

}

}

***Method returns a value:***

In java, a class method is defined with return value ie a class method returns a value.

Syn: returntype methodname(argslist)

{

----

statements;

return statement;

}

**Constructors:-**

* Constructors are used to construct/create objects in memory
* They are mainly used to initialize the variables of a class
* A constructor is also a method of a class
* Constructor name is the classname
* It should not have any return type
* It is defined with/without arguments
* It is an automatically executed method of a class ie. Without calling
* Constructors are executed/invoked when a class object is instantiated (created)
* Constructor will be executed once for each object.

Syntax:

classname([argslist])

{

statemensts;

}

Types:

1. Default Constructor
2. Non-Parameterized Constructor
3. Parameterized constructor

***Default constructor***

If any constructor is not defined in a class then the default constructor can automatically invoked internally for constructing an object.

class sample

{

int n;

void read()

{

n=100;

}

void disp(String msg)

{

System.out.println(msg+n);

}

}

class defcons

{

public static void main(String[] args)

{

sample s=new sample();

s.read();

s.disp("S : ");

}

}

Syntax: classname()

{

Statements;

}

***Parameterized Constructor:***

If a constructor is defined with arguments (parameters) is called parameterized constructor.

Syntax: classname( argslist )

{

statements;

}

For parameterized constructor the values are to be send through the object declaration statement because constructors are invoked when that class objects are instantiated.

Syntax: classname objname = new classname(args)

Constructor overloading

Overloading means the same method is define multiple with different prototypes.

If more than one constructor is defined within a class then is called constructor overloading.

import java.util.\*;

class sample

{

Scanner sc=new Scanner(System.in);

int n;

sample()

{

n=0;

}

sample(int k)

{

n=k;

}

void read()

{

System.out.println("Enter n : ");

n=sc.nextInt();

}

void disp(String msg)

{

System.out.println(msg+n);

}

}

class consoverload

{

public static void main(String[] args)

{

sample s1=new sample();

sample s2=new sample(200);

s1.disp("S1 : ");

s2.disp("S2 : ");

s1.read();

s1.disp("Now, S1 : ");

}

}

**Passing objects as arguments to a method:**

In general, a class methods are defined with arguments like variables, arrays, etc. like that, objects are also passed as arguments to a method. They are defined as follows:

Syntax:

returntype methodname(classname obj)

{

statements;

}

When a class object is passed as argument then it passes along with all its members ie both variables and methods.

Ex.

import java.util.\*;

class sample

{

Scanner sc=new Scanner(System.in);

int n;

sample()

{

n=0;

}

sample(int k)

{

n=k;

}

void read()

{

System.out.println("Enter n : ");

n=sc.nextInt();

}

void disp(String msg)

{

System.out.println(msg+n);

}

void add(sample x,sample y)

{

n=x.n+y.n;

}

}

class passobj

{

public static void main(String[] args)

{

sample s1=new sample(10);

sample s2=new sample(40);

sample s3=new sample();

s3.add(s1,s2);

s1.disp("s1 : ");

s2.disp("s2 : ");

s3.disp("s1+s2 : ");

}

}

**Method returns an object**

Like that a class methods are also returns an object. When a method returns an object then it returns along with all its members. They are defined as follows:

Syntax: **classname** methodname(classname obj)

{

statements;

**return obj;**

}

import java.util.\*;

class distance

{

Scanner sc=new Scanner(System.in);

int m,cm;

distance()

{

m=cm=0;

}

distance(int a,int b)

{

m=a;

cm=b;

}

void read()

{

System.out.println("Enter no.of meters and centimetiers ");

m=sc.nextInt();

cm=sc.nextInt();

}

void disp(String msg)

{

System.out.println(msg+m+" meters and "+cm+" centimeters");

}

distance add(distance x)

{

distance d=new distance();

d.m=m+x.m;

d.cm=cm+x.cm;

if(d.cm>=100)

{

d.m+=1;

d.cm-=100;

}

return d;

}

}

class retobj1

{

public static void main(String[] args)

{

distance d1=new distance();

distance d2=new distance();

System.out.println("Enter distance 1 : ");

d1.read();

System.out.println("Enter distance 2 : ");

d2.read();

distance d3=new distance();

d3=d1.add(d2);

d1.disp("D1 : ");

d2.disp("D2 : ");

d3.disp("D1+D2 : ");

}

}

***this keyword***

in a class, if both variables and arguments are defined with the same name then by default the compiler assigns default value ‘0’. In this case, the compiler get confused. To overcome this problem, **this** keyword is used. It represents the current object. So, it will be applied with variables of a class.

Syntax: this.variable = arg

import java.util.\*;

class sample

{

Scanner sc=new Scanner(System.in);

int n;

sample()

{

n=0;

}

sample(int n)

{

this.n=n;

}

void disp(String msg)

{

System.out.println(msg+n);

}

}

class thisex

{

public static void main(String[] args)

{

sample s1=new sample(10);

sample s2=new sample(50);

s1.disp("s1 : ");

s2.disp("s2 : ");

}

}

**STATIC MEMBERS**

In general, the variables and methods defined within a class are loaded into the memory along with the class instances called objects. So, they are also known as instance variables and instance methods. But sometimes, we need to access the same memory by the no.of class objects. For this, java provides the feature called static members. The static members are of 2 types.

1. Static variables
2. Static methods
3. ***Static variables:***

In general the variables defined within a class are loaded into the memory along with the instance called object. So, they can be called as ‘instance variables’.

But the static variables are loaded into the memory only once along with the class definition. The same memory can be used by no.of that class objects.

The static variables becomes class variables, but not for objects.

Syntax: static datatype variable = value;

Ex. static int s=0;

class sample

{

static int c=0;

sample()

{

c++;

}

void count()

{

System.out.println("No.of objects created : "+c);

}

}

class svar

{ public static void main(String[] args)

{ sample s1=new sample();

sample s2=new sample();

s1.count();

sample s3=new sample();

s3.count();

}

}

1. ***Static method***

In general, the methods defined within a class are loaded into the memory along with the instance of a class called object. So, it can be called as instance method. But the static methods are loaded into the memory only once along with the class definition.

The **static methods can access the static variables** only. But the instance methods can access both the instance and static variables. The static methods are preceded by the keyword ‘static’ followed by method definition.

Syn: static returntype methodname()

{

Statements;

}

The static methods are invoked by the classname and object, but classname is preferable.

Syntax: classname.methodname();

Ex.

class sample

{

int n=100;

static int s=1000;

static void disp()

{

//System.out.println(n);

System.out.println(s);

}

}

class smeth

{

public static void main(String[] args)

{

sample s=new sample();

sample.disp();

}

}

**STRINGS**

A string is a collection of characters ie a character array. Every string is terminated by the null character (\0) in C/C++. But in java, a string can be represented as a class because java is a oops language.

Syntax: char stringname[size];

Ex. char str[20];

***Initialization*** – It is used to store fixed value (constant).

Ex. char str[20]=”welcome”;

In java, the strings are represented by the String classes. They are

1. String class – immutable string
2. StringBuffer class – mutable string

**String class**

It is used to represent the fixed length string in java ie the size of the string cannot be changed. It is also known as immutable string.

***Declaration***: to allocate memory.

Syn 1: String obj = “value”;

Ex. String s=”welcome”;

Syn 2: String obj;

Obj=”value”;

Ex. String s;

s=”welcome”;

***Constructors:***

Syn:1: String obj=new String();

Ex. String str = new String();

Syn:2: String obj = new String(value);

Ex. String str=new String(“welcome”);

***Methods:***

1. **int length()** – It returns an integer representing the length of a string. Length means total no.of characters including spaces.

**Syntax**: **intvar=obj.length()**

class slen

{

public static void main(String[] args)

{

String s="welcome";

int l=s.length();

System.out.println("String : "+s);

System.out.println("Length : "+l);

}

}

1. **String toLowerCase() –** It converts a given string to lower case

**Ex.** String s=”WELCOME TO STRINGS”

System.out.println(s.toLowerCase()); o/p - welcome to strings

1. **String toUpperCase() -**  It converts a given string to upper case

**Ex.** String s=”welcome”;

System.out.println(s.toUpperCase()); o/p- WELCOME

1. **boolean equals(String) –** It returns true if two strings are equal (case sensitive), otherwise false.

Ex. String s1="welcome";

String s2="Welcome";

if(s1.equals(s2))

System.out.println("Equal");

else

System.out.println("Not Equal"); o/p- Not Equal

1. **boolean equalsIgnoreCase(String) -** It returns true if two string values are same irrespective of case.

Ex. String s1="welcome";

String s2="Welcome";

if(s1.equalsIgnoreCase(s2))

System.out.println("Equal");

else

System.out.println("Not Equal"); o/p- Equal

1. **int compareTo(String)** – It can compare one by one character in both strings based on their ASCII values. It returns an integer value either ‘0’ or non-zero(+1/-1)

syntax: int=string1.compareTo(string2)

Return description

1. Equal
2. string1>string2

-1 string1<string2

Ex. if(s1.compareTo(s2)==0)

System.out.println("Equal");

else if(s1.compareTo(s2)>0)

System.out.println("string1 is big");

else

System.out.println("String2 is big");

1. **String concat(String) –** It is used to concatenate or add one string to another ie similar to ‘+’.

class concat

{

public static void main(String[] args)

{

String s1="welcome";

String s2=" to Java Strings";

System.out.println("String1 : "+s1);

System.out.println("String2 : "+s2);

String s3=s1.concat(s2);

System.out.println("After concatenating :");

System.out.println(s3);

}

}

1. **String substring(startindex[,endindex]) –** It returns the portion of text in a string. It returns the sub string from start to end-1 index. If you omit end index then it returns from start to end of the string.

**Ex.**

class substr

{

public static void main(String[] args)

{

String s="welcome to Java Strings";

System.out.println("String is : "+s);

String s1=s.substring(3);

System.out.println("sub String : "+s1);

String s2=s.substring(3,7);

System.out.println("now, substring :"+s2);

String s3=s.substring(3,15);

System.out.println("now, substring :"+s3);

}

}

1. **String replace(oldchar,newchar) – It** replaces oldchar with newchar
2. **String replaceAll(str,replacementstring) – I**t replaces all str with replacement string
3. **String replaceFirst(str,replacementstring)**  - It replace first str with replacement string

class repstr

{

public static void main(String[] args)

{

String s="welcome to bdps, welcome to Strings";

System.out.println("String is : "+s);

String s1=s.replace('o','x');

System.out.println(s1);

String s2=s.replaceFirst("come","Done");

System.out.println(s2);

String s3=s.replaceAll("come","Done");

System.out.println(s3);

}

}

1. **int indexOf(char[, intpos]) –** Itreturns the index value of first occurrence of a given char.

where intpos –represents the position from which it will checked

1. **int lastIndexOf(char) -**  It returns the last index value of a given char.
2. **char charAt(index) -**  It returns the char at a given index.

class indstr

{

public static void main(String[] args)

{

String s="welcome weldone";

System.out.println("String is : "+s);

System.out.println("Index of e : "+s.indexOf('e'));

System.out.println("Last Index of e : "+s.lastIndexOf('e'));

System.out.println("Index of e after 8 : "+s.indexOf('e',8));

}

}

1. **boolean endsWith(string) –** It returns true, if the string ends with a given string
2. **boolean startsWith(string) –** It returns true, if the string starts with a given string.

class strwith

{

public static void main(String[] args)

{

String s="welcome";

System.out.println("String is : "+s);

System.out.print("Start with wel : ");

if(s.startsWith("wel"))

System.out.println("Yes");

else

System.out.println("No");

System.out.print("Ends with come : ");

if(s.endsWith("come"))

System.out.println("Yes");

else

System.out.println("No");

}

}

1. **boolean contentEquals(StringBuffer) –** It checks the value in both String object and StringBuffer object

**ex.** String s1=”welcome”;

StringBuffer s2=”welcome”;

s1.contentEquals(s2) - true

1. **byte[] getBytes(start,end) –** It returns the byte values of a String object into an array.
2. **Void getChars(start,end,char[],int) –** It returns the characters from start to end into char[]

class getbytes

{

public static void main(String[] args) throws Exception

{

String s="welcome";

System.out.println("String is : "+s);

/\*byte b[]=new byte[20];

b=s.getBytes(2,4,b);

System.out.println(b);\*/

char a[]=new char[10];

s.getChars(0,4,a,0);

System.out.println(a);

}

}

**StringBuffer Class**

It represents a variable length string ie the size of the string will be changed ( grow/ shrink). It is also called as ‘Mutable String’. With this, we can insert or add new string to an existing string or remove portion of string from a string.

**Declaration:**

Syn:1: StringBuffer obj = “value”;

Ex. StringBuffer sb=”welcome to bdps”;

Syn:2” StringBuffer obj;

Obj=”value”;

Ex. StringBuffer sb;

Sb=”welcome to java”;

**Constructors**

1. StringBuffer obj=new SstringBuffer();

Ex. StringBuffer sb = new StringBuffer();

Sb=”welcome to string buffer class”;

1. StringBuffer obj=new StringBuffer(value);

Ex. StringBuffer sb=new StringBuffer(“welcome to java”);

**Methods:**

1. **int length()** – Returns the length of a string buffer object
2. **int capacity() –**returns the capacity of a string buffer object ie the default memory allocated.
3. **StringBuffer append(string) -** used to append string buffer object with a string ie Adds a string at end.
4. **void setCharAt(index,char) –** used to set a char at a given index
5. ***StringBuffer delete(int start,int end) :***

Removes the characters from start to end-1 indexes in the String Buffer object.

1. ***StringBuffer deleteCharAt(int index) :***

Removes the character at specified index.

1. ***StringBuffer insert(int offset,datatype var) :***

Insert the string representation of any type of argument into this String Buffer ie char, boolean, int, float, long, double, string, char[], object etc

1. ***StringBuffer reverse() : U****sed to reverse the string Buffer*

**INHERITANCE**

It is one of the most useful feature in oops. It is a mechanism which is used to create new classes from existing classes. It is mainly used for the reusability of code ie the code written in one class can be accessed from other class.

In this, an existing class is known as ‘super’ or ‘Base’ class and a new class is known as ‘sub’ or ‘derived’ class. In inheritance, a sub class has ability to access the properties (variables) and behavior (methods) of a super class.

Here, a derived class can be extended with the features of a base class. So, java uses the keyword extends while implementing a derived class.

Syntax:

class superclassname

{

Variables;

Methods;

}

class subclassname extends superclassname

{

Variables

methods

}

**Types of Inheritances**:

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Single Inheritance

If a sub class (derived) is inherited/derived/extended from only one super class then is called ‘single inheritance’.

Syn: class A

{

---

}

class B extends A

{

----

----

}

import java.util.\*;

class marks

{

Scanner sc=new Scanner(System.in);

protected int a,b,c;

void geta()

{

System.out.println("Enter 3 no.s : ");

a=sc.nextInt();

b=sc.nextInt();

c=sc.nextInt();

}

}

class result extends marks

{

private int tot;

private double avg;

void find()

{

tot=a+b+c;

avg=tot/3.0;

}

void disp()

{

System.out.println("A= "+a+"\nB ="+b+"\nC= "+c);

System.out.println("Total : "+tot+"\nAverage : "+avg);

}

}

class sin2

{

public static void main(String[] args)

{

result r=new result();

r.geta();

r.find();

r.disp();

}

}

1. **Multiple inheritance**

If a subclass is derived from more than one super classes is called multiple inheritance. Java does not multiple inheritance. To achieve this, interfaces were used.

A B

C

1. **Multi level inheritance**

If a sub class is derived from another sub class which has been already derived from a super class, is called multi level inheritance.

A

B

C

1. Hierarchical Inheritance

If number of sub classes are defined for a super class then is called hierarchical inheritance. It will look like as an inverted tree structure. In this, each sub class may also contain another sub classes.

A

B C

class A

{

protected int a;

void geta()

{

a=10;

}

}

class B extends A

{

protected int b;

void getb()

{

b=20;

}

void putb()

{

System.out.println("A = "+a+" B ="+b);

}

}

class C extends A

{

protected int c;

void getc()

{

c=30;

}

void putc()

{

System.out.println("A = "+a+" C ="+c);

}

}

class hier

{

public static void main(String[] args)

{

B b1=new B();

b1.geta();

b1.getb();

System.out.println("From class B :");

b1.putb();

C c1=new C();

c1.geta();

c1.getc();

System.out.println("From class C :");

c1.putc();

}

}

***Constructors in Inheritance:***

The constructors are used to construct objects in memory. The constructors are invoked when a class object is instantiated ie created.

The constructors are also applied in inheritance. If the constructors are defined in both super and sub classes then the super class constructors are invoked first, later the sub class constructors are invoked.

class A

{

A()

{

System.out.println("Constructor in class A");

}

}

class B extends A

{

B()

{

System.out.println("Constructor in class B");

}

}

class inhcons

{

public static void main(String[] args)

{

B b=new B();

}

}

Constructors in Hierarchical

class A

{

A()

{

System.out.println("Constructor in class A");

}

}

class B extends A

{

B()

{

System.out.println("Constructor in class B");

}

}

class C extends A

{

C()

{

System.out.println("Constructor in Class C");

}

}

class inhcons2

{

public static void main(String[] args)

{

System.out.println("---From Class B---");

B b1=new B();

System.out.println("---From Class C---");

C b2=new C();

}

}

Constructors in Multi Level Inheritance

class A

{

A()

{

System.out.println("Constructor in class A");

}

}

class B extends A

{

B()

{

System.out.println("Constructor in class B");

}

}

class C extends B

{

C()

{

System.out.println("Constructor in Class C");

}

}

class inhcons1

{

public static void main(String[] args)

{

C b=new C();

}

}

Note:

If the parameterized constructors are defined in both super and sub classes then it is not possible to send values to the super class parameterized constructor from the sub class object. For this, **super** keyword is used.

***Method Overloading***:

If the no. of methods in a class are defined with the same then is called ‘method overloading’. But the methods are used for different purposes.

In method overloading, the no. of methods are defined with the same, but with different no.of arguments or different types of arguments.

Ex.

class sample

{

void show()

{

System.out.println("--welcome to method overloading---");

}

void show(char a)

{

System.out.println("Character : "+a);

}

void show(int n)

{

System.out.println("Number : "+n);

}

void show(char a,int b)

{

System.out.println("Character : "+a+"\nNumber : "+b);

}

}

class moverload

{

public static void main(String[] args)

{

sample s=new sample();

System.out.println("From show()---");

s.show();

System.out.println("From show(char)---");

s.show('p');

System.out.println("From show(int)---");

s.show(999);

System.out.println("From show(char,int)---");

s.show('B',444);

}

}

***Method Overloading***:

If the no. of methods in a class are defined with the same then is called ‘method overloading’. But the methods are used for different purposes.

In method overloading, the no. of methods are defined with the same, but with different no.of arguments or different types of arguments.

Ex.

class sample

{

void show()

{

System.out.println("--welcome to method overloading---");

}

void show(char a)

{

System.out.println("Character : "+a);

}

void show(int n)

{

System.out.println("Number : "+n);

}

void show(char a,int b)

{

System.out.println("Character : "+a+"\nNumber : "+b);

}

}

class moverload

{

public static void main(String[] args)

{

sample s=new sample();

System.out.println("From show()---");

s.show();

System.out.println("From show(char)---");

s.show('p');

System.out.println("From show(int)---");

s.show(999);

System.out.println("From show(char,int)---");

s.show('B',444);

}

}

class A

{

void show()

{

System.out.println("Show in Class A");

}

void disp1()

{

System.out.println("disp1 in Class A");

}

}

class B extends A

{

void show()

{

System.out.println("Show in Class B");

}

void disp2()

{

System.out.println("disp2 in Class B");

}

}

class methoveride

{

public static void main(String[] args)

{

B b=new B();

b.disp1();

b.disp2();

//in overriding derived class method only be invoked

b.show();

}

}

To invoke the super class method in method overriding ***super*** keyword is used.

***super keyword***

in Java, super keyword is used for representing the super class in inheritance. For this keyword the following purposes are defined.

They are:

1. super.variable
2. super.method();
3. super(argslist)
4. ***super.variable***

in inheritance, if both super and sub classes are defined the same variable then sub class method can recognize its member only. To access the super class variable from sub class method super keyword is used.

Syntax: super.variable

class A

{

protected int a;

void geta()

{

a=10;

}

}

class B extends A

{

int a;

void getb()

{

a=100;

}

void disp()

{

System.out.println("super(a) = "+a+" a= "+a);

System.out.println("---using super ...");

System.out.println("super(a) = "+super.a+" a= "+a);

}

}

class supvar

{

public static void main(String[] args)

{

B b=new B();

b.geta();

b.getb();

b.disp();

}

}

1. ***super.method***

In method overriding, the sub class methods can hide the super class methods. To invoke the super class methods super keyword is used. It is useful to invoke a super class method from a sub class.

Syntax: **super.method([argslist]);**

class A

{

void show()

{

System.out.println("Show in Class A");

}

}

class B extends A

{

void show()

{

super.show();

System.out.println("Show in Class B");

}

}

class supmeth

{

public static void main(String[] args)

{

B b=new B();

b.show();

}

}

1. ***super(arguments)***

In inheritance, if both super and sub classes are defined with parameterized constructors then to send values to super class parameterized constructor from the sub class constructor super(args) is used. Because we have create objects for sub classes only.

Syn: ***super(args);***

class base

{

int a,b;

base(int x,int y)

{

a=x;

b=y;

}

}

class derived extends base

{

int c;

derived(int i,int j,int k)

{

super(i,j);

c=k;

}

void show()

{

System.out.println("a = "+a+" b= "+b+" c= "+c);

}

}

class supcons

{

public static void main(String[] args)

{

derived d=new derived(10,20,30);

d.show();

}

}

***final keyword***

java provides one of the keyword called final. For this keyword the following 3 properties are defined in java.

1. final variable
2. final method
3. final class
4. ***final variable:***

It is used to store fixed value in a variable which can’t be further modified. It is similar to ***const*** keyword in C/C++.

Syntax: final datatype variable = value;

class sample

{

final int a=100;

int x=10;

void show()

{

System.out.println(a);

//a++;

}

void disp()

{

System.out.println(x);

x++;

}

}

class finvar

{

public static void main(String[] args)

{

sample s=new sample();

s.show();

s.disp();

}

}

1. ***final method:***

In method overriding a sub class method can hide a super class method. In this case, to prevent method overriding final method was used.

The final method is used to avoid or prevent method overriding. The final methods are always defined in super class only. A final method is preceded by the keyword final followed by method definition.

Syn: final returntype methodname(args)

{

Statements;

}

class samp

{

final void show()

{

System.out.println("Show in class samp");

//a++;

}

}

class sample extends samp

{

void show() // can't be override

{

System.out.println("show in class sample");

}

}

class fmethod

{

public static void main(String[] args)

{

sample s=new sample();

s.show();

}

}

1. ***final class***

It is used to prevent inheritance. If a class is defined as final class then it can’t be extended by any other class ie cant be defined as super class for any other class.

Syn: final class classname

{ variables;

methods;

}

final class samp

{

final void show()

{

System.out.println("Show in class samp");

}

}

class sample extends samp // can’t be extended

{

void disp()

{

System.out.println("show in class sample");

}

}

class fclass

{

public static void main(String[] args)

{

sample s=new sample();

s.show();

}

}

***Abstract Class***

We know that a final method can’t be overridden in a sub class. Exactly opposite to this, a method in super class must override in a sub class. For this, java provides abstract methods.

An abstract method is a method which does not have any body ie without definition. A class which contain the abstract methods is called ‘Abstract class’. An abstract class contain both the defined and undefined (abstract) methods.

An abstract method is preceded by the keyword ‘abstract’ followed by method declaration. The abstract classes are defined as follows:

Syntax:

abstract class classname

{

returntype methodname(argslist) //defined method

{

Statements;

}

abstract returntype methodname(argslist); // undefined method

}

The abstract method must be overridden in at least one sub class. The abstract method is defined in a sub class as follows:

Syntax:

class subclassname extends abstractclassname

{

***public returntype methodname(argslist) // abstract method definition***

***{***

***-----***

***statements;***

***}***

}

abstract class samp

{

void display()

{

System.out.println("display in class samp");

}

abstract void show();

}

class sample extends samp

{

void show()

{

System.out.println("show in class sample");

}

}

class absmeth

{

public static void main(String[] args)

{

sample s=new sample();

s.display();

s.show();

}

}

**Interfaces**

An interface is also a kind of class, that is, collection of variables and methods. In an interface, all the variables are final variables and all the methods are abstract methods. It is similar to an abstract class.

An interface is identified by the keyword interface followed by interface name.

Syntax:

interface interfacename

{

final datatype variable=value;

…….

abstract returntype methodname([argslist]);

------

}

**Implementing an interface**

The abstract methods declared in an interface must be implemented in at least one sub class. For this, java uses the keyword ‘implements’.

Syntax:

class subclassname implements interfacename

{

-------

public returntype methodname([argslist]) //abstract method

{

statements;

}

}

Note:

* A class can be extended from only one super class.
* A class can be implemented from more than one interfaces.
* An interface can be extended from another interface (inheritance)
* The variables defined in an interface can be final variables, if the final keyword is omitted also.
* The methods defined in an interface can be abstract methods, if the keyword abstracted omitted.

Multiple Inheritance:

Syntax:

class subclassname implements interface1,interface2, …

{

Abstract method(s) definition;

}

**Packages**

A package is a collection of classes, interfaces and sub packages. Mainly used for the re-usability of code ie the code written in one program can be accessed from another program. But in inheritance, a super class code can be accessed from a sub class ie in the same program.

**Types of packages:**

The packages are classified into 2 types. They are

1. Pre-defined packages
2. User-defined packages

Pre-defined packages

The packages that are already defined in java are called ‘Pre-defined packages’.

Ex.

java.lang

java.util

java.io

java.net

java.applet

java.awt etc

DataInputStream class:

Java.io

Syntax:

DataInputStream obj = new DataInputStream(System.in);

DataInputStream d = new DataInputStream(System.in);

Methods:

1. read() - reads a single character

syn: char =obj.read();

eg. char a=d.read();

1. readLine() – reads multiple characters

ex.

1. To read a no

int n = Integer.parseInt(d.readLine());

* 1. To read a float
     1. Ex. float x = Float.parseFloat(d.readLine());
  2. To read a double:
     1. double y= Double.parseDouble(d.readLine());
  3. To read a string

Ex. String str = d.readLine();

For creating user defined packages java uses package statement.

Syntax: package packagename;

A package is a collection of classes and interfaces. So, the classes belongs to a package can be defined under the package statement.

Syntax: public class classname

{

variables

methods

}

To work with a package

1. Create a folder with the package name.
2. Copy all .class files into that folder.

Note: The above process can be done by the java compiler automatically. For this, the following command will be applied.

***Javac –d . filename.java***

Import statement:-

To work with either a pre-defined or a user-defined packages the import statement will be used.

Syntax: import package[.subpackage].\*;

package packagename;

public class classname

{

variables

methods

}

* Create a folder with the packagename
* Copy all .class (bytecode) files into the folder
* Import the package using import statement

Javac -d . filename.java

import packagename.subpackagename.classname/\*;

Studdata – sid, sname, course, fee

Marks - sid, a,b,c, tot,avg

**EXCEPTION HANDLING**

When we are typing a program first time then it may lead to errors. These errors may occurred by typing mistakes. These errors may caused to abnormal termination of the program.

Types of Errors

The errors are classified into 2 types:

1. Compile time errors
2. Runtime errors

Compile time errors:

The errors that are raised at the compile time of a program are called’ compile time errors. They have been occurred by typing mistakes or syntax errors.

Ex.

1. Missing semicolon
2. Unterminated string
3. Misplaced else
4. Missing ‘}’

Runtime Errors

The errors that have occurred at runtime of a program are called runtime errors. These errors can automatically raised by JRE (Java Runtime Enironment) and throw to the exception class called ‘Exception’.

The runtime errors are called as exceptions. For handling these exceptions java provides the feature called ‘Exception handling’. It is a mechanism to handle the runtime exceptions.

Types of Exceptions:

They are in two types.

1. Pre-defined Exceptions
2. User-defined Exceptions

**Pre-defined Exceptions:**

the exceptions that are defined in java are called ‘pre-defined’ exceptions. For handling these some of the exception handlers are provided.

**User-defined Exceptions:**

**The** exceptions that are defined by the user are called ‘user-defined’ exceptions. The user defined exceptions are created, raised and handled by the user.

To handle either a user-defined or pre-defined exceptions the exception handling mechanism is used. This mechanism provide the following statements.

1. try
2. catch
3. throw
4. throws
5. finally

**try-**  it contain the statements caused to the exceptions.

Syntax: try

{

statements;

}

**catch** – it is used to catch (handle) the exceptions raised in try block.

Syntax: catch(Exceptiontype obj)

{

Statements;

}

Note: for a try, one or more catch blocks will be defined.

**throw – I**t is used to throw or raise an exception by the user.

Syntax:

throw new Exceptionclassname(message);

note: the pre-defined exceptions are automatically raised based on code.

**throws**: it is used to throw the exception to outside of a class

syntax : throws Exceptionname

**finally**: it is a default catch statement. It will be executed when the catch block may or may not executed.

Syntax :

finally

{

Statements;

}

Structure of try-catch

Syntax:

try

{

----

Statements;

}

catch(Exceptionname obj)

{

Statements;

}

**Pre-defined exceptions:**

The exceptions that are already defined in java are called pre-defined exceptions. These exceptions can automatically raise and throw by JRE. They can be handled by the user. For this, Java provides some of the exception handlers. They are

1. ArithmeticException - to handle the arithmetic exceptions
2. IOException - to handle I/O exceptions
3. ArrayIndexOutOfBoundsException
4. NumberFormatException
5. InputMismatchException
6. InterruptedException
7. FileNotFoundException
8. ArrayStoreIndexException

….

Multiple catch statements:

Syntax:

try

{

---

Statements;

}

catch(exceptiontype1 obj)

{

Statements;

}

catch(Exceptiontype2 obj)

{

Statements;

}

Nested try:

It means one try block may contains another try block.

Syntax:

try

{

----

try

{

Statements;

}

catch(Exceptiontype obj)

{

Statements;

}

-----

}

catch(Exceptiontype obj)

{

statements;

}

User defined Exceptions

The exceptions that have been defined by the user are called user-defined exceptions. To work with user defined exceptions, the exceptions are created, raised and handled by the user.

To create a user defined exception

An exception is the subclass for the super class called Exception. To create an exception the following steps are required.

1. To create a subclass with the exception name extending from Exception class.
2. To define a parameterized constructor with String object as argument.
3. Send the String object to the super class parameterized constructor.

Syntax:

class exceptionclassname extends Exception

{

exceptionclassname(String s)

{

super(s);

}

}

To raise an exception

In order to raise a user defined exception throw statement is used.

Syntax: throw new exceptionclassname(message);

To handle the exception

To handle either pre-defined or user-defined exception catch block is used.

Syntax:

catch(exceptionclassname obj)

{

Statements;

}

import java.util.\*;

class MarksExceedException extends Exception

{

MarksExceedException(String msg)

{

super(msg);

}

}

class MarksException

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int a,b,c;

try

{

System.out.println("Enter 3 subjects marks : ");

a=sc.nextInt();

b=sc.nextInt();

c=sc.nextInt();

if (a>100 || b>100 || c>100)

{

throw new MarksExceedException("Marks entered > 100");

}

else

{

int tot=a+b+c;

double avg=tot/3.0;

System.out.println("Total = "+tot+"\nAverage :"+avg);

}

}

catch (MarksExceedException e)

{

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

}

}

}

**Multithreading**

It means

A thread is a process or a program. Every program must contain one thread called main. Each thread has begin, end and body.

Eg. {

-----

Statements;

}

Thread Life cycle:

The stages occurred while executing a thread is called thread life cycle. A thread life cycle has the following stages (states).

1. New born state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state

It can starts its execution by start() method and killed by stop()

***Runnable State***

If a thread has been waiting for the processor time for its execution then is called runnable state. The threads in runnable state are executed in round-robin manner ie first-cum-first-serve.

***Running state***

If the processor has given time for a thread exection is called running state. For each thread a processor has given a certain time period called time slice. Within the time slice, the thread may or may not completed its execution. If not completed it will be send to runnable state.

***Blocked State***

It means a threads execution has suppressed for some time. A thread has been send to a blocked state using – suspend(), wait() and sleep(). The blocked state thread have been activated by using – resume() and notify() methods.

***Dead State***

If a thread has completed its execution then it automatically send to the dead state, is called natural death.

If a thread has send to a dead state using stop(), is called premature death.

**Creating a Thread**

In Java, the threads are created in 2 ways.

1. Extending from a Thread class
2. Implementing from a Runnable interface

setPriority(int)

1 to 10

1. MIN\_PRIORITY
2. - NORMAL\_PRIORITY

10 – MAX\_PRIORITY

Syn:

class threadclassname extends Thread

{

public void run()

{

}

}

Interrupt()-

Implementing from Runnable interface:

class threadclassname implements Runnable

{

public void run()

{

Statements;

}

}

The user defined thread object can be instantiated with Thread class object.

Syntax: Thread obj=new Thread(threadobj);

Thread obj = Thread.currentThread();

getName() – returns the name of a thread

join() – used to wait for the completion of a thread

thread synchronization:

it is used to execute the threads one after another. For this, synchronized method is used.

Syn: synchronized returntype methodname(args)

{

----

Statements;

}

**Java I/O Streams :**

Java programs perform I/O operations through streams. A stream is an abstraction that either produces or consumes information. In java, the streams are classified into 2 types.

1. Input streams
2. Output streams

***Input Stream***

An input stream extracts data from the file or input device and sends it to the program.

***Output Stream***

An output stream takes data from the program and sends it to the file or output device.

Input stream may refer to a disk file, key board and a network socket. Output stream refer to the console (monitor), disk file and network connection.

Java defined 2 types of stream operations.

1. Byte stream operations
2. Character stream operations

**Byte streams :**

It provides an efficient way for handling input and output bytes. Byte streams are used for reading and writing binary data.

Java byte stream classes are derived from 2 abstract classes called InputStream and OutputStream.

***InputStream classes :***

Input stream classes are used to read 8 bit bytes of information. It supports number of sub classes for various input related functions. They are

1. ByteArrayInputStream
2. BufferedInputStream
3. FileInputStream
4. DataInputStream etc.

***OutputStream classes :***

Output stream classes are derived from the class OutputStream. It can provide several sub classes to perform output operations. They are:

1) ByteArrayOutputStream

2) BufferedOutputStream

3) FileOutputStream

4) DataOutputStream etc.

**Character streams :**

Character streams are used for handling input and output characters. They are in 2 types.

1. Reader
2. Writer

***Reader class :***

This class is a character input stream that reads a sequence of Unicode characters. Several sub classes of this class can be used for performing input operations. The sub classes are:

1. BufferedReader
2. CharArrayReader
3. InputStreamReader
4. FileReader etc.

***Writer class :***

This class is a character output stream that writes a sequence of characters. Several sub classes of this class can be used for performing output operations. The sub classes are:

1. BufferedWriter
2. CharArrayWriter
3. InputStreamWriter
4. PrintWriter
5. FileWriter etc.

**Files**

In general, the user entered data or output of a program is stored in the primary memory called RAM, temporarily. To store that data permanently in the secondary memory (Hard Disk) java uses the feature called files. A file is a collection of related data or records. For handling files, java provides a class called File.

**Class File :**

It is used for representation of file and directory path names.

**Constructor:**

1) File(String pathname) :

It creates a new file instance with given path name .

Syntax: File obj=new File(string);

Ex. File f=new File(“d:\java7-8\abc.txt”)

**Methods :**

***1) boolean createNewFile() :*** used to create a new File.

***2) boolean exists() :*** checks whether the file / directory is exists or not.

***3) String getName() :*** Returns the name of the file / directory

***4) String getPath() :***  Converts this path name into a path name string.

***5) boolean isDirectory() :*** Tests whether the given path name is a directory or not.

***6) boolean isFile() :*** Tests whether the given path name is a normal file or not.

***7) long length() :*** Returns the length of the file denoted by this path name.

***9) boolean mkdir() :*** Creates the directory named by this path name.

**Program: file1*.java***

import java.io.\*;

public class file1

{

public static void main(String args[]) throws Exception

{

File f=new File("d:\\java7-8\\r1.txt");

if(f.exists())

System.out.println("File already exists");

else

{

boolean b=f.createNewFile();

if(b)

System.out.println("File created successfully");

else

System.out.println("Unable to create a file");

}

}

}

**Program: file2*.java***

import java.io.\*;

public class file2

{

public static void main(String args[]) throws Exception

{

File f=new File("E:\\raji\\java\\programs\\raji1");

if(f.exists())

System.out.println("Directory already exists");

else

{

boolean b=f.mkdir();

if(b)

System.out.println("Directory created successfully");

else

System.out.println("Unable to create a Directory");

}

}

}

**FileOutputStream :**

A FileOutputStream is an output stream for writing data onto a file.

**Constructors :**

***1) FileOutputStream(File file) :***

Creates a file output stream with specified file object.

Ex. File f = new File("sample1.txt");

FileOutputStream fos=new FileOutputStream(f);

***2) FileOutputStream(File file, boolean append) :***

Creates a file output stream with specified file object

and append state.

EX. FileOutputStream fos=new FileOutputStream(f,true);

***3) FileOutputStream(String name[, boolean append]) :***

Creates an output file stream with the specified name

and append state.

Ex.

1. FileOutputStream fos=new FileOutputStream(“sample.txt”);
2. FileOutputStream fos=new FileOutputStream(“sample.txt”,true); // for append

**Methods :**

***1) void close() :*** - Closes the file output stream

Ex. fos.close();

***2) void write(int b) :*** - Writes the specified byte data onto a file.

Ex. fos.write(‘a’);

**Note :**

***eof :*** It is a constant which refers to end of file indicator, its value is -1. To get this character from keyboard press

Ctrl + Z.

**Program : file4*.java***

import java.io.\*;

public class file4

{

public static void main(String args[]) throws Exception

{

File f=new File("r2.txt"); // or “r2.dat”

int ch;

boolean b=f.exists();

if(b==false)

f.createNewFile();

FileOutputStream fos=new FileOutputStream(f);

// FileOutputStream fos=new FileOutputStream(f,true);

System.out.println("Enter data to the file (Ctril+Z to stop) ");

ch=System.in.read();

while(ch!=-1)

{

fos.write(ch);

ch=System.in.read(); if not end of file

}

fos.close();

}

}

**Program : file5*.java***

import java.io.\*;

public class file5

{

public static void main(String args[]) throws Exception

{

int ch;

// FileOutputStream fos=new FileOutputStream("r3.txt");

FileOutputStream fos=new FileOutputStream("r3.txt",true);

System.out.println("Enter data to the file (Ctril+Z to stop) ");

ch=System.in.read();

while(ch!=-1)

{

fos.write(ch);

ch=System.in.read();

}

fos.close();

}

}

**Class FileInputStream :**

A FileInputStream is obtains input bytes from a file ie takes input from a file.

**Constructors :**

***1) FileInputStream(File file) : -***

Creates a file input stream with specified ***file*** object

File f = new File(“sample.txt”);

FileInputStream fis = new FileInputStream(f);

***2) FileInputStream(String name) :***

Creates a file input stream with specified ***name*** .

Ex. FileInputStream fis = new FileInputStream(“Sample.txt”);

**Methods :**

***1) void close() : -*** Closes a file

fis.close();

***2) int read() :***

Reads a byte of data from a file

int a = fis.read();

**Program : file6*.java***

import java.io.\*;

public class file6

{

public static void main(String args[]) throws Exception

{

File f=new File("r2.txt");

int ch;

boolean b=f.exists();

if(b==false)

{

System.out.println("File not Found");

System.exit(0);

}

FileInputStream fis=new FileInputStream(f);

System.out.println("Reading data from the file");

ch=fis.read();

while(ch!=-1)

{

System.out.print((char)ch);

ch=fis.read();

}

fis.close();

}

}

**Program : file7*.java***

import java.io.\*;

public class file7

{

public static void main(String args[]) throws Exception

{

FileInputStream fis=new FileInputStream("r3.txt");

int ch;

System.out.println("Reading data from the file");

ch=fis.read();

while(ch!=-1)

{

System.out.print((char)ch);

ch=fis.read();

}

fis.close();

}

}

**FileWriter :**

It is used to write characters onto an output file

**Constructors :**

1. ***FileWriter(File obj, Boolean append) :***

* opens a file object as file with/without append mode.

Ex. File f=new File(“sample.txt”);

FileWriter fw=new FileWriter(f,true);

1. ***FileWriter(String fileName, Boolean append) :***

* Opens a specified file with/without append mode

Ex. FileWriter fw=new FileWriter(“sample.txt”,true);

**Methods :**

***1) void close() :*** to close a file

***2) void write(int b) :*** to write data onto a file

**Program : file8*.java***

import java.io.\*;

public class file8

{

public static void main(String args[]) throws Exception

{

File f=new File("r4.txt");

int ch;

boolean b=f.exists();

if(b==false)

f.createNewFile();

FileWriter fw=new FileWriter(f);

// FileWriter fw=new FileWriter(f,true);

System.out.println("Enter data to the file (Ctril+Z to stop) ");

ch=System.in.read();

while(ch!=-1)

{

fw.write(ch);

ch=System.in.read();

}

fw.close();

}

}

**Program : file9*.java***

import java.io.\*;

public class file9

{

public static void main(String args[]) throws Exception

{

int ch;

// FileWriter fw=new FileWriter("h4.txt");

FileWriter fw=new FileWriter("r4.txt",true);

System.out.println("Enter data to the file (Ctril+Z to stop) ");

ch=System.in.read();

while(ch!=-1)

{

fw.write(ch);

ch=System.in.read();

}

fw.close();

}

}

**Class FileReader :**

It is used to read characters from input file.

**Constructors :**

***1) FileReader (File file) :***

***2) FileReader(String fileName) :***

**Methods :**

***1) void close() :***

***2) int read() :***

**Program : file10*.java***

import java.io.\*;

public class file10

{

public static void main(String args[]) throws Exception

{

File f=new File("r4.txt");

int ch;

boolean b=f.exists();

if(b==false)

{

System.out.println("File not Found");

System.exit(0);

}

FileReader fr=new FileReader(f);

System.out.println("Reading data from the file");

ch=fr.read();

while(ch!=-1)

{

System.out.print((char)ch);

ch=fr.read();

}

fr.close();

}

}

Utility package:

**Class Math :**

The class Math contains methods for performing basic numeric operations such as exponential, square root,etc.

**Fields :**

***1) static double E :***

***2) static double PI :***

**Methods**

***1) static double abs(double a) :***

It Returns absolute value of given value.

***2) static double ceil(double a) :***

It Rounds up the given value

***3) static double floor(double a) :***

It Rounds down the given value.

***4) static double pow(double a, double b) :***

***5) static double sqrt(double a) :***

**Program : math1*.java***

import java.io.\*;

import java.util.\*;

public class math1

{

public static void main(String args[]) throws Exception

{

int a,b;

double d;

BufferedReader br=new BufferedReader(new InputStreamReader(

System.in));

System.out.println("E = "+Math.E);

System.out.println("PI = "+Math.PI);

System.out.print("Enter any integer (abs) : ");

a=Integer.parseInt(br.readLine());

System.out.println("Given value : "+a);

System.out.println("Absolute value : "+Math.abs(a));

System.out.print("\nEnter any integer (sqrt) : ");

a=Integer.parseInt(br.readLine());

System.out.println("Given value : "+a);

System.out.println("Square root : "+Math.sqrt(a));

System.out.print("\nEnter 2 integers (pow) : ");

a=Integer.parseInt(br.readLine());

b=Integer.parseInt(br.readLine());

System.out.println("Exponential value : "+Math.pow(a,b));

System.out.print("\nEnter any double (floor & ceil) : ");

d=Double.parseDouble(br.readLine());

System.out.println("Given value : "+d);

System.out.println("Ceil value : "+Math.ceil(d));

System.out.println("Floor value : "+Math.floor(d));

}

}

**Class Date :**

The class Date represents a specific instant in Date and time.

**Constructors :**

***1) Date() :***

Allocates a Date object and initializes it with the default system date and time.

***2) Date(int year, int month, int date) :***

Allocates a Date object and initializes it with the specified date.

Year : Year – 1900(eg1900+116=2016)

Month : Starts from 0

***3) Date(int year, int month, int date, int hrs, int min, int sec) :***

Allocates a Date object and initializes it with the specified date and time.

**Methods :**

***1) boolean after(Date d) :***

Tests if this date is after the specified date.

***2) boolean before(Date d) :***

Tests if this date is before the specified date.

***3) int compareTo(Date anotherDate) :***

Compares 2 dates.

**Returns :**

0 à 2 dates are equal

> 0 à first date is greater than second date

< 0 à first date is less than second date

**Program : date1*.java***

***import java.io.\*;***

***import java.util.\*;***

***public class date1***

***{***

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

***{***

***Date d1=new Date();***

***Date d2=new Date(116,8,10);***

***Date d3=new Date(117,5,17,5,55,0);***

***System.out.println(d1);***

***System.out.println(d2);***

***System.out.println(d3);***

***}***

***}***

**Program : date2*.java***

import java.io.\*;

import java.util.\*;

public class date2

{

public static void main(String args[])

{

int n;

boolean b;

Date d=new Date();

Date d1=new Date();

Date d2=new Date(119,4,25);

Date d3=new Date(112,11,21,12,30,20);

n=d.compareTo(d1);

if(n==0)

System.out.println("d == d1");

else

System.out.println("d != d1");

b=d1.before(d2);

if(b)

System.out.println("d1 < d2");

else

System.out.println("d1 > d2");

b=d1.after(d3);

if(b)

System.out.println("d1 > d3");

else

System.out.println("d1 < d3");

}

}

**Class Calendar :**

Calendar is an abstract base class for converting between a Date object and a set of integer fields such as YEAR, MONTH, DAY, HOURS and so on.

**Fields :**

1. static int JANUARY to DECEMBER
2. static int SUNDAY to SATURDAY
3. static int AM and PM
4. static int AM\_PM
5. static int DATE (day of the month)
6. static int DAY\_OF\_MONTH (day of the month)
7. static int DAY\_OF\_WEEK

10 static int DAY\_OF\_YEAR

11 static int HOUR

12 static int MINUTE

***13*** static int SECOND

14 static int WEEK\_OF\_MONTH

15 static int WEEK\_OF\_YEAR

Etc.

**Methods :**

***1) static Calendar getInstance() :***

Gets a Calendar using the default time zone and locale.

***2) String toString() :***

Returns a String representation of the Calendar.

**Program : cal1*.java***

import java.io.\*;

import java.util.\*;

public class cal1

{

public static void main(String args[])

{

Calendar c=Calendar.getInstance();

System.out.println(c);

}

}

**Program : cal2*.java***

import java.io.\*;

import java.util.\*;

public class cal2

{

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

{

Calendar c=Calendar.getInstance();

String s=c.toString();

String s1[]=s.split(",");

for(int i=0;i<s1.length;i++)

{

System.out.println("\n"+s1[i]);

if(i%10==0)

System.in.read();

}

}

}

This is to print the data in next page after printing 20 records

**COLLECTION FRAME WORK**(java.util package)

A collection frame work is provided in java.util package. All the classes in Collection frame work have been divided into 3 groups.

1.Lists:- It stores a group of elements like an array.

Ex:- Vector, Stack, LinkedList, ArrayList,etc.

2. Sets :- HashSet, LinkedHashSet,TreeSet,etc.

Note: Sets will not allow duplicate elements, But Lists will allow duplicates.

3.Maps:- It stores elements in Key, value pairs.

ex:- HashMap,Hashtable.

\* Collection object stores only references of other objects. It cannot store primitive data types ie char, int, float etc. They

store only objects.

**Lists**

**Class Vector :**

The vector class implements a growable array of objects. The size of vector can grow or shrink as needed to accommodate adding and removing items after the vector has been created.

**Constructors :**

***1) Vector() :***

Constructs an empty vector with default size 100 and its

standard capacity increment is zero.

***2) Vector(int initialcapacity) :***

Constructs an empty vector with the specified initial

capacity and with its capacity increment is Zero.

***3) Vector(int initialcapacity, int capacityincrement) :***

Constructs an empty vector with the specified initial

capacity or capacity increment.

**Methods :**

***1) boolean add(Object o) :***

Appends the specified element to the end of this vector.

***2) void addElement(Object obj) :***

Adds the specified component to the end of this vector,

increasing its size by one.

***3) void clear() :***

Removes all elements from this vector.

***4) Object get(int index) :***

Returns the element at the specified position in this vector.

***5) int indexOf(Object element) :***

It returns index of specified object.(If element Not

found return -1)

***6) void insertElementAt(Object obj, int index) :***

Inserts the specifed object as a component in this vector

at the specified index.

***7) boolean isEmpty() :***

Tests if the vector has no components.

***8) boolean remove(Object o) :***

Removes the first occurrence of the specified element in this vector.

***9) void setElementAt(Object obj, int index) :***

Set the component at the specified index of this vector to be the specified object.

***16) Object[] toArray() :***

Returns an array containing all of the elements in this vector.

**creation of collection Objects**

class\_name <type\_name> object\_name =new class\_name<typename>();

eg:

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

**Program : vtr*.java***

import java.io.\*;

import java.util.\*;

public class vtr

{

public static void main(String args[]) throws Exception

{

int opt,n,pos,k,i;

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

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

while(true)

{

System.out.println("\nVECTOR MENU");

System.out.println("-------------------");

System.out.println("1.Add");

System.out.println("2.Display");

System.out.println("3.Insert");

System.out.println("4.Delete");

System.out.println("5.Search");

System.out.println("6.Replace");

System.out.println("7.Exit");

System.out.println("-------------------");

System.out.print("\nEnter your option : ");

opt=Integer.parseInt(br.readLine());

switch(opt)

{

case 1:

System.out.print("\nEnter element to add : ");

n=Integer.parseInt(br.readLine());

v.addElement(n);

System.out.println("\nElement added");

break;

case 2:

if(v.isEmpty())

System.out.println("\nVector is empty");

else

{

Object obj1[]=v.toArray();

System.out.print("\nGiven Vactor elements : \n");

for(i=0;i<obj1.length;i++)

{

System.out.print(obj1[i]+" ");

}

}

break;

case 3:

System.out.print("\nEnter element to insert : ");

n=Integer.parseInt(br.readLine());

System.out.println("\nEnter position : ");

pos=Integer.parseInt(br.readLine());

v.insertElementAt(n,pos);

System.out.println("\nElement is inserted");

break;

case 4:

if(v.isEmpty())

System.out.println("\nVector is empty");

else

{

System.out.print("\nEnter element to delete : ");

n=Integer.parseInt(br.readLine());

Integer obj=new Integer(n);

boolean b=v.remove(obj);

if(b)

System.out.println("\nElement is deleted");

else

System.out.println("\nElement not found");

}

break;

case 5:

if(v.isEmpty())

System.out.println("\nVector is empty");

else

{

System.out.print("\nEnter element to search : ");

n=Integer.parseInt(br.readLine());

k=v.indexOf(n);

if(k==-1)

System.out.println("\nElement not found");

else

System.out.println("\nElement found at position : "+k);

}

break;

case 6:

System.out.print("\nEnter element to replace : ");

n=Integer.parseInt(br.readLine());

System.out.print("\nEnter position : ");

pos=Integer.parseInt(br.readLine());

v.setElementAt(n,pos);

System.out.println("\nElement is replaced");

break;

case 7:

System.exit(0);

break;

}

}

}

}

**Class ArrayList**

Resizable-array implementation of the List.

**Constructors**

1) ArrayList()

Constructs an empty list with an initial capacity of

ten.

2) ArrayList(int initialCapacity)

Constructs an empty list with the specified initial

capacity.

**Methods**

1) void add(int index, Object element)

Inserts the specified element at the specified position

in this list.

2) boolean add(Object o)

Appends the specified element to the end of this list.

3) void clear()

Removes all of the elements from this list.

4) Object get(int index)

Returns the element at the specified position in this

list.

5) int indexOf(Object elem)

Searches for the first occurence of the given

argument, testing for equality using the equals method.

5) boolean isEmpty()

Tests if this list has no elements.

6) Object remove(int index)

Removes the element at the specified position in this

list.

7) void removeRange(int fromIndex, int toIndex)

Removes from this List all of the elements whose

index is between fromIndex, inclusive and toIndex,

exclusive.

8) Object set(int index, Object element)

Replaces the element at the specified position in this

list with the specified element.

9) int size()

Returns the number of elements in this list.

10) Object[] toArray()

Returns an array containing all of the elements in

this list in the correct order.

**program**

import java.io.\*;

import java.util.\*;

class Alist

{

public static void main(String args[])

{

int opt,n,pos,k,i;

ArrayList<Integer> AL=new ArrayList<Integer>();

Scanner sc=new Scanner(System.in);

while(true)

{

System.out.println("\nArrayList MENU");

System.out.println("----------------------");

System.out.println("1.Add");

System.out.println("2.Display");

System.out.println("3.Insert");

System.out.println("4.Delete");

System.out.println("5.Search");

System.out.println("6.Replace");

System.out.println("7.Exit");

System.out.println("-------------------");

System.out.print("\nEnter your option : ");

opt=sc.nextInt();

switch(opt)

{

case 1:

System.out.print("\nEnter element to add : ");

n=sc.nextInt();

AL.add(n);

System.out.println("\nElement is added");

break;

case 2:

if(AL.isEmpty())

System.out.println("\nList is empty");

else

{

Object a[]=AL.toArray();

System.out.print("\nGiven Elements : ");

for(i=0;i<a.length;i++)

{

System.out.print(a[i]+" ");

}

}

break;

case 3:

System.out.print("\nEnter element to insert : ");

n=sc.nextInt();

System.out.print("\nEnter position : ");

pos=sc.nextInt();

AL.add(pos,n);

System.out.println("\nElement is inserted");

break;

case 4:

if(AL.isEmpty())

System.out.println("\nList is empty");

else

{

System.out.print("\nEnter element to delete : ");

n=sc.nextInt();

boolean b=AL.remove(new Integer(n));

if(b)

System.out.print("\nElement is deleted");

else

System.out.print("\nElement not found");

}

break;

case 5:

if(AL.isEmpty())

System.out.println("\nList is empty");

else

{

System.out.print("\nEnter element to search : ");

n=sc.nextInt();

k=AL.indexOf(n);

if(k==-1)

System.out.println("\nElement not found");

else

System.out.print("\nElement found at position : "+k);

}

break;

case 6:

if(AL.isEmpty() )

System.out.println("List is empty");

else

{

System.out.print("\nEnter element to replace : ");

n=sc.nextInt();

System.out.print("\nEnter position : ");

pos=sc.nextInt();

AL.set(pos,n);

System.out.println("\nElement is replaced");

}

break;

case 7:

System.exit(0);

break;

}

}

}

}

**APPLETS**

An applet is also a java program which is useful to develop window based applications ie GUI applications. For developing applets java provides a package called java.applet. every applet is a subclass for the super class called Applet.

Java applets does not required the main method whereas the stand-alone programs start with main() method.

The stand-alone programs are run by the java interpreter ie java. But the applet programs are run by java enabled browser or appletviewer.

To develop an applet:

To develop an applet program the following steps are required.

1. To create an applet:

Java uses two packages called java.applet and java.awt. an applet is a sub class for the super class called Applet which provided in java.applet package.

For displaying text on an applet window java uses the paint method which takes Graphics class as argument. It is provided in java.awt package.

Syntax:

class appletclassname extends Applet

{

public static void paint(Graphics g)

{

Statements;

}

}

***To display text***:- drawString method is used

drawstring()- used to display text in an applet window, method of Graphics class.

syntax: obj.drawString(text,x,y)

1. To compile an applet - uses javac compiler to compile an applet

Syntax: javac filename.java

It generates the bytecode file called **.class**

1. Embed the java code into an html file

An applet program can be executed by java enabled browser. This browser can recognize the html code. so, create an html file and embed the java code into that file by using <applet> tag.

Syntax:

<html>

<applet code=”.classfilename” width=value height=value>

</applet>

</html>

Save file as .html or .htm

1. To run an applet:

The applets are run by the ***appletviewer*** tool. It can be used as follows:

Syntax: appletviewer filename.html

**Class Graphics :**

The Graphics class is the abstract base class for all graphics contexts that allow an application to draw components.

Note: This class belongs to java.awt package.

**Methods :**

***1) void drawString(String str,int x,int y) :***

Draws the text given by the specified string.

***2) drawLine(int x1,int y1,int x2,int y2) :***

Draws the line using the current color, between the points (x1,y1) and (x2,y2).

***3) drawRect(int x,int y,int width,int height) :***

Draws the outline of the specified rectangle.

***4) fillRect(int x,int y,int width,int height) :***

Fills the specified rectangle.

***5) drawOval(int x,int y,int width,int height) :***

Draws the outline of a oval.

***6) fillOval(int x,int y,int width,int height) :***

Fills the specified oval.

**Note :**

If width and height values of oval are equal then oval becomes circle.

**Program : applet3*.java***

import java.applet.\*;

import java.awt.\*;

/\* <applet code=applet3 height=500 width=500>

</applet> \*/

public class applet3 extends Applet

{

public void paint(Graphics g)

{

g.drawString("RAJI",100,250);

showStatus("welcome to Java Applets");

}

}

**Class Color :**

It is used to provide create custom colors for applying the drawing objects in the applet window.

**Constructor :**

Color obj=new Color(int r,int g,int b)

Color c1=new Color(255,0,0);

Color c2=new Color(0,200,200);

***Color(int r,int g,int b) :***

Creates opaque RGB color with the specified red, green and blue values in the range (0 to 255)

**Class Font : -** used to define custom fonts.

The Font class represents fonts.

***Syn: Font obj=newFont(String name,int style,int size)***

Creates a new Font from the specified name, style and size.

**Font Names :**

1. Dialog
2. DialogInput
3. MonoSpaced
4. Serif
5. SeanSerif .

**Font Styles :**

1. PLAIN
2. BOLD
3. ITALIC
4. BOLD + ITALIC

Font f=new Font(“Dialog”,Font.BOLD,15);

**Note :**

Font styles are static fields, hence call the above field with the class name Font.( eg: Font.PLAIN,..)

**Class Label :**

A Label object is a component for placing text in a container. It is used to display Titles or Column titles. It has read-only text.

**Fields :**

1. static int CENTER : center align
2. static int LEFT : left align
3. static int RIGHT : right align

**Constructors :**

***1) Label****()* **-**  Constructs an empty label.

***2) Label(String text)***  -     Constructs a new label with the specified text with left

justified.

***3) Label(String text, int alignment)***-     Constructs a new label with the specified

text with the specified alignment.

**Methods :**

***1) String getText() :*** - Gets the text of the Label.

***2) Void setText(String text) :*** - Sets the text for a Label.

***3) void setAlignment(int alignment) :*** - Sets the alignment for a label.

**Program : apwt1.*java***

import java.applet.\*;

import java.awt.\*;

/\*<applet code=awt1 height=200 width=100>

</applet> \*/

public class awt1 extends Applet

{

Label lb1,lb2,lb3;

public void init()

{

lb1=new Label();

lb1.setText("Label-1");

lb1.setAlignment(Label.CENTER);

lb2=new Label("Label-2");

lb2.setAlignment(Label.LEFT);

lb3=new Label("Label-3",Label.RIGHT);

}

public void start()

{

add(lb1);

add(lb2);

add(lb3);

}

}

**Class Button :**

This class creates a labeled button. The application cause some action to happen when the button is pushed.

**Constructors :**

***1) Button() :*** Constructs a button without label.

***2) Button(String str) :*** Constructs a button with specified label.

**Methods :**

***1) void setLabel(String str) :*** Sets the button’s label ie text.

***2) void addActionListener(ActionListener l) :*** used to receive action events from a button.

***3) String getActionCommand() :*** Returns the name of the action event fired by this button.

***4) String getLabel() :*** Gets the label of a button.

**Program : apwt2.*java***

import java.applet.\*;

import java.awt.\*;

/\* <applet code=apwt2 height=200 width=200>

</applet> \*/

public class apwt2 extends Applet

{

Button b1,b2;

public void init()

{

b1=new Button();

b1.setLabel("Button-1");

b2=new Button("Button-2");

}

public void start()

{

add(b1);

add(b2);

}

}

**Class TextField :**

A text field object is a text component that allows for editing of a single line of text.

**Constructors :**

***1) TextField() :*** Constructs a new text field.

***2) TextField(int column) :***  Constructs a text field with the specified number of columns.

***3) TextField(String str) :*** Constructs a text field with the default text

***4) TextField(String str,int column) :***  Constructs a new text field initialized with specified text to be displayed and wide enough to hold the specified number of columns.

**Methods :**

***1) void setColumns(int columns) :***  Sets the number of columns

***2) void setText(String str) :***  Sets the text to be displayed

***3) String getText() :*** Returns the text in a text field

***4) void setBackground(Color c) :*** Sets the background color

**Program : apwt5.*java***

import java.applet.\*;

import java.awt.\*;

/\* <applet code=apwt5 height=200 width=500>

</applet>

\*/

public class apwt5 extends Applet

{

TextField t1,t2,t3,t4;

public void init()

{

t1=new TextField();

t1.setColumns(10);

t1.setText("JAVA");

t2=new TextField(10);

t2.setText("WELCOME");

t3=new TextField("BDPS");

t4=new TextField("WELCOME TO BDPS",10);

}

public void start()

{

add(t1);

add(t2);

add(t3);

add(t4);

}

}

**Class Checkbox :**

The Checkbox is a graphical component that can be either an on(true) or off(false) state.

**Constructors :**

***1) Checkbox() :*** Creates a check box with no label.

***2) Checkbox(String label) :***  Creates a check box with specified label.

***3) Checkbox(String label,boolean state) :***

Creates a check box with specified label and specified state.

***4) Checkbox(String label,boolean state,CheckboxGroup group) :***

Creates a check box with specified label and sets to the specified state and in the specified check box group.

**Methods :**

***1) String getLabel() :***  Gets the label of the Checkbox.

***2) boolean getState() :***  Determines where this check box is in the “on” or “off” state.

***3) void setLabel(String Lable) :***  Sets this check box’s label to be the string argument.

***4) void setState(boolean state) :*** Sets the state of this check box to be the specified state.

***5) void addItemListener(ItemListener l) :***

Adds the specified itemlistener to receive item events from this check box

**Program : apwt6.*java***

import java.applet.\*;

import java.awt.\*;

/\* <applet code=apwt6 height=200 width=500>

</applet> \*/

public class apwt6 extends Applet

{

Checkbox c1,c2,c3;

public void init()

{

c1=new Checkbox();

c1.setLabel("DOT NET");

c2=new Checkbox("JAVA");

c2.setState(true);

c3=new Checkbox("ORACLE",false);

}

public void start()

{

add(c1);

add(c2);

add(c3);

}

}

**Interface ItemListener :**

This Listener interface for receiving item events.

**Methods :**

***1) void itemStateChanged(ItemEvent e) :***

Invoked when an item has been selected or deselected by the user.

**Class ItemEvent :**

A semantic event which indicates that the item was selected or deselected.

**Methods :**

***1) Object getItem() :***

Returns the item affected by the Event.

**Program : apwt7.*java***

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

/\* <applet code=apwt7 height=200 width=500>

</applet> \*/

public class apwt7 extends Applet implements ItemListener

{

Checkbox c1,c2,c3;

TextField t;

public void init()

{

c1=new Checkbox();

c1.setLabel("DOT NET");

c2=new Checkbox("JAVA");

c2.setState(true);

c3=new Checkbox("ORACLE",false);

t=new TextField(20);

c1.addItemListener(this);

c2.addItemListener(this);

c3.addItemListener(this);

}

public void start()

{

add(c1);

add(c2);

add(c3);

add(t);

}

public void itemStateChanged(ItemEvent e)

{

if(c1.getState())

t.setText(c1.getLabel());

else if(c2.getState())

t.setText(c2.getLabel());

else if(c3.getState())

t.setText(c3.getLabel());

else

t.setText(" ");

}

}

**Class CheckboxGroup :**

The CheckboxGroup is used to group together a set of checkbox buttons.

**Constructors :**

***1) CheckboxGroup() :***

Creates a new instance of checkbox group.

**Methods :**

***1) Checkbox getSelectedCheckbox() :***

Gets the current choice from this check box group.

**Program : apwt8.*java***

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

/\* <applet code=apwt8 height=200 width=200>

</applet> \*/

public class apwt8 extends Applet implements ItemListener,ActionListener

{

Checkbox cb1,cb2,cb3;

CheckboxGroup cbg;

TextField tf;

Button bn;

public void init()

{

cbg=new CheckboxGroup();

cb1=new Checkbox("SQL",true,cbg);

cb2=new Checkbox("DOT NET",true,cbg);

cb3=new Checkbox("ORACLE",true,cbg);

tf=new TextField(20);

bn=new Button("SELECT");

cb1.addItemListener(this);

cb2.addItemListener(this);

cb3.addItemListener(this);

bn.addActionListener(this);

}

public void start()

{

add(cb1);

add(cb2);

add(cb3);

add(bn);

add(tf);

}

public void itemStateChanged(ItemEvent ie)

{

if(cb1.getState())

tf.setText(cb1.getLabel());

if(cb2.getState())

tf.setText(cb2.getLabel());

else

tf.setText(cb3.getLabel());

}

public void actionPerformed(ActionEvent e)

{

if(cb1.getState())

showStatus(cb1.getLabel());

if(cb2.getState())

showStatus(cb2.getLabel());

else

showStatus(cb3.getLabel());

}

}

**Class Choice :**

The Choice class presents a pop up menu of choices. The current choice is displayed as the title of menu.

**Constructors :**

***1) Choice() :***

Creates a new choice menu.

**Methods :**

***1) void add(String item) :***

Adds an item to this choice menu.

***2) void addItem(String item) :***

***3) void remove(int position) :*** Removes an item at a given position

***4) void remove(String item) :*** Removes the first occurrence of item from the choice menu.

***5) void removeAll() :***  Removes all items from the choice menu.

**Program : apwt10.*java***

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

/\* <applet code=apwt10 height=200 width=200>

</applet> \*/

public class apwt10 extends Applet implements ActionListener

{

Choice c;

Button bn1,bn2,bn3;

TextField tf;

public void init()

{

c=new Choice();

c.add("santoor");

c.add("pears");

c.add("dove");

tf=new TextField(10);

bn1=new Button("ADD");

bn2=new Button("REMOVE");

bn3=new Button("CLEAR");

bn1.addActionListener(this);

bn2.addActionListener(this);

bn3.addActionListener(this);

}

public void start()

{

add(c);

add(tf);

add(bn1);

add(bn2);

add(bn3);

}

public void actionPerformed(ActionEvent ae)

{

Button b=(Button)ae.getSource();

if(b==bn1)

c.add(tf.getText());

if(b==bn2)

c.remove(c.getSelectedItem());

if(b==bn3)

c.removeAll();

}

}

**Layouts :**

Layout is a design used to place components on a container.

**Class Container :**

It is a generic AWT container class, which holds AWT components.

**Constructors :**

***1) Container() :***

Constructs a new container.

**Methods :**

***1) component add(Component comp) :***

Appends the specified component to the end of this container.

***2) component add(Component comp,int index) :***

Adds the specified component to the given position.

***3) component add(String name,Component comp) :***

Adds the specified component to this container.

***4) void setLayout(LayoutManager ob) :***

Sets the layout manager for this container.

**Class FlowLayout :**

A flow layout arranges components in a left-right manner, just like a lines of text in a paragraph.

**Fields :**

***1) static int CENTER :***

This value indicates that each row of component should be centered.

***1) static int LEFT :***

This value indicates that each row of component should be left justified.

***3) static int RIGHT :***

This value indicates that each row of component should be right justified.

**Constructors :**

***1) FlowLayout() :***

Constructs a new flow layout with a center alignment and a default 5-unit horizontal and vertical gap.

***2) FlowLayout(int align) :***

Constructs a new flow layout with specified alignment and default 5-unit horizontal and vertical gap.

***3) FlowLayout(int align,int gap,int hgap,int vgap) :***

Constructs a new flow layout with specified alignment and the indicated and the indicated horizontal and vertical gap.

**Note :**

The default layout of Applet is FlowLayout.

**Program : apwt18.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

/\*<applet code=apwt18 height=200 width=200>

</applet> \*/

public class apwt18 extends Applet

{

Button b1,b2;

FlowLayout f;

public void init()

{

b1=new Button("Button-1");

b2=new Button("Button-2");

// f=new FlowLayout(FlowLayout.CENTER); // or f=new FlowLayout();

// f=new FlowLayout(FlowLayout.LEFT);

// f=new FlowLayout(FlowLayout.RIGHT);

f=new FlowLayout(FlowLayout.CENTER,20,20);

this.setLayout(f);

}

public void start()

{

add(b1);

add(b2);

}

}

**Class GridLayout :**

The grid layout class is layout manager that arranges components in rows and columns manner.

**Constructors :**

***1) GridLayout() :***

Creates a new grid layout with a default of one column per component in a single row.

***2) GridLayout(int rows,int columns) :***

Creates a new grid layout with specified number of rows and columns.

***3) GridLayout(int rows,int columns,int hgap,int vgap) :***

Creates a grid layout with specified number of rows, columns and with the indicated horizontal and vertical gap.

**Program : apwt19.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

/\*<applet code=apwt19 height=200 width=200>

</applet> \*/

public class apwt19 extends Applet

{

Button b1,b2,b3,b4;

GridLayout g;

public void init()

{

// g=new GridLayout();

// g=new GridLayout(2,2);

g=new GridLayout(2,2,30,30);

b1=new Button("Button-1");

b2=new Button("Button-2");

b3=new Button("Button-3");

b4=new Button("Button-4");

this.setLayout(g);

}

public void start()

{

add(b1);

add(b2);

add(b3);

add(b4);

}

}

**Class BorderLayout :**

A border layout layouts a container arranging and resizing its components to fit in 5 regions. North, South, East, West and Center. Each region may contain no more than one component and is identified by a corresponding constant : NORTH, SOUTH, EAST, WEST AND CENTER.

**Fields :**

1. **static string CENTER :** The center layout constraint
2. **static string EAST :** The east layout constraint

1. **static string NORTH :** The north layout constraint
2. **static string SOUTH :** The south layout constraint
3. **static string WEST :** The west layout constraint

**Constructors :**

***1) BorderLayout() :***

Constructs a new border layout with no gap between the components.

***2) BorderLayout(int hgap,int vgap) :***

Constructs a border layout with the specified gap between components.

**Program : apwt20.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

/\*<applet code=apwt20 height=200 width=200>

</applet> \*/

public class apwt20 extends Applet

{

Button b1,b2,b3,b4,b5;

public void init()

{

//this.setLayout(new BorderLayout());

this.setLayout(new BorderLayout(20,20));

b1=new Button("North");

b2=new Button("East");

b3=new Button("West");

b4=new Button("South");

b5=new Button("Center");

}

public void start()

{

add(b1,BorderLayout.NORTH);

add(b2,BorderLayout.EAST);

add(b3,BorderLayout.WEST);

add(b4,BorderLayout.SOUTH);

add(b5,BorderLayout.CENTER);

}

}

**Class Panel :**

It is a simplest container class. A panel provides space in which an application can attach any other components including other panels.

**Constructors :**

***1) Panel() :***

Creates a new panel using default layout manager.

***2) Panel(LayoutManager layout) :***

Creates a new panel using specified layout manager.

**Program : awt21.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

/\*<applet code=awt21 height=200 width=200>

</applet> \*/

public class awt21 extends Applet

{

Panel p;

Button b1,b2,b3,b4,b5;

public void init()

{

this.setLayout(null);

p=new Panel();

p.setLayout(new BorderLayout());

//p.setLayout(new BorderLayout(20,20));

b1=new Button("North");

b2=new Button("East");

b3=new Button("West");

b4=new Button("South");

b5=new Button("Center");

p.setBounds(100,50,200,100);

}

public void start()

{

p.add(b1,BorderLayout.NORTH);

p.add(b2,BorderLayout.EAST);

p.add(b3,BorderLayout.WEST);

p.add(b4,BorderLayout.SOUTH);

p.add(b5,BorderLayout.CENTER);

add(p);

}

}

**Class CardLayout :**

A card layout object is a layout manager for a container. It reads each component in the container as a card. Only one card is visible at a time and the container acts as a stack of cards.

**Constructors :**

***1) CardLayout() :***

Creates a new card layout with gaps of size zero.

***2) CardLayout(int hgap,int vgap) :***

Creates a new card layout with the specified gaps.

**Methods :**

***1) void addLayoutComponent(String name,component comp) :***

It adds the component comp to the layout, associating it with the string specified by name

***2) void first(Container parent) :***

Flips to the first card of the container.

***3) void last(Container parent) :***

Flips to the last card of the container.

***4) void next(Container parent) :***

Flips to the next card of the specified container.

***5) void previous(Container parent) :***

Flips to the previous card of the specifed container.

***6) void show(Container parent,String name) :***

Flips to the component that was added to this layout with the specified name using addLayoutManager.

**Program : apwt22.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

/\* <applet code=apwt22 height=500 width=500>

</applet> \*/

public class apwt22 extends Applet implements ActionListener

{

Button b1,b2,b3,b4,b5,first,last,previous,next,show;

Panel p,pn,pt,p1,p2,p3,p4,p5;

CardLayout c;

GridLayout g,g1;

TextField tf;

public void init()

{

g=new GridLayout(3,1);

this.setLayout(g);

c=new CardLayout();

p=new Panel();

p.setLayout(c);

p1=new Panel();

p1.setLayout(new GridLayout(1,1));

p2=new Panel();

p2.setLayout(new GridLayout(1,1));

p3=new Panel();

p3.setLayout(new GridLayout(1,1));

p4=new Panel();

p4.setLayout(new GridLayout(1,1));

p5=new Panel();

p5.setLayout(new GridLayout(1,1));

b1=new Button("card-1");

b2=new Button("card-2");

b3=new Button("card-3");

b4=new Button("card-4");

b5=new Button("card-5");

p1.add(b1);

p2.add(b2);

p3.add(b3);

p4.add(b4);

p5.add(b5);

p.add("card1",p1);

p.add("card2",p2);

p.add("card3",p3);

p.add("card4",p4);

p.add("card5",p5);

pn=new Panel();

g1=new GridLayout(3,2);

pn.setLayout(g1);

first=new Button("First");

last=new Button("Last");

previous=new Button("Previous");

next=new Button("Next");

show=new Button("Show");

first.addActionListener(this);

last.addActionListener(this);

previous.addActionListener(this);

next.addActionListener(this);

show.addActionListener(this);

pn.add(first);

pn.add(last);

pn.add(previous);

pn.add(next);

pn.add(show);

tf=new TextField(20);

pt=new Panel();

pt.add(tf);

}

public void start()

{

add(p);

add(pt);

add(pn);

}

public void actionPerformed(ActionEvent ae)

{

Button b=(Button)ae.getSource();

if(b==first)

c.first(p);

if(b==last)

c.last(p);

if(b==previous)

c.previous(p);

if(b==next)

c.next(p);

if(b==show)

c.show(p,tf.getText());

}

}

**Window Programming :**

Java provides a set of API functions for doing window programming.

They are : Window,Frame.

**Class Window :**

A Window object is a top level window with no border and no menu bar. The default layout of window is BorderLayout.

A Window must have either a Frame dialog or another Window defined as its owner when it is constructed.

**Constructors :**

***1) Window(Frame owner) :***

Constructs a new invisible window with the specified Frame as its owner.

***2) Window(Window owner) :***

Constructs a new invisible window with the specified Window as its owner.

**Methods :**

***1) void addWindowListener(WindowListener wl) :***

Adds the specified WindowListener to receive window events from this window.

***2) void hide() :***

Hides this window.

***3) void show() :***

Makes the window visible.

(The above 2 methods are depricated methods.)

**Class Frame :**

A Frame is a top level window with a title and a border. The Frame is the sub class of Window.

**Connstructors :**

***1) Frame() :***

Constructs a new instance of frame that is initially invisible.

***2) Frame(String title) :***

Constructs a new initially invisible frame object wih the specified title.

**Methods :**

***1) void setTitle(String title) :***

Sets the title of this Frame to the specified string

***2) void setSize(int width,int height) :***

Resize this component with specified width and height.

***3) void setVisible(boolean b) :***

Shows or hides this components depending on the value of parameter b.

**Program : apwt23.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

/\*

<applet code=apwt23 height=200 width=200>

</applet>

\*/

class wnd extends Frame

{

public wnd()

{

super("My Window");

//super();

}

}

public class apwt23 extends Applet

{

wnd w;

public void init()

{

w=new wnd();

}

public void start()

{

w.setSize(200,200);

w.setVisible(true);

// w.setTitle("MyWindow"); if super();

}

}

**Program : apwt24.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

/\*

<applet code=awt24 height=300 width=300>

</applet>\*/

class wnd extends Frame

{

public wnd(String st)

{

this.setTitle(st);

}

}

public class awt24 extends Applet implements ActionListener

{

wnd w;

Button b1,b2;

public void init()

{

this.setLayout(null);

w=new wnd("my window");

b1=new Button("show");

b2=new Button("hide");

b1.addActionListener(this);

b2.addActionListener(this);

w.setBounds(50,50,200,150);

b1.setBounds(10,200,100,50);

b2.setBounds(150,200,100,50);

}

public void start()

{

w.setVisible(true);

add(b1);

add(b2);

}

public void actionPerformed(ActionEvent ae)

{

Button b=(Button)ae.getSource();

/\*

if(b==b1)

w.show();

else

w.hide();

\*/

if(b==b1)

w.setVisible(true);

else

w.setVisible(false);

}

}

**Standalone window application :**

In the above program, the window is displayed in an applet. It is also possible to display a window directly without using applet.

**Program : wnd1.*java***

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

class window extends Frame

{

public window()

{

super("MYWINDOW");

}

}

public class wnd1

{

public static void main(String args[])

{

window w=new window();

w.setLayout(null);

Button b=new Button("exit");

listener l=new listener(b);

b.addActionListener(l);

w.setBounds(100,100,300,300);

b.setBounds(100,100,75,35);

w.add(b);

w.setVisible(true);

}

}

class listener implements ActionListener

{

Button bt;

listener(Button btt)

{

bt=btt;

}

public void actionPerformed(ActionEvent ae)

{

Button bn=(Button)ae.getSource();

if(bn==bt)

System.exit(0);

}

}

(here System.exit(0) is used to terminate the program, otherwise we have to type Ctrl + C at the command propmpt to terminate the program.)

**Interface WindowListener :**

This listener interface for receiving window events.

**Methods :**

***1) void windowActivated(WindowEvent we) :***

Invoked when the window is set to be the active window.

***2) void windowClosed(WindowEvent we) :***

Invoked when the window has been closed as the result of calling dispose on the window.

***3) void windowClosing(WindowEvent we) :***

Invoked when the user attempts to close the window from the window's system menu.

**Program : wnd2.*java***

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

class window extends Frame

{

public window()

{

super("MY WINDOW");

}

}

public class wnd2

{

public static void main(String args[])

{

window w=new window();

w.setLayout(null);

wadapter wa=new wadapter();

w.addWindowListener(wa);

w.setBounds(10,10,300,300);

w.setVisible(true);

}

}

class wadapter extends WindowAdapter

{

wnd2 wd;

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

}

**Interface MouseListener :**

This listener interface for receiving mouse events

**Methods :**

***1) void mouseClicked(MouseEvent e)***

Invoked when the mouse button has been clicked (pressed,released) on a

component

***2) void mouseEntered(MouseEvent e)***

Invoked when the mouse enters a component

***3) void mouseExited(MouseEvent e)***

Invoked when the mouse exists a component

***4) void mousePressed(MouseEvent e)***

Invoked when the mouse button has been pressed on a component

***5) void mouseReleased(MouseEvent e)***

Invoked when the mouse button has been released on a component

**Class MouseEvent :**

An event which indicates that a mouse action occurred on a component.

**Adapter implementation techniques :**

There are 3 techniques to implement an adapter to a component.

1. Outer class technique.

2. Inner class technique.

3. Anonomous inner class technique.

**Outer class technique :**

In this technique the adapter class is defined outside the component class.

**Program : adpt1.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

/\*<applet code=adpt1 height=200 width=200>

</applet> \*/

public class adpt1 extends Applet

{

public void init()

{

madapter a=new madapter(this);

this.addMouseListener(a);

}

}

class madapter extends MouseAdapter

{

adpt1 ad;

public madapter(adpt1 ap)

{

ad=ap;

}

public void mouseClicked(MouseEvent e)

{

ad.showStatus("Mouse Clicked.........");

}

}

**Inner class technique :**

In this technique, the adapter class is defined inside the component class.

**Program : adpt2.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

/\*<applet code=adpt2 height=200 width=200>

</applet> \*/

public class adpt2 extends Applet

{

public void init()

{

madapter a=new madapter();

this.addMouseListener(a);

}

class madapter extends MouseAdapter

{

public void mouseClicked(MouseEvent e)

{

showStatus("Mouse Clicked.........");

}

}

}

**Anonomous inner class technique :**

In this technique, the event handler of a give adapter will define the process of adding listener to a component.

**Program : adpt3.*java***

import java.io.\*;

import java.applet.\*;

import java.awt.\*;

import java.awt.event.\*;

/\*<applet code=adpt3 height=200 width=200>

</applet> \*/

public class adpt3 extends Applet

{

public void init()

{

this.addMouseListener(new MouseAdapter()

{

public void mouseClicked(MouseEvent e)

{

showStatus("Mouse Clicked");

}

});

}

}

**Class MenuBar :**

This class is used to add menubar to a frame.

**Constructors :**

***1) MenuBar() :***

Creates a new menu bar.

**Methods :**

***1) Menu add(Menu m) :***

Adds the specified menu to the menu bar.

**Class Menu :**

A Menu object is a pull down menu component that is deployed from a menu bar.

**Constructors :**

***1) Menu() :***

Constructs a new menu with empty label.

***2) Menu(String label) :***

Constructs a new menu with the specified label.

**Methods :**

***1) MenuItem add(MenuItem m) :***

Adds the specified menu item to this menu.

**Class MenuItem :**

All items in a Menu must belong to the class MenuItem.

**Constructors :**

***1) MenuItem(String label) :***

Constructs a menu item with specified label and no keyboard shortcut.

**Class CheckboxMenuItem :**

This class represents a check box that can be included in a menu.

**Constructors :**

***1) CheckboxMenuItem(String label) :***

Creates a CheckboxMenuItem with the specified label.

**Program : menu1.*java***

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

class MyWindow extends Frame

{

public MyWindow()

{

super("My Window");

MenuBar mb=new MenuBar();

setMenuBar(mb);

Menu f=new Menu("File");

f.add(new MenuItem("New"));

f.add(new MenuItem("Open"));

f.add(new MenuItem("Save"));

f.add(new MenuItem("Close"));

mb.add(f);

Menu e=new Menu("Edit");

e.add(new MenuItem("Cut"));

e.add(new MenuItem("Copy"));

e.add(new MenuItem("Paste"));

Menu sub=new Menu("Special");

sub.add(new MenuItem("First"));

sub.add(new MenuItem("second"));

e.add(new CheckboxMenuItem("Testing"));

e.add(sub);

mb.add(e);

}

}

public class menu1

{

public static void main(String args[])throws Exception

{

MyWindow mw=new MyWindow();

myadapter ma=new myadapter();

mw.addWindowListener(ma);

mw.setBounds(10,10,300,300);

mw.setVisible(true);

}

}

class myadapter extends WindowAdapter

{

public void windowClosing(WindowEvent e)

{

System.exit(0);

}

}

**Program : menu2.*java***

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

class MyWindow extends Frame implements ActionListener,ItemListener

{

MenuBar mb;

Menu f,e,sub;

TextField tf;

MenuItem n,o,s,c,ct,cp,ps,ft,se;

CheckboxMenuItem sp;

public MyWindow()

{

super("My Window");

mb=new MenuBar();

setMenuBar(mb);

tf=new TextField(15);

setLayout(null);

tf.setBounds(20,200,100,30);

add(tf);

f=new Menu("File");

n=new MenuItem("new");

o=new MenuItem("open");

s=new MenuItem("save");

c=new MenuItem("close");

n.addActionListener(this);

o.addActionListener(this);

s.addActionListener(this);

c.addActionListener(this);

f.add(n);

f.add(o);

f.add(s);

f.add(c);

mb.add(f);

e=new Menu("Edit");

ct=new MenuItem("cut");

cp=new MenuItem("copy");

ps=new MenuItem("paste");

ct.addActionListener(this);

cp.addActionListener(this);

ps.addActionListener(this);

e.add(ct);

e.add(cp);

e.add(ps);

sub=new Menu("special");

ft=new MenuItem("first");

se=new MenuItem("second");

ft.addActionListener(this);

se.addActionListener(this);

sub.add(ft);

sub.add(se);

e.add(sub);

sp=new CheckboxMenuItem("testing");

sp.addItemListener(this);

e.add(sp);

mb.add(e);

}

public void actionPerformed(ActionEvent ae)

{

String st=ae.getActionCommand();

tf.setText(st);

}

public void itemStateChanged(ItemEvent ie)

{

tf.setText(sp.getLabel());

}

}

public class menu2

{

public static void main(String args[])throws Exception

{

MyWindow mw=new MyWindow();

myadapter ma=new myadapter();

mw.addWindowListener(ma);

mw.setBounds(10,10,300,300);

mw.setVisible(true);

}

}

class myadapter extends WindowAdapter

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

}

**Class FileDialog :**

FileDialog class displays a dialog window from which the user can select a file.

**Fields :**

***1) static int LOAD :***

A constant indicates that, the purpose of file dialog window is to located a file from which to read.

***2) static int SAVE :***

A constant indicates that, the purpose of file dialog window is to located a file to which to write.

**Constructors :**

***1) FileDialog(Frame parent) :***

Creates a file dialog for loading a file.

***2) FileDialog(Frame parent,String title) :***

Creates a file dialog window with specified title for loading a file.

***3) FileDialog(Frame parent,String title,int mode) :***

Creates a file dialog window with specified title for loading a file / saving a file.

**Methods :**

***1) String getFile() :***

Gets the selected file of this file dialog.

***2) public void show() :***

Makes the dialog visible.

**Program : fd.*java***

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

class MyWindow extends Frame implements ActionListener

{

FileDialog f;

Button b;

TextField tf;

public MyWindow()

{

super("My Window");

this.setLayout(null);

tf=new TextField(15);

tf.setBounds(100,100,100,30);

b=new Button("select");

b.addActionListener(this);

b.setBounds(100,200,50,20);

add(tf);

add(b);

// f=new FileDialog(this);

// f=new FileDialog(this,"FileDialog");

// f=new FileDialog(this,"SaveDialog",FileDialog.SAVE);

}

public void actionPerformed(ActionEvent ae)

{

f.show();

tf.setText(f.getFile());

}

}

public class fd

{

public static void main(String args[])throws Exception

{

MyWindow mw=new MyWindow();

myadapter ma=new myadapter();

mw.addWindowListener(ma);

mw.setBounds(10,10,300,300);

mw.setVisible(true);

}

}

class myadapter extends WindowAdapter

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

}

**Java Swing**

Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based(GUI) applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, etc.

**Class Jcomponent**

**T**his class is the super class for all swing components except top level containers. Th e top level containers are JFrame,JApplet and JDialog.

**Class Container**

A generic Abstract Window Toolkit(AWT) container ,It

Can contain other AWT and Swing components.

**Class JApplet**

public class JApplet extends Applet

An extended version of Applet.It supports for the JFC/Swing component architecture.

Container getContentPane()

This Method belongs to JApplet class,It Returns the Container object for this Japplet.

JFrame

JLabel

JButton

JTextField

JCheckBox

JRadioButton

JList,etc.

Program :

import javax.swing.\*;

import java.awt.\*;

/\* <applet code=sw1 height=200 width=500>

</applet> \*/

public class sw1 extends JApplet

{

JButton b1,b2;

Container cnt;

public void init()

{

cnt=this.getContentPane();

cnt.setLayout(new FlowLayout());

b1=new JButton("Button-1");

b2=new JButton("Button-2");

}

public void start()

{

cnt.add(b1);

cnt.add(b2);

}

}

Program:

import javax.swing.\*;

public class sw2

{

public static void main(String[] args)

{

JFrame f=new JFrame();

JButton b=new JButton("click");

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

f.add(b);

f.setSize(400,500);

f.setLayout(null);

f.setVisible(true);

}

}

**JRadioButton**

An implementation of a radio button -- an item that can be selected or deselected, and which displays its state to the user.

**constructos**

JRadioButton(String text)

Creates an unselected radio button with the specified text.

JRadioButton(String text, boolean selected)

Creates a radio button with the specified text and selection state.

**Class ButtonGroup**

This class is used to create a multiple-exclusion scope for a set of buttons. Creating a set of buttons with the same ButtonGroup object means that turning "on" one of those buttons turns off all other buttons in the group.

**constructor**

ButtonGroup()

Creates a new ButtonGroup.

**Method**

void add(Button b)

Adds the button to the group.

Program

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

/\* <applet code=sw3 height=200 width=500>

</applet> \*/

public class sw3 extends JApplet implements ItemListener

{

JRadioButton b1,b2,b3;

JTextField t;

Container cnt;

ButtonGroup bg;

public void init()

{

cnt=this.getContentPane();

cnt.setLayout(new FlowLayout());

bg=new ButtonGroup();

b1=new JRadioButton("DOT NET");

b2=new JRadioButton("JAVA");

b3=new JRadioButton("ORACLE",true);

bg.add(b1);

bg.add(b2);

bg.add(b3);

b1.addItemListener(this);

b2.addItemListener(this);

b3.addItemListener(this);

t=new JTextField(20);

}

public void start()

{

cnt.add(b1);

cnt.add(b2);

cnt.add(b3);

cnt.add(t);

}

public void itemStateChanged(ItemEvent e)

{

JRadioButton rb=(JRadioButton)e.getItem();

t.setText(rb.getText());

}

}

**JComboBox**

A ComboBox is a drop-down list.

**constructors**

JComboBox()

Creates a JComboBox.

JComboBox(Object[] items)

Creates a JComboBox that contains the elements in the specified array.

Methods

void addItem(Object ob)

Adds an item to the item list.

Object getSelectedItem()

Returns the current selected item.

Program

import javax.swing.\*;

import java.awt.\*;

/\* <applet code=sw4 height=200 width=200>

</applet> \*/

public class sw4 extends JApplet

{

JComboBox cb;

Container cnt;

public void init()

{

cnt=this.getContentPane();

cnt.setLayout(new FlowLayout());

cb=new JComboBox();

cb.addItem("Santoor");

cb.addItem("Pears");

cb.addItem("Dove");

}

public void start()

{

cnt.add(cb);

}

}

**JList**

A component that allows the user to select one or more objects from a list.

**Constructors**

JList()

Constructs a JList with an empty list.

JList(ListModel dataModel)

Constructs a JList that displays the elements in the specified List Model;

JList(Object[] listData)

Constructs a JList that displays the elements in the specified array.

**Methods**

Object getSelectedValue()

Returns the first selected value, or null if the selection is empty.

**Class DefaultListModel**

This class implements the java.util.Vector API

**Constructor**

DefaultListModel() ;

**Methods**

**void add(int index, Object element)**

Inserts the specified element at the specified position

in this list.

**void addElement(Object obj)**

Adds the specified component to the end of this list.

**Interface ListSelectionListener**

The listener that's notified when a lists selection value changes.

**Method**

Public void valueChanged(ListSelectionEvent e)

Called whenever the value of the selection changes.

**class ListSelectionEvent**

An event that characterizes a change in the current selection.

Program

import javax.swing.\*;

import javax.swing.event.\*;

import java.awt.\*;

import java.awt.event.\*;

/\* <applet code=sw5 height=200 width=250>

</applet> \*/

public class sw5 extends JApplet implements ListSelectionListener

{

Container cnt;

JTextField tf;

JList lst;

DefaultListModel dlm;

public void init()

{

cnt=this.getContentPane();

cnt.setLayout(new FlowLayout());

tf=new JTextField(20);

dlm=new DefaultListModel();

dlm.addElement("Oracle");

dlm.addElement("Java");

dlm.addElement("DotNet");

lst=new JList(dlm);

lst.addListSelectionListener(this);

}

public void start()

{

cnt.add(lst);

cnt.add(tf);

}

public void valueChanged(ListSelectionEvent e)

{

String st=(String)lst.getSelectedValue();

tf.setText(st);

}

}

**Class JScrollPane**

Provides a scrollable view of a lightweight component.

A JScrollPane manages a viewport, optional vertical and horizontal scroll bars

**Fields**

HORIZONTAL\_SCROLLBAR, HORIZONTAL\_SCROLLBAR\_ALWAYS, HORIZONTAL\_SCROLLBAR\_AS\_NEEDED, HORIZONTAL\_SCROLLBAR\_NEVER

VERTICAL\_SCROLLBAR, VERTICAL\_SCROLLBAR\_ALWAYS, VERTICAL\_SCROLLBAR\_AS\_NEEDED, VERTICAL\_SCROLLBAR\_NEVER

**Constructors**

JScrollPane(Component view)

Creates a JScrollPane that displays the contents of the specified component, where both horizontal and vertical scrollbars appear whenever the component's contents are larger than the view.

JScrollPane(Component view, int vsbPolicy, int hsbPolicy)

Creates a JScrollPane that displays the view component in a viewport whose view position can be controlled with a pair of scrollbars.

Program

import javax.swing.\*;

import javax.swing.event.\*;

import java.awt.\*;

import java.awt.event.\*;

/\* <applet code=sw6 height=300 width=300>

</applet> \*/

public class sw6 extends JApplet implements ListSelectionListener,ActionListener

{

Container cnt;

JTextField tf;

JList lst;

JButton b;

DefaultListModel dlm;

JScrollPane jsp;

int hsb,vsb;

public void init()

{

cnt=this.getContentPane();

cnt.setLayout(new FlowLayout());

tf=new JTextField(20);

dlm=new DefaultListModel();

dlm.addElement("Oracle");

dlm.addElement("Java");

dlm.addElement("DotNet");

lst=new JList(dlm);

lst.addListSelectionListener(this);

vsb=JScrollPane.VERTICAL\_SCROLLBAR\_AS\_NEEDED;

hsb=JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED;

jsp=new JScrollPane(lst,vsb,hsb);

b=new JButton("AddElement");

b.addActionListener(this);

}

public void start()

{

cnt.add(jsp);

cnt.add(tf);

cnt.add(b);

}

public void valueChanged(ListSelectionEvent e)

{

String st=(String)lst.getSelectedValue();

tf.setText(st);

}

public void actionPerformed(ActionEvent ae)

{

dlm.addElement(tf.getText());

tf.setText(“ “);

}

}