JAVA

Introduction:-

Java is a distributed technology developed by James Gosling in the year 1995. Java is a general purpose programming language it is used to develop desktop applications, mobile applications ,Enterprise applications ,Big-data processing, Embded systems and so on.

Initally JAVA is called as oak language, it’s name was aalready as registered company, so the name was renamed as JAVA.

Java is an Island name in Indonesia , the name called JAVA Island.

JAVA is a purely obvject oriented programming language because if we want to construct a Java program atleast class or object is mandatory to write a JAVA program, without class or object it is impossible to write a JAVA program.

APPLICATIONS:-

There are many devices where JAVA is currently used. Some of them are as foloows:-

\*Desktop applications such as Adopbe reader, Media player, Antivirue etc…

\*Web applications sucxh as [www.google.com](http://www.google.com), [www.facebook.com](http://www.facebook.com) etc…

\*Enterprise applications such as banking applications etc…

\*Mobile applications , games, smart cards, embded systems etc…

Types of JAVA:-

JAVA released to the market in three categories:-

1.J2SE(JAVA to Standard Edition)+

2.J2ME(JAVA to Micro or Mobile edition)

3.J2EE(JAVA to Enterprise Edition)

J2SE:-

J2SE is basically used for developing client side applications or programs. By using J2SE we can develop stand alone applications(window based applications or desktop applications) such as adobe reader, meia player, antivirus etc…

J2ME:-

J2ME is used for developing server side applications or programs. J2ME is used for creating mobile applications. Currently J2ME, Android are used for creating Mobile applications.

J2EE:-

J2EE is used for developing server side applications or programs. By using J2EE we can develop enterprise applications such as banking applications.

Features of JAVA:-

1.Simple:-Learning and practice JAVA is easy because it uses c and c++ sayntax.

2.Object oriented programming language:- JAVA is purely object oriented programming language because if we want top write a JAVA program atleast a class or object is mandatory tom write a java program.

3.Distributed:- JAVA is designed for use on network, it has extensive library which works in Agrement with TCP/IP protocol(Transmission control protocol and internet protocol).

4.Secure:- JAVA is designed for use on internet. JAVA enables the construction of virus free, tamper free systems.

5.Robust:- JAVA programs will not crash because of it’s exception handling and memory management features.

6.Platform independent:- A platform or technology is said to be platform independent if and only if which can run on all available operating systems with respect to it’s development and compilation.

7.Portable:- JAVA does not have implementation dependent aspects and it gives same results on any machine.

8.Architectural neutral language:- JAVA byte code is not machine dependent, it can run on any machine with any processor and with any Os(operating system).

9.High performance:- Along with the enterpuner there will be JIT(just in time) compiler which enables the speed of execution.

10.Multi threaded:- Executing different parts of the program simultaneously is called Multi threading. This is an essential feature to design server side programs.

11.Dynamic:- sign server side programs.

11.Dynamic:- We can develop programs in JAVA which dynamically change on internet.

Programming structure

\*Comments:-

Comments are description about the aim and features of the program. Comments increases readability of a program. There are two types of comments in JAVA.

\*Single line comment:- These comment starts with ‘ // ‘ .

\*Multi line comment:- These comments start with ‘ /\* ‘ and ends with ‘ \*/ ‘.

Class in JAVA:-

\*A class is a blueprint of an object or a class is a plan of software objects.

\*A class is a logical representation(imaginaryrepresentation).

\*A class can be declared by using keyboard called “CLASS”.

\*A class is a user defined data type.

SYNTAX FOR A CLASS:-

CLASS<CLASS-NAME>

{

Variable declaration

Construction declaration

Method declaeation

}

Initialization:-

Assigning a value to a variable at very first time is called initialization.

Eg:- a=10;

Variable declaration:-

Data types followed by the variable name is known as variable declaration.

Int a; //initialising a variable

Int a =10; //assigning a value to the variable

Constructor declaration:-

\*A constructor is a place for initialising variables during object construction.

\*Constructor declarations rules

1.Constructor has no return type.

2.Constructor looks like a method.

3.Constructor name and class name both are same.

Example:-

class simple // class name

{

int a , b ; //variable declaration

Simple()

{

a=10;

b=20; //constructor declaration

}

int add() //method declaration

{

return a+b;

}

}

OBJECT:-

\*Physical representation of a class is called an object(Instant of a class is called an object).

\*Software objects can be created and stored in computer’s memory called RAM.

\*Software objects can be represented in the form of real time objects.

\*Object is a collection of variables and methods.

OBJECT Construction:-

Syntax: class name Variable name=new class name();

Eg:- Simple obj = new Simple( );

\*The new operator creates an object in the computer memory by using constructor.

\*The new operator return a reference (memory location) to the object is created. Here obj is called object reference.

\*Using object reference we can access variable and methods in that particular object.

\*Sample code for accessing variables methods:-

The variable and methods can be accessed by using dot operator.

int sum=obj.a //accessing a variable.

int sum=obj.add(); //accessing a method

//print statement in java

System.out.println(“Hello java”);

//Java main method

The main method is the starting point of JVM(JAVA virtual machine) to start execution of a program without main method.

JVM will not execute the program.The syntax of main method is

public static void main(String args[])

{

//STATEMENTS

}

public 🡪access specifier

static 🡪keyboard

void 🡪Return type

main 🡪method

args 🡪array of string type

Steps to follow to write a JAVA program:-

Step1:-Start---------run--------notepad

Step2:- Type JAVA program code

Step3:- save the program as:

Filename: program.java

Save as type: all files

Step4:- compile the program as :C:\>java program.java

Step5:- Run the program as: C:\>java program name

Ex:-

class First

{

public static void main(String args[])

{

System.out.println("Welcome to java")

}

}

output:-

open command prompt and type

C:\Users \>E:

E:\cd javaex // javaex is the file name

E:\javaex > javac First.java

E:\ javaex > java First

Welcome to java //output

\*Write a java program tom find sum of two numbers

class Sum

{

public static void main(Strinng args[])

{

int x=10,y=20;

int z=x+y;

System.out.println(z);

System.out.println("sum of two numbers ="+z);

}

}

output:-

sum of two numbers=30

Formatting output:-

class Format

{

public static void main(String args[])

{

int a=1,b=2,c=3,d=4;

System.out.println(a+"\t"+b);

System.out.println(b+"\n"+b);

System.out.println(":"+c);

System.out.println(); //this throws cursor to the next line

System.out.println(d);

}

}

output:-

1. 1 22

2. 2

3. :3

4.

5. 4

\*Write a program to find sum of two numbers using object construction

class Simple

{

int a,b;

Simple()

{

a=10;

b=20;

}

int add()

{

return a+b;

}

public static void main(String args[])

{

Simple obj=new Simple(); //creating a simple object

int sum=obj.add();

System.out.println(sum);

System.out.println("sum of two numbers ="+sum);

}

}

output:-

30

sum of two numbers=30

Hashcode:-

Hashcoide is a unique identificatioon number alloted to the objects by the JVM .This hashcode number is also called reference number.

To know the hashcode number(reference) of an object we can use hashcode method of object class as shown imn below:

Employee e1 = new Employee(); //e1 is the reference of the Employee object

System.out.println(ee1.hashcode()); //display hash code stored in e1

Naming conventions:-

Naming conventions specify the rules to be followed by a java programmer while writing the names of packages, classes, methods etc...

Rule no.1:-

Package names are written in small letters.

eg: java.io ,java.lang , java.awt etc.

Rule no.2:-

Each word of class name and intereference name starts with a capital.

eg: ample, add two numbers

Rule no.3:-

Methiod name starts with small letters then each word starts with a capitalp.

eg: Sum() ,SumTwoNumbers() , minValue()

Rule no.4:-

Variable names also follow thew same above method Rules.

eg: Sum , Count , totalCount

Rule no.5:-

Constants should be written using all capital lettters.

eg; PI , COUNT

Rule no.6:-

Keywords are reserved wprds and written in small letters.

eg: int,short,float, public, void

Data Types:-

The classification of data items is called data types. JAVA defines eight simple types of data-byte , short, int, long, char, double, and boolean. These can be put in four groupas:

1.INTEGER DATA TYPE:-

These data type stores integer numbers

DATA TYPE Memory size Range

Byte 1 byte -128 to 127

Short 2 byte -32768 to 32767

Int 4 byte -214783648 to 2147483647

Long 8 byte -9223372036854775808 to 9223372036854775807

2.FLOAT DATA TYPE:-

The data type stores handling floating point numbers

DATA TYPE Memory size Range

FLOAT 4 byte -3.4e38 t0 3.4e38

DOUBLE 8 byte -1.7e308 t0 1.7e308

3.CHARACTER DATA TYPE;-

This data type represents a single character. Char data type in java uses two bytes of memeory also called UNICODE System.

DATA TYPE Memory size Range

char 2 byte 0 to 65535

4.BOLEAN DATA TYPE:-

It can handle truth values either true or false

eg:-boolean response=true;

\*OPERATOR:-

Operator in java is a symbol which is used to perform operations between the operands or variables.

Ex;- +,-,\*,/ etc...

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

-Unary operator

-Arithmtic operator

-Shift operator

-Relational operator

-Bitwise operator

-Logical operator

-Ternary operator

-Assignment operator

Operator type category precedence

Unary postfix exp++ , exp--

prefix ++exp , --exp //exp-expression

Arithmatic Multiplicative \* / %

Additive + -

Shift Shift << >>

Relational Comparison < > < = >=

Equality == !=

Bitwise bitwise AND &

bitwise exclusive OR ^

bitwise inclusive OR |

Logical logical AND &&

logical OR ||

Ternary ternary ? :

Assignment assignment =+=-=\*=/=%= etc

JAVA UNARY Operator:-

The java unary operatir requires only one operand. Unary operators are used to perform various operations i.e...,

-incrementing and decrementing a value by one

-negating an expression

-inverting the value of boolean

Ex:-

class Demo

{

public static void main(Srtring args[])

{

int x=10;

System.out.println(x++);

System.out.println(++x);

System.out.println(x--);

System.out.println(--x);

}

}

**Output:-**

**22**

**21**

Ex:-

class Demo1

{

public static void main(String args[])

{

int a=10;

int b=10;

System.out.println(a++ + ++a);

System.out.println(b++ + b++);

}

}

Output:-

10

12

12

10

**~ and ! (negation and inclusion)**

**~=-n+10**

**!=opposite**

Ex:-

class Demo2

{

public static void main(String args[])

{

int a=10;

int b=-10;

boolean c= true;

boolean d= false;

System.out.println(~a);

System.out.println(~b);

System.out.println(!c);

System.out.println(!d);

}

}

**Output:-**

-11

9

false

true

**Arithmatic operator:-**

Arithmatic operators are used to perform addition, subtraction, multiplication and division. They act as basic mathematical operations.

Ex:-

class Demo3

{

public static void main(String args[])

{

int a=10;

int b=5;

System.out.println(a+b);

System.out.println(a-b);

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

System.out.println(a/b);

System.out.println(a%b);

}

}

**Output:-**

15

5

50

2

0

Ex:-

class Demo4

{

public static void main(String args[])

{

System.out.println(10\*10/5+3-1\*4/2);

}

}

**Output:-**

21

**JAVA LEFT SHIFT OPERATOR:-**

The left shift operator << is used to shift all the bits in a value to the left side of a specified number of times.

//java left shift operator -acts as a multiplier with 2(i.e..2^)

class Demo5

{

public static void main(String args[])

{

System.out.println(10<<2);

System.out.println(10<<3);

System.out.println(20<<2);

System.out.println(15<<4);

}

}

**Output:-**

40

80

80

240

**JAVA RIGHT SHIFT OPERATOR:-**

The right shift operator >> is used to move left operands value to write by the number of bits specified by the right operand.

//java Right shift operator - acts as a divider with 2 power(i.e..2^)

class Demo6

{

public static void main(String args[])

{

System.out.println(10>>2);

System.out.println(20>>2);

System.out.println(20>>3);

}

}

**Output:-**

2

5

2

**AND operator**

&& - logical AND

& - bitwisw AND

The logical AND operator does not check second condition, if first condition is false. It checks second second condition only if first one is true.

The bitwise AND operator always checks both conditions whether first condition is true or false.

class Demo7

{

public static void main(String args[])

{

int a =10;

int b=5;

int c=20;

System.out.println(a<b&&a++<c);

System.out.println(a);

System.out.println(a<b&a++<c);

System.out.println(a);

}

}

**Output:-**

false

10

false

11

**OR operator**

|| - logical OR

| - bitwise or

The logical OR operator does not check second condition if first condition is true. It checks second condition if first condition is false

T bitwise OR operator always checks second condition whether first condition is true or false.

class Demo8

{

public static void main(String args[])

{

int a=10;

int b=5;

int c=20;

System.out.println(a>b||a++<c);

System.out.println(a);

System.out.println(a>b|a++<c);

System.out.println(a);

}

}

**Output;-**

true

10

true

11

**TERNARY OPERATOR**

Ternary operator is used as one linear replacement for if then else statements. It is the only condition operator which takes three operations.

Syntax:- EXP-1 ? EXP-2 : EXP-3 ;

class Demo9

{

public static void main(String args[])

{

int a=2;

int b=5;

int min=(a<b)?a:b;

System.out.println(min);

}

}

**Output;-**

2

Ex:-

class Demo10

{

public static void main(String args[])

{

int a=12;

int b=15;

int c=7;

int large=a>b?(a>c?a:c):(b>c?b:c);

System.out.println("The largest number is"+ large);

}

}

**Output:-**

The largest number is15

**Assignment operator(=)**

Assignment operator is one of the most common operator. It is used to assign the values on it’s right to the operand on it’s left.

class Demo11

{

public static void main(String args[])

{

int a=10;

a+=3;

System.out.println(a);

a=a-4;

System.out.println(a);

a\*=2;

System.out.println(a);

a/=2;

System.out.println(a);

}

}

**Output:-**

13

9

18

9

CONTROL STATEMENTS IN JAVA:-

Control statements are the statements which alter the flow of execution and provide better control to the program on the flow of execution.

\*SELECTION CONTROL STATEMENTS:-

In java ,if statements is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition is true then the statements are executed if condition is false it won’t executes.

In JAVA there are four types of IF statements which are given below:

1.Simple IF statement.

2.IF-else stattements

3.IF-ELSE-IF statements.

4.Nested IF statements.

IF Statement:-

In IF statement first condition is tested ,if condition is true then the respective block is executed if condition is false then it won’t executes.

Syntax:-

If(condition)

{

Ststements;

}

Ex:-

class Demo12

{

public static void main(String args[])

{

int x=10;

int y=12;

if(x+y>20)

{

int add=x+y;

System.out.println("x+y="+add);

System.out.println("x+y is greater than 20");

}

}

}

**Output:-**

x+y=22

x+y is greater than 20

Ex:-

class Demo13

{

public static void main(String args[])

{

int num=6;

if(num%2==0)

{

System.out.println(num+"is an even number");

}

}

}

**Output:-**

6is an even number

IF-ELSE Statement:-

The IF-ELSE statement is a alternative ti IF statement, which uses another block of code that is else block. The ELSE block is executed if the condition of the IF block is evaluated as false.

SYNTAX:-

If

{

Statement 1;

}

else

{

Ststement2;

}

Ex:-

class Demo14

{

public static void main(String args[])

{

int num=7;

if(num%2==0)

{

System.out.println(num+"is a even number");

}

else

{

System.out.println(num+"is an odd number");

}

}

}

**Output:-**

7is an odd number

IF-ELSE-IF Statement:-

The IF-ELSE-IF statement contains the if statement followed by the multiple else statements.

In the IF-ELSE-IF ladder statement we are checking the condition multiple number of times.

Based on the condition the appropriate block of is executed.

SYNTAX:-

IF

{

Statement 1 ;

}

ELSE IF

{

STATEMENT 2 ;

}

Else if

{

Statement 3 ;

}

ELSE

{

Statement n ;

}

EX :-

class Demo15

{

public static void main(String args[])

{

int a=5,b=7,c=6;

if(a>b&&a>c)

{

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

}

else if(b>c)

{

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

}

else

{

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

}

}

}

**Output:-**

b is big

NESTED IF STATEMENT :-

In nested if the statement the if statement contains an if or if else statement inside another if.

SYNTAX :-

IF

{

Statement 1 ;

}

IF

{

Statement 2 ;

}

ELSE

{

Statement 3 ;

}

}

ELSE

{

Statement n ;

}

EX :-

class Demo16

{

public static void main(String args[])

{

String address="Delhi,India";

if(address.endsWith ("India"))

{

if(address.endsWith("meerut"))

{

System.out.println("Your city is meerut");

}

else if(address.endsWith("Noida"))

{

System.out.println("Your city is Noida");

}

else

{

System.out.println("NONE");

}

}

else

{

System.out.println("your are not living in India");

}

}

}

**Output:-**

NONE

`

SWITCHED Statements:-

The switch statements is similar to the if else if statements.The switch statements contains multiple block of code called cases and a single case is executed based on the expoession.

Syntax:-

Switch

{

Case value 1:

Break;

Case value 1:

Break;

Case value n:

Break;

Default:

}

Here depending on the value of the expression a perticular corresponding case will be executed.

EX:-

//color demo

class Demo17

{

public static void main(String args[])

{

char color='b';

switch (color)

{

case 'r':System.out.println("red"); break;

case 'g':System.out.println("green"); break;

case 'b':System.out.println("blue"); break;

case 'y':System.out.println("yellow"); break;

case 'w':System.out.println("white"); break;

default:System.out.println("no color selected");

}

}

}

**Output:-**

blue

JAVA ITTERATION STATEMENTS:-

Itteration statements are FOR, WHILE, DO WHILE. These statements are used to repeat some set of instructions specified number of times called loops.

A loop repeatedly executes the same set of instructions until a termination condition is met.

WHILE LOOP:-

While loop repeats a group of statements as long as condition is true. Once condition is false, the loop is terminated. In while loop, the condition is tested first if it is true then only the statements are executed.

While loop is called as entry control loop.

SYNTAX:-

Whilr

{

Statements;

Increment or decrement(++ or --);

}

Ex:-

//Program to write numbers from 1 to 20

class Demo18

{

public static void main(String args[])

{

int i=1;

while(i<=20)

{

System.out.println(i + "\t");

i++;

}

}

}

**Output:-**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

Ex:-

//Program to print even numbers

class Demo19

{

public static void main (String args[])

{

int i=0;

System.out.println("printing the list of first 10 even numbers");

while(i<=10)

{

System.out.println(i);

i=i+2;

}

}

}

**Output:-**

printing the list of first 10 even numbers

0

2

4

6

8

10

Ex:-

//Program to print odd numbers

class Demo20

{

public static void main (String args[])

{

int i=1;

System.out.println("printing the list of first 10 odd numbers");

while(i<=10)

{

System.out.println(i);

i=i+2;

}

}

}

**Output:-**

printing the list of first 10 even numbers

1

3

5

7

9

Ex:-

class Demo21

{

public static void main(String args[])

{

int i=10;

while(i>1)

{

System.out.println(i);

i--;

}

}

}

**Output:-**

10

9

8

7

6

5

4

3

2

//if you replace i-- with i++ we can get infinity values

Ex:-

class Demo22

{

public static void main(String args[])

{

int i=1;

int sum = 0;

while(i<=10)

{

sum +=i;

System.out.println(i);

i++;

}

System.out.println("sum of 1-10 numbers is="+ sum);

}

}

**Output:-**

1

2

3

4

5

6

7

8

9

10

sum of 1-10 numbers is=55

DO-WHILE LOOP:-

DO-WHILE loop represents a group of statements as long as condition is true. In do-while loop, the statements are executed first and then condition is tested.

Do-while loop is also called as exit control loop.

Syntax:-

Do

{

Statements;

Increment/decrement(i++ or i--);

}

Wjhile(condition);

Ex:-

class Demo23

{

public static void main(String args[])

{

int i=1;

do

{

System.out.println(i);

i++;

}

while(i<=20);

}

}

**Output:-**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

Ex:-

class Demo24

{

public static void main(String args[])

{

int i=20;

do

{

System.out.println(i);

i--;

}

while(i>=1);

}

}

**Output:-**

20

19

18

17

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1

Ex:-

class Demo25

{

public static void main(String args[])

{

do

{

System.out.println("Infinite do-while loop");

}

while(true);

}

}

**Output:-**

Infinite do-while loop

Infinite do-while loop

Infinite do-while loop

Infinite do-while loop

// prints infinite times

Ex:-

class Demo26

{

public static void main(String args[])

{

int i=1;

do

{

if(i%2==0)

{

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

}

i++;

}

while(i<11);

}

}

**Output:-**

2

4

6

8

10

FOR LOOP:-

For loop is also same as do while or while loop ,but it is more compact syntactically. The for loop executes a group of statements as long as condition is true.

Syntax:-

For(expression 1; expression 2; expression3)

{

Statements;

}

Here expression 1 is used to initialize the variables, expression 2 is used for condition checking and expression 3 is used for increment or decrement variable value.

Ex:-

class Demo27

{

public static void main(String args[])

{

int i;

for(i=1;i<=20;i++)

{System.out.println(i);

}

}

}

**Output:-**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

NESTED FOR LOOP:-

If we have a for loop inside the another for loop it is known as nested for loop.

The inner loop executes completely when outer loop executes.

Ex:-

class Demo28

{

public static void main(String args[])

{

int i,j;

for(i=1;i<=3;i++)

{

for(j=1;j<=3;j++)

{

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

}

}

}

}

**Output:-**

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

Ex:-

class Demo29

{

public static void main(String args[])

{

int i,j;

for(i=1;i<=5;i++)

{

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

{

System.out.println("\*");

}

System.out.println( );

}

}

}

**Output:-**

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

Ex:-

class Demo30

{

public static void main(String args[])

{

int item=6,j;

for(int i=1;i<=item;i++)

{

for(j=item;j>=i;j--)

{

System.out.println("\*");

}

System.out.println();

}

}

}

**Output:-**

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

\*

Accepting input data from the keyboard:-

To accept data from the keyboard create a BuffererReader object.

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

Now, we can read the data coming from the keyboard using the read() and readLine() methods available in BufferedReader class.

Accepting a single character from keyboard:-

Char ch=(char)br.read();

Accepting a string from keyboard:-

String str=br.readLine();

Accepting an integer value from keyboard:-

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

parseInt method is a static method in integers class, so it can be called using class name as Integer.parseInt().

Accepting a float value from keyboard:-

float n=Float.parseFloat(br.readLine());

Accepting a double value from keyboard:-

double n=Double.parseDouble(br.readLine());

Ex:-

import java.io.\*;

class StudentDemo1

{

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

{

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

System.out.println("Enter roll number:");

int rno=Integer.parseInt(br.readLine());

System.out.println("Enter students name:");

String name=br.readLine();

//System.out.println("Enter gender(M/F):");

//Char gender=(gender)br.readLine();

System.out.println("ROLL.NO:-"+rno);

System.out.println("NAME:-"+name);

//System.out.println("GENDER:-"+gender);

}

}

**Output:-**

Enter roll number:

1

Enter students name:

ayyappa

Enter gender(M/F):

ROLL.NO:-1

NAME:-ayyappa

**Arrays:-**

A array represents a group of elements of same datatype.Array’s are classified into two types:-

1.single-dimensional array

2.two-dimensional array

1.Single – Dimensional array:-

A one dimensional array(or) single dimensional array represents roe or column of elements.

For example the marks obtained by a student in 5 different subjects can be represented by 1D array.

We can declare 1D array and directly store elements at the time of it’s declaration as

int marks[ ]={60,65,70,75,80};

we can create a 1D array by declaring the array first and then allocate memory for it by using new operator as:

int marks[ ]=new int[5];

marks[0]=60;

marks[1]=65;

marks[2]=70;

marks[3]=75;

marks[4]=80;

we can pass the values from the keyboard to the array by using a loop as given here:

for(int i=0;i<5;i++)

{

marks[i]=Integer.parseInt(br.readLine());

}

Ex:-

import java.io.\*;

class Array1

{

public static void main(String args[])

{

int marks[]={50,60,55,67,70};

System.out.println("Length="+marks.length);

for(int i=0;i<5;i++)

{

System.out.println(marks[i]+"\t");

}

}

}

**Output:-**

Length=5

50

60

55

67

70

Ex:-

import java.io.\*;

class ArrayDemo1

{

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

{

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

System.out.println("How many elements:");

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

int marks[]=new int[n];

System.out.println("Enter elements into the array:");

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

{

marks[i]=Integer.parseInt(br.readLine());

}

System.out.print("The entered elements in the array are:");

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

{

System.out.println(a[i]+"\t");

}

}

}

**Output:-**

How many elements:

2

Enter elements into the array:

5

5

The entered elements in the array are:

5

5

Two dimensional array:-

A two dimensional array represents several rows and columns of data.

To represent a two dimensional array,we should use two pairs of square braces[ ] [ ] after the array name.

For example, the marks obtained by a group of students in five subjects can be represented by a 2D array.

We can declare a two dimensional array and directly store elements at the time of it’s declaration as:-

int marks[ ] [ ]={{50,60,65,70,75},{62,65,70,70,81},{72,66,78,90,68}};

we can create a two dimensional array by declaring the array first and then we can allot memory for it by using new operator as:-

int marks[ ] [ ];

marks=new int[3][5]; -allot memory for storing 15 elements.

These two elements also can be written as:

int marks[ ] [ ]=new int[3][5];

Ex:-

import java.io.\*;

class Matrix

{

public static void main(String args[])

{

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

for(int i=0;i<2;i++)

{

System.out.println();

for(int j=0;j<3;j++)

{

System.out.println(x[i][j]+"\t");

}

}

}

}

**Output:-**

1

2

3

4

5

6

STRINGS:-

A string is a collection of characters. Strings are represented as string objects in java.

Creating strings;-

\*we can declare a string variable and directly store a string literal using assignment operator.

String str=”Hello”;

\*we can create string object using new operator with some data.

String s1=new String(“Java”);

\*we can create a string by using character array also.

Char arr[ ]={‘p’,’r’,’o’,’g’,’r’,’a’,’m’};

|  |  |
| --- | --- |
| METHOD | DESCRIPTION |
| String concat(String str) | Concatenates calling String with str |
| Int length | Returns length of a String |
| Char charAt(int index) | Returns the character at specified location(from 0). |
| int compareTo(String str) | Returns a negative value if calling string is less than str, a positive value if calling string is greater than str or 0 if strings are equal |
| Boolean equals(String str) | Returns true if calling string equals to str. |
| Boolean equals IgnoreCase(String str) | Same as above but ignores the case. |
| Boolean startsWith(String prefix) | Returns true if calling string starts with prefix. |
| Boolean endsWith(String suffix) | Returns true if calling string ends with suffix. |
| int indexOf(String str) | Returns first occurrence of str in string. |
| int lastIndexOf(String str) | Returns last occurrence of str in the string` |
| String replace(Char old char,char new char) | Returns a new string that is obtained by replacing all characters old char in string with new char. |
| String substring(int begin Index) | Return a new string consisting of all characters from begin index with the end of the string. |
| String substring(int begin Index, int end Index) | Returns a new string consisting of all characters from begin index with the end index. |
| String to lowercase() | Converts all characters into lowercase. |
| String to uppercase() | Converts all characters into lowercase. |
| String trim() | Eliminates all leading and training spaces. |

EX:-

import java.io.\*;

class StringDemo

{

public static void main(String args[])

{

String str1=new String("Hello");

String str2=new String("Hello");

if(str1==str2)

{

System.out.println("Equals");

}

else

{

System.out.println("Not equala");

}

}

}

**Output:-**

Not equala

**Equals method:-**

The equals method in the string class is a over ridden method. This over ridden method compares the value of both the strings.

If both the strings having same value it returns true otherwise it returns false.

Ex:-

import java.io.\*;

class StringDemo1

{

public static void main(String args[])

{

String str1=new String("Hello");

String str2=new String("Hello");

if(str1.equals(str2))

{

System.out.println("Equals");

}

else

{

System.out.println("Not equala");

}

}

}

**Output:-**

Equals

Ex:-

import java.io.\*;

class StringDemo5

{

public static void main(String args[])

{

String str1="He"+"llo";

String str2="Hel"+"lo";

if(str1==str2)

{

System.out.println("Equals");

}

else

{

System.out.println("Not Equals");

}

}

}

**Output:-**

Equals

Ex:-

import java.io.\*;

class StringDemo6

{

public static void main(String args[])

{

String s="llo";

String str1="He"+s;

String str2="He"+"lo";

if(str1==str2)

{

System.out.println("Equals");

}

else

{

System.out.println("Not Equals");

}

}

}

**Output:-**

Not Equals

Ex:-

class Concatination

{

public static void main(String args[])

{

String str1=new String("Hellol");

String str2=str1.concat("Matrix");

System.out.println(str1);

System.out.println(str2);

}

}

Ex:-

import java.lang.\*;

class StringDemo7

{

public static void main(String args[])

{

String str=new String("Hello NeoApplications");

// get substring from 3rd index until brfore 8th index

String Sub1=str.substring(3,8);

//from 7th index until end of the string

String Sub2=str.substring(7);

System.out.println(Sub1);

System.out.println(Sub2);

}

}

**Output:-**

Hellol

HellolMatrix

Ex:-

class StringOperations

{

public static void main(String args[])

{

String str=new String("Hello Matrix");

char ch=str.charAt(6);

System.out.println(ch);

boolean b=str.endsWith("Matrix");

System.out.println(b);

System.out.println(str.indexOf('o'));

System.out.println(str.lastIndexOf('t'));

}

}

**Output:-**

M

true

4

8

Ex:-

import java.io.\*;

class Strmethods

{

public static void main(String args[])

{

String s1="Bhagirath";

System.out.println("s1 replace a with z="+s1.replace('a','z'));

String s2="ViewSonic";

System.out.println("s1 and s2 trim="+s1.trim()+s2.trim());

System.out.println("s1 and s2 concat="+s1.concat(s2));

System.out.println("s1 substring(n)="+s1.substring(5));

System.out.println("s1 substring(n,m)="+s1.substring(5,8));

int i=100;

//converts the parameter to string

System.out.println("s1.valueOf(variable)="+(s1.valueOf(i)).length());

System.out.println("Start with"+s1.startsWith("p"));

System.out.println("Ends with"+s1.endsWith("h"));

}

}

**Output:-**

s1 replace a with z=Bhzgirzth

s1 and s2 trim=BhagirathViewSonic

s1 and s2 concat=BhagirathViewSonic

s1 substring(n)=rath

s1 substring(n,m)=rat

s1.valueOf(variable)=3

Start withfalse

Ends withtrue

NOTE:-

We can decide objects broadly as mutable and immutable objects. Mutable objects are those objects whose content can be modified. Immutable objects are those objects once created can’t be modified.

**String Buffer:-**

String buffer objects are mutable, so they can be modified. The methods that directly manipulate data of the object are available in string buffer class.

**Creating String Buffer:-**

\*we can create a String Buffer object by using new operator and pass the string to the object as:-

StringBuffer sb=new Stringbuffer(“Hello”);

\*we can create a StringBuffer object by first allotting memory to the string buffer object using new operator and later storing the string into it as:-

StringBuffer sb=new StringBuffer(30);

\*In general a stringbuffer object will be created with a default capacity of 16 characters. Here, stringbuffer object is created on empty object with capacity for string 30 characters. Even if we declare the capacity as 30, it is possible to store more than 30 characters into the stringbuffer.

**String Buffer class methods:-**

**String Buffer append(x):-**

X may be int,float, double, char, string, or StringBuffer*.* It will be append to calling StringBuffer.

Eg:-StringBuffer sb=new StringBuffer(“uni”);

Sb.append(“versity”);

Proceeding two statements produces: university is added to the end of “uni”.

**stringBuffer insert(inti,x);**

x may be Boolean, byte, int, float, double, char, string, stringbuffer. It will be inserted into the string buffer at the position of represented by i.eg:-StringBuffer sb=new StringBuffer (“Intelligent person”);

sb.insert(11,”YOUNG”);

System.out.println(sb);

o/p:-Intelligent young person.

**String buffer delete:-**

\*String Buffer Delete(int start, int end):

Removes characters from start to end.

Eg:- String Buffer sb=new String Buffer(“university”);

Sb.delete(0,3) deletes the characters from the beginning to 2nd character(“uni”) and the resultant string will be “versity”.

**String Buffer reverse:-**

\*string buffer reverse(): reverses character sequence in the string buffer.

🡪if the string buffer contains”abc”, it becomes “cba”.

**Index Of:-**

\*int indexOf(String str):- This reurns the first occurance of subsatring’str’ in the StingBuffer object.

Ex:-StringBuffer sb=new StringBuffer(“This is a book”);

int n=sb.indexOf(“is”);

System.out.println(n);

**lastIndexOf:-**

\*int lastIndexOf(String str): This returns the last occurrence of substring ‘str’ in the StringBuffer object.

Ex:-

StringBuffer sb=new StringBuffer(“This is a book”);

Int n= sb.lastIndexOf(“is”);

System.out.println(n);

**String to String:-**

\*String to String(); converts StringBuffer into a string.

**Length:-**

\*int length: returns length of the StringBuffer.

**StringBuffer replace:\_**

\*stringbuffer replace(int i, int j, String str): this replaces the character from I to j-1 by the string ’str’ in the StringBuffer object.

Ex:-

StringBuffer sb=new StringBuffer(“High cost”);

Sb.replace(0,4,”low”);

We are replacing first 4 characters of sb by the 3 characters “low”.

Now sb contains “low cost”.

**Substring:-**

\*String SubString(int i): This retrives a SubString from the StringBuffer object starting i th position till the end.

Ex:-

StringBuffer sb=new StringBuffer(“New Delhi”);

String s=sb. substring(4);

System.out.println(s);

\*String SubString(int i ,j): This retrives a substring from the StringBuffer object starting from the i’th position to the j-1 th position.

Ex:-

StringBuffer sb=new StringBuffer(“New Delhi”);

String s=sb.substring(0,3);

System.out.println(s);

Ex:-

import java.io.\*;

class Mutable

{

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

{

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

System.out.print("Enter sur name:");

String sur=br.readLine();

System.out.print("Enter mid name:");

String mid=br.readLine();

System.out.print("Enter last name:");

String last=br.readLine();

//create string buffer object

StringBuffer Sb=new StringBuffer();

//append sur to Sb

Sb.append(sur);

//append last name to Sb

Sb.append(last);

//display the name till now

System.out.println("Name="+Sb);

//insert mid after sur name in Sb

int n=sur.length(); //n represents number of characters in surname

Sb.insert(n,mid);

//display full name

System.out.println("full name="+Sb);

//reverse and display the name

System.out.println("In reverse="+Sb.reverse());

}

}

**Output:-**

Enter sur name:konathalapalli

Enter mid name:venkata

Enter last name:surya

Name=konathalapallisurya

full name=konathalapallivenkatasurya

In reverse=ayrusataknevillapalahtanok

Ex:-

import java.io.\*;

class Strbuf1

{

public static void main(String args[])

{

StringBuffer s1=new StringBuffer();

StringBuffer s2=new StringBuffer("Bhagirath");

StringBuffer s3=new StringBuffer(s2);

StringBuffer s4=new StringBuffer(100);

System.out.println("s1 is="+s1);

System.out.println("s2 is="+s2);

System.out.println("s3 is="+s3);

System.out.println("s1 length is="+s1.length());

System.out.println("s2 length is="+s2.length());

System.out.println("s3 length is="+s3.length());

System.out.println("s4 length is="+s4.length());

System.out.println("s1 capacity is="+s1.capacity());

System.out.println("s2 capacity is="+s2.capacity());

System.out.println("s3 capacity is="+s3.capacity());

System.out.println("s4 capacity is="+s4.capacity());

}

}

**Output:-**

s1 is=

s2 is=Bhagirath

s3 is=Bhagirath

s1 length is=0

s2 length is=9

s3 length is=9

s4 length is=0

s1 capacity is=16

s2 capacity is=25

s3 capacity is=25

s4 capacity is=100

Ex:-

class Student

{

int rollno;

String name;

void display()

{

System.out.println("Student roll number is="+rollno);

System.out.println("Student name="+name);

}

}

class StudentDemo0

{

public static void main(String args[])

{

Student s=new Student();

s.display();

}

}

**Output:-**

Student roll number is=0

Student name=null

**Note:-** when the programmer does not initialize the instance variable, java compiler will write code and initialize the variables with default values.

|  |  |
| --- | --- |
| **Data type** | **Default values** |
| Int | 0 |
| float | 0.0 |
| double | 0.0 |
| char | Space |
| String | Null |
| Class | null |
| boolean | false |

**Initializing Instance variable:-**

**Type-1:**

We can initializing instance variable directly in the class using assignment operator. In this type every object is initialized with the same data.

Ex:-

class Student

{

int rollno=101;

String name="Ayyappa";

Void display()

{

System.out.println("Student roll number is:"+rollno);

System.out.println("Student name is:"+name);

}

}

class Student1

{

public static void main(String args[])

{

Student s=new Student();

s.display();

Student s1=new Student();

s1.display();

}

}

**Output:-**

Student roll number is=101

Stuydent name=Ayyappa

Student roll number is=101

Stuydent name=Ayyappa

**Type-2:-**

We can initialize one class instance variable in another class using the reference variable.

Ex:-

class Student

{

int rollno;

String name;

void display()

{

System.out.println("Student roll number is="+rollno);

System.out.println("Stuydent name="+name);

}

}

class StudentDemo3

{

public static void main(String args[])

{

Student s=new Student();

s.rollno=101;

s.name="ramesh";

s.display();

Student s1=new Student();

s1.rollno=102;

s1.name="suresh";

s1.display();

}

}

**Output:-**

Student roll number is=101

Stuydent name=ramesh

Student roll number is=102

Stuydent name=suresh

**Access specifiers:-**

An access specifier is a keyword that represents how to avess a member of a class. There are four access specifiers in java.

1.**Private**:-

Private members of a class are not available outside the class.

2.**Public**:-

Public members of a class are available anywhere outside the class.

3.**Protected**:-

Protected members are available outside the class.

4.**Default**:-

If no access specifier is used then default specifier is used by java compiler. Default members are available outside the class.

**Type-3:-**

We can initialize instance variable using a constructor.

**Constructor:-**

🡪A constructor is similar to a method that initializes the instance variable of a class.

🡪A constructor name and class must be same.

🡪A constructor may have or may not have parameters. Parameters are local variable to receive data.

🡪A constructor without any parameters is called default constructor.

eg:-

class Student

{

int roll no;

String name;

Student()

{

roll no=1;

name="Ayyappa";

}

}

🡪A constructor with one or more parameters uis called parameterized constructor.

eg:-

class Student{

int rollno;

String namr;

Student(int r,String n)

{

rollno=r;

namer=n;

}

}

🡪A constructor does not return any value, not even void.

🡪A constructor is called and executed at the same time of creating an object.

🡪A constructor is called anly once per object.

🡪Default constrictor is used to initialize every object with same data. Where as parameterized constructor is used to initialize each object with different data.

🡪If no constructor is written in a class then java compiler will provide default values.

Ex:-

class Student

{

int rollno;

String name;

Student()

{

rollno=101;

name="Ayyappa";

}

void display()

{

System.out.println("Student roll number is:"+rollno);

System.out.println("Student name is:"+name);

}

}

class StudentDemo4

{

public static void main(String args[])

{

Student s1=new Student();

System.out.println("s1 object contains:");

s1.display();

Student s2=new Student();

System.out.println("s2 object contains:");

s2.display();

}

}

**Output:-**

s1 object contains:

Student roll number is:101

Student name is:Ayyappa

s2 object contains:

Student roll number is:101

Student name is:Ayyappa

Ex:-

class Student

{

int rollno;

String name;

Student(int r,String n)

{

rollno=r;

name=n;

}

void display()

{

System.out.println("Student roll number is:"+rollno);

System.out.println("Student name is:"+name);

}

}

class StudentDemo5

{

public static void main(String args[])

{

Student s1=new Student(101,"ramesh");

System.out.println("s1 object contains:");

s1.display();

Student s2=new Student(102,"suresh");

System.out.println("s2 object contains:");

s2.display();

}

}

**Output:-**

s1 object contains:

Student roll number is:101

Student name is:ramesh

s2 object contains:

Student roll number is:102

Student name is:suresh

**The Keyword “This”:-**

There will be situations where a method wants to refer to the object which invoked it. To perform this we use ‘this’ keyword. There are no restrictions to use this keyword we can use this inside any method for referring the current object.

It refers to:-

1.present class instance variable.

2.present class method.

3.present class constructor.

Ex:-

class Person

{

String name;

Person()

{

this("Ayyappa");

this.display();

}

Person(String name)

{

this.name=name;

}

void display()

{

System.out.println("Person name is="+name);

}

}

class ThisDemo

{

public static void main(String args[])

{

Person p=new Person();

}

}

**Output:-**

Person name is=Ayyappa

**Instance method:-**

\*Methods which act upon instance variables of a class are called instance methods.

\*To call instance methods use objectname.methodname.

\*Instance variable is a variable whose separate copy is available in every object.

\*Any modifications to instance variables in one object will not affect the instance variable of the other objects. This variables are crated on heap.

Ex:-

class Sample

{

int x=10;

void display()

{

x++;

System.out.println("x value is:"+x);

}

}

class SDemo

{

public static void main(String args[])

{

Sample s1=new Sample();

System.out.println("s1 object contains:");

s1.display();

Sample s2=new Object();

System.out.println("s2 Object contains:");

s2.display();

}

}

**Output:-**

s1 object contains:

x value is:11

s2 Object contains:

x value is:11

**Static method:-**

\*Static methods can read and out upon static variables.

\*Static methods cannot read and act upon instance variables.

\*static variables is a variable whose single copy is shared by all the objects.

\*Static methods are declared using keyword “static”.

\*Static methods can be called using objectname.Methodname(or) classname.Methodname.

\*From any object, if static variable is modified it affects all the objects. Static variables are stored on the method area.

Ex:-

class Sample

{

static int x=10;

static void display()

{

x++;

System.out.println("xvalue is:"+x);

}

}

class SDemo1

{

public static void main(String args[])

{

Sample s1=new Sample();

System.out.println("calling objectname.methodname:");

s1.display();

System.out.println("calling classnaME.METHODNAME:");

Sample.display();

}

}

**Output:-**

calling objectname.methodname:

xvalue is:11

calling classnaME.METHODNAME:

xvalue is:12

**Inner class:-**

A class within another class is called inner class. Where the programmer wants to restrict the access of entire code of a class, creates an inner class as a private class. The way to access the inner class is through it’s outer class only.

\*Inner class is a safety mechanism.

\*Inner class is hidden in outer class from other classes.

\*Only inner class can be private.

\*An object to inner class can be created only it’s outer class.

\*An object to inner class cannot be created in any other class.

\*Outer class objects and inner class objects are created in separate memory locations.

\*Outer class members are available to inner class objects.

\*Inner class members are referenced as this.member;

\*Outer class members are referenced as outerclass.this.member;

Ex:-

class Bank

{

private double bal,rate;

Bank(double b, double r)

{

bal=b;

rate=r;

}

void display()

{

Interest in=new Interest();

in.calculateInterest();

System.out.println("New Balance:"+bal);

}

private class Interest

{

void calculateInterest()

{

System.out.println("Balance="+bal);

double interest=bal\*rate/100;

System.out.println("Interest="+interest);

bal+=interest;

}

}

}

class InnerDemo

{

public static void main(String args[])

{

Bank account=new Bank(20000,5);

account.display();

}

}

**Output:-**

Balance=20000.0

Interest=1000.0

New Balance:21000.0

**Inheritance :-**

**1.super class or parent class:-**

\*The class that is having common information is called parent class.

\*Parent is also called as base class or super class.

**2.sub class or child class:-**

\*The class that is having specific information and reusing common information is called child class.

\*Child class is also called as derived class or sub class.

**Meaning of Inheritance:-**

Creating a child object by reusing parent class variable and methods. Using child object reference we can access parent class variables, parent class methods, child class variables and child class methods.

Ex:-

public class Parent

{

public int p=1;

public void display()

{

System.out.println("Parent value="+p);

}

}

Ex:-

public class Child extends Parent

{

public int c=2;

public void Show()

{

System.out.println("Child value="+c);

}

}

Ex:-

public class Test

{

public static void main(String args[])

{

Child obj=new Child();

System.out.println(obj.p);

obj.display();

System.out.println(obj.c);

obj.Show();

}

}

**Output:-**

1

Parent value=1

2

Child value=2

**Types of Inheritance:-**

1.**single inheritance:-**

When a class inherits another class as single inheritance.

For example Dog class inherits the animal class so there is the single inheritance.

**Syntax:-**

class A

{

---------

}

class B extends A

{

---------

}

Ex:-

class Animal

{

void eat()

{

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

}

}

class Dog extends Animal

{

void bark()

{

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

}

}

class Test1

{

public static void main(String args[])

{

Dog d=new Dog();

d.bark();

d.eat();

}

}

**Output:-**

barking....

eating....

**Multilevel Inheritance:-**

When there is a chain of inheritance, it is known as multilevel inheritance.

For example BabyDog class inherits the dog class. Which again inherits the animal class, so there is a multilevel inheritance.

**Syntax:-**

class A

{

---------

}

class B extends A

{

---------

}

class C extends B

{

---------

}

Ex:-

class Animal

{

void eat()

{

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

}

}

class Dog extends Animal

{

void bark()

{

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

}

}

class BabyDog extends Dog

{

void weep()

{

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

}

}

class Test2

{

public static void main(String args[])

{

BabyDog b=new BabyDog();

b.bark();

b.eat();

b.weep();

}

}

**Output:-**

barking....

eating....

weeping.....

**Hierarchical Inheritance:-**

When two or more classes inherits a single class it is known as hierarchical inheritance.

For example Dog and Cat classes inherits the animal class, so there is hierarchical inheritance.

**Syntax:-**

class A

{

----------

}

class B extends A

{

---------

]

class C extends A

{

---------

}

Ex:-

class Animal

{

void eat()

{

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

}

}

class Dog extends Animal

{

void bark()

{

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

}

}

class Cat extends Animal

{

void meow()

{

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

}

}

class Test3

{

public static void main(String args[])

{

Cat c=new Cat();

c.meow();

c.eat();

Dog d=new Dog();

d.bark();

d.eat();

}

}

**Output:-**

meowing.....

eating....

barking....

eating....

**Hybrid inheritance:-**

It is the combination of two or more types of inheritance.

**Syntax:-**

class A

{

---------

}

class B extends A

{

---------

}

class C extends A

{

---------

}

class D extends B,C

{

----------

}

Ex:-

class Animal

{

void eat()

{

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

}

}

class Dog extends Animal

{

void bark()

{

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

}

}

class Cat extends Animal

{

void meow()

{

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

}

}

class BabyDog extends Dog

{

void play()

{

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

}

}

class Test4

{

public static void main(String args[])

{

BabyDog b=new BabyDog();

b.eat();

b.bark();

b.play();

}

}

**Output:-**

eating.....

barking.....

playing.....

**Polymorphism:-**

\*Polymorphism came from two Greek words ‘poly’ means many and ‘morphism’ means forms.

\*If the same method has ability to take more than one form to perform several tasks then it is called polymorphism.

\*It is of two types:-1) Dynamic polymorphism

2) Static polymorphism.

1.**Dynamic polymorphism:-**

The polymorphism exhibited at the run time is called dynamic polymorphism. In the dynamic polymorphism a method call is linked with method body at the time of execution by JVM. Java compiler does not know which method is called at the time of compilation. This is also known as dynamic binding run time polymorphism.

Method overloading and Method overriding are examples of dynamic polymorphism in java.

**Method overloading:-**

Writing two or more methods with the same name but with a difference in the method signatures is called method overloading . Method signature represents the method name along with the method parameters. In the method overlading JVM understands which method is called depending upon the difference in the method signature. The difference may be due to the following:

\*There is a difference in the no.of parameters

Void add(int a, int b);

Void add(int a , int b, int c);

\*There is a difference in the data types of parameters

Void add(int a, intb);

Void add(double a, double b);

\*There is a difference in the sequence of parameters

Void swap(int a,char b)

Void swap(char a, int b)

Ex:-

class Sample

{

void add(int a, int b)

{

System.out.println("Sum of two numbers="+(a+b));

}

void add(int a, int b, int c)

{

System.out.println("Sum of three numbers="+(a+b+c));

}

}

class OverLoad

{

public static void main(String args[])

{

Sample s=new Sample();

s.add(20,25);

s.add(20,25,30);

}

}

**Output:-**

Sum of two numbers=45

Sum of three numbers=75

**Method overriding:-**

Writng teo or more methods in super and sub class with the same name and same signatures is called method overriding. In method overring JVM executes a method depending on the type on the of the object.

Ex:-

import java.io.\*;

class Animal

{

void move()

{

System.out.println("Animals can move");

}

}

class Dog extends Animal

{

void move()

{

System.out.println("Dog can walk and run");

}

}

public class OverRide1

{

public static void main(String args[])

{

Animal a=new Animal();

a.move();

Animal b=new Dog();

b.move();

}

}

**Output:-**

Animals can move

Dog can walk and run

**Static polymorphism:-**

The polymorphism exhibited at compile time is called static polymorphism. Here the compiler knows which method is called at the compilation. This is also called compile time polymorphism or static binding.

Ex:-

import java.io.\*;

class Animal

{

static void move()

{

System.out.println("Animals can move");

}

}

class Dog extends Animal

{

static void move()

{

System.out.println("Dog can walk and run");

}

}

class StaticPoly

{

public static void main(String args[])

{

Animal.move();

Dog.move();

}

}

**Output;-**

Animals can move

Dog can walk and run

**Abstract class:-**

\*A method with method body is called concrete methods. In general any class will have all concrete methods. A method without method body is called abstract method.

\*A class thart contains abstract method is called abstract class.

\*It is possible to implements the abstract methods differently in the sub classes of an abstract class.

\*the abstract methods and abstract class should be declared using the keyword abstract.

\*we cannot create objects to abstracts class because it is having incomplete code. Whenever an abstract class is created, sub class should be created to it and the abstract methods should be implemented in the sub classes, then we can create objects to the subclasses.

Ex:-

import java.io.\*;

abstract class Figure

{

double dim1;

double dim2;

Figure(double a,double b)

{

dim1=a;

dim2=b;

}

abstract double area();

}

class Rectangle extends Figure

{

Rectangle(double a, double b)

{

super(a,b);

}

double area()

{

System.out.println("Inside area of rectangle");

return dim1\*dim2;

}

}

class Triangle extends Figure

{

Triangle(double a,double b)

{

super(a,b);

}

double area()

{

System.out.println("Inside area of Triangle");

return dim1\*dim2/2;

}

}

class AbstractAreas

{

public static void main(String args[])

{

//Figure f=new Figure(10,10); -illegal now

Rectangle r=new Rectangle(9,5);

Triangle t=new Triangle(10,8);

System.out.println("Area is"+r.area());

System.out.println("Area is"+t.area());

}

}

**Output:-**

Inside area of rectangle

Area is45.0

Inside area of Triangle

Area is40.0

**Interface:-**

\*A programmer uses an abstract class when there are some common features shared by all the objects.

\*A programmer writes an interface when all the features have different implementations for different objects. All the methods in an interface are abstract methods.

\*An interface contains zero or more abstract methods.

\*All the methods of a interface are public, abstract by default.

\*All the methods of the interface should br implemented, then it’s implementation classes.

\*If any one of the methods is not implemented, then that implementation class should be declared as abstract.

\*We cannot create an object to interface.

\*We can create a reference implement another interface.

\*An interface can extend another interface.

\*A class can implement multiple interface.

Ex:-

interface Shape

{

void area();

void volume();

double pi=3.14;

}

class Circle implements Shape

{

double r;

Circle(double radius)

{

r=radius;

}

public void area()

{

System.out.println("Area of circle is:"+pi\*r\*r);

}

public void volume()

{

System.out.println("volume of circle is:"+2\*pi\*r);

}

}

class Rectangle implements Shape

{

double l, b;

Rectangle(double length, double breadth)

{

l=length;

b=breadth;

}

public void area()

{

System.out.println("Area of rectangle is:"+l\*b);

}

public void volume()

{

System.out.println("volume of rectangle is:"+2\*(l+b));

}

}

class InterfaceDemo

{

public static void main(String args[])

{

Circle ob1=new Circle(10.2);

ob1.area();

ob1.volume();

Rectangle ob2=new Rectangle(12.6,23.55);

ob2.area();

ob2.volume();

}

}

**Output:-**

Area of circle is:326.68559999999997

volume of circle is:64.056

Area of rectangle is:296.73

volume of rectangle is:72.3

Ex:-

interface FirstInterface

{

int id=20;

void printMessage();

}

interface SecondInterface

{

int count=10;

void printDetail();

}

class MultipleInterface implements FirstInterface,SecondInterface

{

public void printMessage()

{

System.out.println("This is print merssage method");

}

public void printDetail()

{

System.out.println("This is print detail method");

}

public static void main(String args[])

{

MultipleInterface obj=new MultipleInterface();

obj.printMessage();

obj.printDetail();

System.out.println("id="+obj.id);

System.out.println("count="+obj.count);

}

}

**Output;-**

This is print merssage method

This is print detail method

id=20

count=10

**Packages:-**

It is necessary in software development to create several classes and interfaces. After creating these classes and interfaces it is better if they are divided into some groups depending on their relationships. Thus the classes and interfaces which handle similar or same tasks are put into same directory. The directory or folder is called as a package.

A package is a container of classes and interfaces. A package represents a directory that contain related group of classes and interfaces.

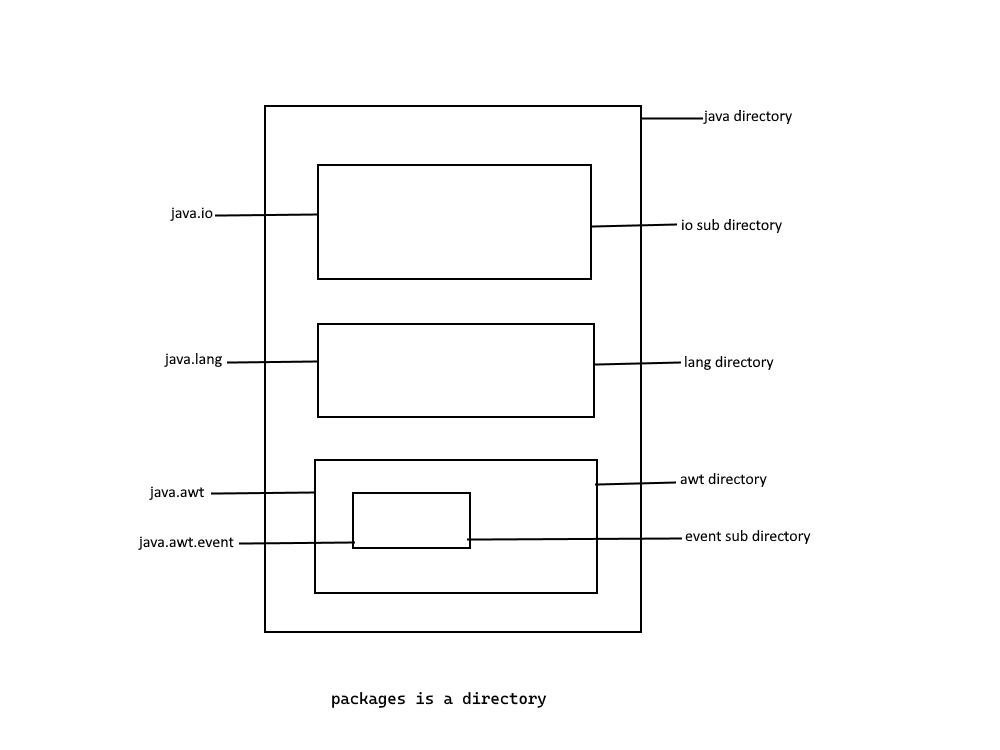
For example when we write statement like

import java.io.\*;

Here we are importing classes of java.io package.

Here java is a directory name and io is the another subdirectory within it. The ‘\*’ represents all the cl;asses and interfaces of that io sub directory.

We can create our own packages called user defined packages. User defined packages can also be imported into other classes and used exactly in the same way as the built-in-packages. Packages provides reusability.



**General form of creating a package:**-

Package package name;

Eg:-package matrix;

\*The first statement in the program must be package statement while creating a package.

\*while creating a package except instance variables declare all the members and the class itself as public then only the public members are available outside the package to other programs.

Ex:-

package Matrix;

public class Addition

{

private double d1,d2;

public Addition(double a, double b)

{

d1=a;

d2=b;

}

public void sum()

{

System.out.println("Sum of two given numbers is:"+(d1+d2));

}

}

Javac -d . Addition.java

The -d option tells the java compiler to create a separate directory and place the . class file in that directory(packages). The . after -d indicates that the package should be created in the current directory. So the package matrix with addition class is ready.

Now create an object for addition class, we can write as

Matrix.Addition obj=new Matrix.Addition(10,20);

Ex:-

import Matrix.Addition;

class Use

{

public static void main(String args[])

{

Addition obj1=new Addition(10,20);

obj1.sum();

}

}

**Output:-**

Sum of two given numbers is:30.0

**Creating a subpackage in a package;-**

We can create subpackage in a package in a formate

Package packagename.subpackagename;

In the following program we are creating tech package inside dream package by writing the statement as

Package Dream.tech;

Ex:-

package dream.tech;

public class Sample

{

public void Show()

{

System.out.println("Welcome to dreamtech");

}

}

//javac -d . Sample.java

Ex:-

import dream.tech.Sample;

class Use1

{

public static void main(String args[])

{

Sample s=new Sample();

s.Show();

}

}

**Output:-**

Welcome to dreamtech

**Exception handling:-**

\*A abnormal event in a program is called exception.

\*Exception may occur at compile time or at run time,

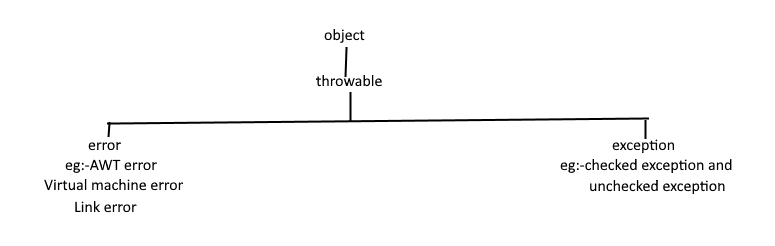
\*Exception which occur at compile time are called checked exceptions.

Ex:- class not found exception , no such method exception, no such a field exception etc..

\*Exception which occur at run time are called unchecked exception.

Ex:- arrayindex out of Bounds exception, Arithmetic exception, number formate exception.

\*Exceptions are represented as class in java.



\*An exception can be handled by the programmer where as an error cannot be handled by the programmer. When there is an exception the programmer should do the following tasks:

***Exceptional handling mechanisms:-***

**1.Try block:-**

If the programmer suspects any exception in program statements he should write them inside Try block.

**Syntax:-**

try

{

statement 1;

statement 2;

-----------

-----------

statement n;

}

**2.Catch block:-**

When there is an exception in try block JVM will not terminate the program abnormally. JVM stores exception details in an exception stack and then JVM jumps into each block. The programmer should display exception details and any message to the user in Catch block.

**Syntax:-**

catch(Exceptionclass obj)

{

statements;

}

**3.Finally block:-**

Programmer should close all the files and database by writing them inside finally block. Finally block is executed whether there is exception or not.

**Syntax:-**

finally

{

statements;

}

**Important points:-**

\*Even through multiple exceptions are found in the program, only one exception is raised at a time.

\*We can handle multiple exceptions by writing multiple catch blocks.

\*a single try block can be followed by several catch blocks.

\*Catch block does not always exit without a try, but a try block exit without a catch block.

\*Finally block is executed whether there is exception or not.

Ex:-

import java.lang.\*;

class ExceptionalTest

{

public static void main(String args[])

{

String str=new String("Hello");

System.out.println(str.charAt(0));

System.out.println(str.charAt(1));

System.out.println(str.charAt(2));

System.out.println(str.charAt(3));

System.out.println(str.charAt(4));

System.out.println(str.charAt(5));

System.out.println("End of the program");

}

}

**Output:-**

H

e

l

l

o

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 5

at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)

at java.base/java.lang.String.charAt(String.java:1512)

at ExceptionalTest.main(ExceptionalTest.java:12)

Ex:-

import java.lang.\*;

class ExceptionalTest1

{

public static void main(String args[])

{

String str=new String("Hello");

try

{

System.out.println(str.charAt(0));

System.out.println(str.charAt(1));

System.out.println(str.charAt(2));

System.out.println(str.charAt(3));

System.out.println(str.charAt(4));

System.out.println(str.charAt(5));

System.out.println("End of the program");

}

catch(StringIndexOutOfBoundsException excep)

{

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

}

System.out.println("End of the program");

}

}

**Output:-**

H

e

l

l

o

String index out of range: 5

End of the program

Ex:-

import java.lang.\*;

class ExceptionalTest2

{

public static void main(String args[])

{

String str=null;

try

{

System.out.println(str.charAt(0));

System.out.println(str.charAt(1));

System.out.println(str.charAt(2));

System.out.println(str.charAt(3));

System.out.println(str.charAt(4));

System.out.println(str.charAt(5));

System.out.println("End of the program");

}

catch(StringIndexOutOfBoundsException excep)

{

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

}

finally

{

System.out.println("End of the program");

}

}

}

**Output:-**

End of the program

Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.charAt(int)" because "<local1>" is null

at ExceptionalTest2.main(ExceptionalTest2.java:9)

Ex:-

class ExceptionalExample

{

public static void main(String args[])

{

try

{

System.out.println("Open files");

//int n=args.length;

int n=10;

System.out.println("n="+n);

int a=45/n;

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

int b[]={10,19,12,13};

b[50]=100;

}

catch(ArithmeticException ae)

{

System.out.println(ae);

System.out.println("plz type data while executing the program");

}

catch(ArrayIndexOutOfBoundsException aie)

{

System.out.println(aie);

System.out.println("Array index is not with in the Range");

}

finally

{

System.out.println("Close files");

}

}

}

**Output:-**

Open files

n=10

a=4

java.lang.ArrayIndexOutOfBoundsException: Index 50 out of bounds for length 4

Array index is not with in the Range

Close files

**Threads:-**

**Multithreading programming:-**

A thread represents execution of statements. The way the statements are executed is of two types:

1.Single tasking

2.Multi tasking

**1.Single tasking:-**

Executing only one task at a time is called single tasking. In single tasking the microprocessor will be settle idle for most of the time. This means microprocessor time is wasted.

**2.Multi tasking:-**

Executing more than one task at a time is called multitasking.

Multitasking of two types:-

**1.process based multitasking:-**

Executing several programs simultaneously is called process based multitasking.

**2.Thread based multitasking;-**

Executing different parts of the same program simultaneously with the help of thread is called thread based multitasking.

Advantages of multitasking is utilizing the process time in an optimum way.

Java provides built in support for multithreading programming. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called thread.

Thread is a smallest unit of code. Thread is also defined as a subprocess. A thread sometimes called an execution context (or) a light weight process.

**Uses of thread:-**

\*Threads are used in designing service side programs to handle multiple clients at a time.

\*threads are used in games and animations.

**Creating a thread and running it:-**

1.create a class that extends thread class or implements runnable interface.

Both thread class and runnable interface are available in java.lang package

Class Myclass extends Thread

(or)

Class Myclass implements Runnable.

2.Now in this class, wreite a run() method as:

Public void run()

{

Statements;

}

By default this run method is recognized and executed by a thread.

3.create an object to Myclass, so that the run() method is available for execution.

Myclass obj=new Myclass();

4.Now create a thread and attach it to the object.

Thread t= new Thread(obj);

5.Run the thread, For this purpose, we should use start() method of Thread class.

t.start():

Now , the thread will start execution on the object of Myclass. In that object run() method is found(), hence it will execute the statements inside that run() method.

Ex:-

import java.lang.\*;

class Current

{

public static void main(String args[])

{

System.out.println("This is the first statement");

Thread t=Thread.currentThread();

System.out.println("current thread="+t);

System.out.println("its name:"+t.getName());

System.out.println("its priority:"+t.getPriority());

}

}

**Output:-**

This is the first statement

current thread=Thread[main,5,main]

its name:main

its priority:5

Ex:-

import java.lang.\*;

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=10000;i++)

{

System.out.println(i);

}

}

}

class TDemo

{

public static void main (String args[])

{

MyThread obj=new MyThread();

Thread t=new Thread(obj);

t.start();

}

}

**Output:-**

1

2

3

4

5

6

7

8

9

10

// prints infinite values

**Terminating a thread:-**

1.Create a Boolean type variable and initialize it to table.

boolean stop = false;

2.we want to terminate the thread when the user press<enter> key. So when user presses that button, make the boolean type variable as true.

Stop=true;

3.Check the variable in run() method when it is true, make the thread return from the run() method,

public void run()

{

if(stop==true)

return;

}

Ex:-

import java.lang.\*;

import java.io.\*;

class MyTd extends Thread

{

boolean stop=false;

public void run()

{

for(int i=1;i<=100000;i++)

{

System.out.println(i);

if(stop)return;

}

}

}

class Terminate

{

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

{

MyTd obj=new MyTd();

Thread t=new Thread(obj);

t.start();

//stop the thread when entry key pressed

System.in.read();

obj.stop=true;

}

}

**Output:-**

1

2

3

4

5

6

7

8

9

10

// stops when you press the enter button

**Multi tasking using Threads:-**

In multitasking several tasks are executed at a time. For this purpose we need more than one thread. For example to perform 2 tasks we can take 2 threads and attach them to the 2 tasks. Then those tasks are simultaneously are executed by the two threads. Using more than one thread is called multithreading.

Ex:-

import java.lang.\*;

class Theatre extends Thread

{

String str;

Theatre(String str)

{

this.str=str;

}

public void run()

{

for(int i=1;i<=10;i++)

{

System.out.println(str+":"+i);

try

{

Thread.sleep(2000);

}

catch(InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

class TDemo1

{

public static void main(String args[])

{

Theatre obj1=new Theatre("cut tickets");

Theatre obj2=new Theatre("show chair");

Thread t1=new Thread(obj1);

Thread t2=new Thread(obj2);

t1.start();

t2.start();

}

}

**Outpur:-**

show chair:1

cut tickets:1

show chair:2

cut tickets:2

cut tickets:3

show chair:3

cut tickets:4

show chair:4

cut tickets:5

show chair:5

cut tickets:6

show chair:6

cut tickets:7

show chair:7

cut tickets:8

show chair:8

cut tickets:9

show chair:9

cut tickets:10

show chair:10

// prints with delay time of 2 seconds

**Collection of items:-**

**Array list:-**

The array list class is a resizable array. Which can be found in java.util package.

The difference between a built in array and array list in java is that the size of an array cannot be modified (if you want to add or remove elements to an array you have to create a new one).

While elements can be added and removed from an array list whwnever you want.

**Syntax:-**

ArrayList<String>al=new ArrayList<String>();

Create an array list object called cars that will store things:

import java.util.Arraylist;

ArrayList<String> cars=new ArrayList<String>();

**Add items:-**

The array list class has many useful methods. To add elements to the array list, use the add() method.

**Accessing an item:-**

Use the get() method and refer to the index number to access an element in the array list.

**Change an item:-**

To modify an element use the set() method and refer to index number.

**Remove an item:-**

To remove an element use the remove() method and refer to index number.

**ArrayList size:-**

To find out how many elements an array list have use the size() method.

**Loop through an array list:-**

Loop through the elements of an array list with a for loop and use the size() method to specify how many items the loop should run.

Ex:-

import java.util.ArrayList;

public class ArrayListEx

{

public static void main(String args[])

{

ArrayList<String>cars=new ArrayList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("Mazda");

System.out.println(cars);

System.out.println(cars.get(0));

cars.set(0,"Benz");

System.out.println("New ArrayList:"+cars);

cars.remove(0);

System.out.println(cars);

System.out.println(cars.size());

for(int i=0;i<cars.size();i++)

{

System.out.println(cars.get(i));

}

}

}

**Output;-**

[Volvo, BMW, Ford, Mazda]

Volvo

New ArrayList:[Benz, BMW, Ford, Mazda]

[BMW, Ford, Mazda]

3

BMW

Ford

Mazda

**Linked list:-**

The linked list class is a collection which contains many objects of the same type, just like the array list.

The linked list class can be used in the same way as the array list class it supports all the array list methods but they are built very differently.

**Linked list methods:-**

|  |  |
| --- | --- |
| Method | Description |
| addFirst() | Adds an item to the beginning of the list |
| addLast() | Adds an item to the end of the list |
| removeFirst() | Remove an item from the beginning of the list |
| removeLast() | Remove an item from the end of the list |
| getFirst() | Get the item at the beginning of the list |
| getLast() | Get the item at the end of the list |

Ex:-

import java.util.LinkedList;

public class LinkedListEx

{

public static void main(String args[])

{

LinkedList<String> cars=new LinkedList<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.addFirst("Mazda");

System.out.println(cars);

cars.addLast("Benz");

System.out.println(cars);

cars.removeFirst();

System.out.println(cars);

cars.removeLast();

System.out.println(cars);

System.out.println(cars.getFirst());

System.out.println(cars.getLast());

}

}

**Output:-**

[Mazda, Volvo, BMW, Ford]

[Mazda, Volvo, BMW, Ford, Benz]

[Volvo, BMW, Ford, Benz]

[Volvo, BMW, Ford]

Volvo

Ford

**Hash set:-**

A hash set is a collection of Items where every item is unique and it is found in the java.util package.

Create a hash set object called cars that will store strings.

Import java.util.HashSet;

Hashset<String> cars=new HashSet<String>();

**Add items:-**

To add an items to it use the add() method.

**Check if an item exists:-**

To check whether an item exists in a HashSet, use the contains() method.

**Remove an item:-**

To remove an item use the remove() method.

**Clear:-**

To remove all the items we use the clear() method.

**HashSet size:-**

To findout how many items there are use the size() method.

Ex:-

import java.util.HashSet;

public class HashSetEx

{

public static void main(String args[])

{

HashSet<String> cars=new HashSet<String>();

cars.add("Volvo");

cars.add("BMW");

cars.add("Ford");

cars.add("BMW");

cars.add("Mazda");

System.out.println(cars);

System.out.println(cars.contains("Mazda"));

cars.remove("Volvo");

System.out.println(cars);

cars.clear();

System.out.println(cars);

}

}

**Output:-**

[Volvo, Mazda, Ford, BMW]

true

[Mazda, Ford, BMW]

[]

**Hash map:-**

A hash map stores items in “key/value” pairs, and you can access them by an index of another type eg.a string.

One object is used as a key(index) to another object(value). It can store different types:String keys and integer values are the same type like string keys and string values.

Create a HashMap object called capitalcities that will store string keys and string values.

import java.util.HashMap;

hashMap<String,String>CapitalCities=new HashMap<String,String>();

**Add items:-**

In hashmap class to add items use the put() method.

**Access an item:-**

To access a value in the HashMap use the get() method and refer to it’s key.

**Remove an item:-**

To remove an item use the remove() method and refer to the key.

**Clear:-**

To remove all items use the clear() method.

**HashMap size:-**

To findout how many items there are, use the size() method.

Ex:-

import java.util.HashMap;

public class HashMapEx

{

public static void main(String args[])

{

HashMap<String,String> CapitalCities=new HashMap<String,String>();

CapitalCities.put("England","London");

CapitalCities.put("Germany","Berlin");

CapitalCities.put("Norway","Oslo");

CapitalCities.put("USA","Washington DC");

System.out.println(CapitalCities);

System.out.println(CapitalCities.get("England"));

CapitalCities.remove("England");

System.out.println(CapitalCities);

CapitalCities.clear();

System.out.println(CapitalCities);

System.out.println(CapitalCities.size());

}

}

**Output:-**

{USA=Washington DC, Norway=Oslo, England=London, Germany=Berlin}

London

{USA=Washington DC, Norway=Oslo, Germany=Berlin}

{}

0

**AWT(Abstract Window toolkit):-**

**Window:-**

A window represents a box shaped area on screen. Window does not have border and title. A frame is a top level window that is not contained in another window. A frame contains border and title.

**Creating the frame:-**

\*We can create a frame by creating frame class object.

Frame obj=new Frame();

\*Create a class that extends Frame class then create a object to that class.

class MyClass extends Frame

MyClass obj = new MyClass();

\*After creating the frame we need to set width and height using setSize() method as:

obj.setSize(400,350);

\*we can display the frame using setVisible() method as :

obj.setVisible(true);

Ex:-

import java.awt.\*;

class MyFrame

{

public static void main(String args[])

{

Frame f1=new Frame();

f1.setSize(500,150);

f1.setTitle("GUI World");

f1.setVisible(true);

}

}

**Code to close a frame:-**

public void windowClosing(windowEvent we)

{

System.exit(0);

}

Ex:-

import java.awt.\*;

import java.awt.event.\*;

class MyFrame2 extends Frame

{

public static void main(String args[])

{

MyFrame2 f1=new MyFrame2();

f1.setSize(500,200);

f1.setTitle("GUI WORLD");

f1.setVisible(true);

f1.addWindowListener(new MyClass());

}

}

class MyClass extends WindowAdapter

{

public void windowClosing(WindowEvent ie)

{

System.exit(0);

}

}

**Creating push buttons:-**

Button class is useful to create push buttons. A push button triggers a series of events.

\*To create push button: Button b1=new Button(“label”);

\*To get the label of the button: String l=bl.getLabel();

\*To set the label of the button: bl.setLabrl(“Label”);

\*To get the label of the button clicked :String str=ae.getActionCommand();

\*where ae is the object of ActionEvent.

Ex:-

import java.awt.\*;

import java.awt.event.\*;

class MyButton extends Frame implements ActionListener

{

Button b1,b2;

MyButton()

{

setLayout(null);

b1=new Button("Red");

b2=new Button("Yellow");

b1.setBounds(100,50,75,40);

b2.setBounds(100,120,75,40);

b1.addActionListener(this);

b2.addActionListener(this);

add(b1);

add(b2);

addWindowListener(new WindowAdapter()

{

public void windowClosing(WindowEvent We)

{

System.exit(0);

}

});

}

public void actionPerformed(ActionEvent ae)

{

String str=ae.getActionCommand();

if(str.equals("Red"))

setBackground(Color.red);

if(str.equals("Yellow"))

setBackground(Color.yellow);

}

public static void main(String args[])

{

MyButton ob=new MyButton();

ob.setSize(500,200);

ob.setTitle("Buttons....!");

ob.setVisible(true);

}

}

**Checkbox:-**

A checkbox is a square shaped box which provides set of options to the user.

\*To create a checkbox

Checkbox cb=new Checkbox("label");

\*To create a checked checkbox

checkbox cb=new Checkboc("label",null,true);

\*To get the state of a checkbox

boolean b=cb.getState();

\*To set the state of a checkbox

cb.setState(true);

\*To get the label of a checkbox

String s=cb.getLabel();

Ex:-

import java.awt.\*;

import java.awt.event.\*;

class MyCheckbox extends Frame implements ItemListener

{

Checkbox cb1,cb2;

String msg;

MyCheckbox()

{

setLayout(new FlowLayout());

cb1=new Checkbox("Bold",null,true);

cb2=new Checkbox("italic");

add(cb1);

add(cb2);

cb1.addItemListener(this);

cb2.addItemListener(this);

addWindowListener(new WindowAdapter()

{

public void windoeClosing(WindowEvent we)

{

System.exit(0);

}

});

}

public void itemStateChanged(ItemEvent ie)

{

repaint();

}

public static void main(String args[])

{

MyCheckbox ob=new MyCheckbox();

ob.setTitle("Checkbox Demo");

ob.setSize(500,200);

ob.setVisible(true);

}

public void paint(Graphics g)

{

g.drawString("Checkbox state:",20,100);

msg="Bold:"+cb1.getState();

g.drawString(msg,20,120);

msg="italic:"+cb2.getState();

g.drawString(msg,20,160);

}

}

**Choice menu:-**

Choice menu is a pop down list of items. Only one item can be selected.

\*To create choice menu

Choice ch=new Choice();

\*To add items to the choice menu

ch.add("text");

\*To know the name of the item selected from the choice menu

String s=ch.getSelectedItem();

\*To know the index of the currently selected item

int i=h,getSelectedIndex();

\*This method returns -1, if nothing is selected.

Ex:-

import java.awt.\*;

import java.awt.event.\*;

class MyChoice extends Frame implements ItemListener

{

Choice ch;

MyChoice()

{

setLayout(new FlowLayout());

ch=new Choice();

ch.add("India");

ch.add("Pakistan");

ch.add("China");

ch.add("Afganistan");

ch.add("Sri Lanka");

ch.add("Bamgladesh");

add(ch);

ch.addItemListener(this);

addWindowListener(new WindowAdapter()

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

});

}

public void itemStateChanged(ItemEvent ie)

{

repaint();

}

public void paint(Graphics g)

{

g.drawString("U selected:",20,100);

g.drawString(ch.getSelectedItem(),20,120);

}

public static void main(String args[])

{

MyChoice ob=new MyChoice();

ob.setSize(500,200);

ob.setTitle("ChoiceDemo");

ob.setVisible(true);

}

}

**Radio button:-**

A radio button represents a round shaped button such that only can be selected from a panel. Radio buttons can be selected using checkbox group class and checkbox classes.

\*To create a radio button

CheckboxGroup cbg=new CheckboGroup();

\*Checkbox cb=new Checkbox("label",cbg,true);

\*To know the selected Checkbox

Checkbox cb =cbg.getSelectedCheckbox();

\*To know the selected checkbox label

String label=cbg.getSelectedCheckbox().getLabel();

Ex:-

import java.awt.\*;

import java.awt.event.\*;

class MyRadio extends Frame implements ItemListener

{

String msg=" ";

CheckboxGroup cbg;

Checkbox y,n;

MyRadio()

{

setLayout(new FlowLayout());

cbg=new CheckboxGroup();

y=new Checkbox("Yes",cbg,true);

n=new Checkbox("No",cbg,false);

add(y);

add(n);

y.addItemListener(this);

n.addItemListener(this);

addWindowListener(new WindowAdapter()

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

});

}

public void itemStateChanged(ItemEvent ie)

{

repaint();

}

public void paint(Graphics g)

{

g.drawString("You Selected"+cbg.getSelectedCheckbox().getLabel(),100,50);

}

public static void main(String args[])

{

MyRadio ob=new MyRadio();

ob.setTitle("RadioButton");

ob.setBackground(Color.red);

ob.setSize(600,200);

ob.setVisible(true);

}

}

**Label:-**

\*A label is a constant text i.e.. is displayed with a text.

\*To create a label

Label l=new Label(“text”,alignmentConstant);

**Note:-**

alignmentConstant: Label.Right, Label.Left, Label.CENTER

**Textfield:-**

TextField allows a user to enter a single line of text.

\*To create a text field

TextField tf=new TextField(25);

(or)

TextField tf=new TextField("defaulttext",25);

\*To get the text from a textfield

String s=tf.getText();

\*To set the text into a text field

tf.setText("text");

\*To hide the text being typed into the TextField by a character

tf.setEchoChar('char');

**Text Area:-**

Text area is similar to a textfield but it accepts more than one line of text.

\*To create a TextArea

TextArea ta=new TextArea();

(or)

TextArea ta=new TextArea(rows,columns);

**Note:-**

Textarea supports getText() and setText method.

Ex:-

import java.awt.\*;

import java.awt.event.\*;

class MyText extends Frame implements ActionListener

{

Label n,p;

TextField name,pass;

MyText()

{

setLayout(new FlowLayout());

n=new Label("Enter name:",Label.RIGHT);

p=new Label("Enter password:",Label.RIGHT);

name=new TextField(20);

pass=new TextField(20);

pass.setEchoChar('\*');

add(n); add(name);

add(p); add(pass);

name.addActionListener(this);

pass.addActionListener(this);

addWindowListener(new WindowAdapter()

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

});

}

public void actionPerformed(ActionEvent ae)

{

repaint();

}

public void paint(Graphics g)

{

g.drawString("Name:"+name.getText(),50,120);

g.drawString("Pass:"+pass.getText(),50,150);

}

public static void main(String args[])

{

MyText ob=new MyText();

ob.setSize(500,200);

ob.setTitle("User name and password demo");

ob.setVisible(true);

}

}