

JAVA Installation and Execution Steps

1.Download jdk

<https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html>

<https://www.oracle.com/in/java/technologies/javase/javase-jdk8-downloads.html>
or

<https://www.oracle.com/java/technologies/downloads/#java8-linux>

2.Copy to the respective folder

3. Extract the file
`tar zxvf jdk-8u301-linux-x64.tar.gz`

4. Set the path in your programs folder

`export PATH=$PATH:/home/boss/Downloads/software/jdk1.8.0_251/bin/`

5. Write a simple program

6. Save the program with .java extension

7. Compile the program

8. Execute the program

Program1

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

Save the program as '**sample.java**'

Compile with the command

javac sample.java

Execute the program

java sample.java

Program2

class Hello

```
{
public static void main(String args[])
{
System.out.println("Hello World");
}
}
```

(extra) //Command line arguments

class CmdLine

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

int count=args.length;
for(i=0;i<count;i++)
{
System.out.println(args[i]); //args[0],args[1],args[2],args[3],args[4]
}

//System.out.println("Hello World");
}
}
```

(extra) //method overloading

```
class ClassexamplewithMethods {

public static void main(String[] args) {
//final int var1=100;
int ans=methodA(100,200);
int ans1=methodA(100,200,300);
System.out.println(ans);

System.out.println(ans1);
System.out.println(methodA("2000",'v'));
// var1=200;
}
static int methodA(int a,int b)
{

//return(a+b);
```

```

        int temp;
        temp=a+b;
        System.out.println("inside first method");
        return(temp);
    }
    static int methodA(int a,int b,int c)
    {
        System.out.println("\n inside 2nd method");
        return(a+b+c);
        /*int temp;
        temp=a+b+c;

        return(temp);*/
    }
    static String methodA(String a,char c)
    {

        System.out.println("\n inside third method");
        String s= a+c;
        return(s);
    }
}

```

java CmdLine Rose Lilly Jasmine Hibiscus Lotus

Program3

```

class Numbers
{
    public static void main(String args[])
    {
        int a=10,b=15;
        Char c1='A',c2='B';
        System.out.println("Addition of numbers"+(a+b));
        System.out.println("Concatenation of characters"+c1+c2);
    }
}

```

(Extra)

```

class TernaryEx
{
    public static void main(String args[])
    {
        int a=10;
        int b=5;
    }
}

```

```
int great=(a>b)?a:b;

System.out.println("greatest number"+great);
}
}
```

```
(Extra)
class ForEx
{
public static void main(String args[])
{
//int i;

for (int i=1; i<=10; i++)
{
System.out.println("i="+i);
if (i==5)
//break;
continue;
System.out.println("Welcome");
}

}
}
```

Program4 (Calculate compound interest)

```
import java.util.*;
import java.lang.Math.*;
class Interest
{
public static void main(String args[])
{
double principal,rate;

Scanner sc=new Scanner(System.in);
System.out.println("Enter the Principal");
principal=sc.nextInt();
System.out.println("Enter the rate of interest");
rate=sc.nextInt();

System.out.println("Enter the number of times interest is compounded");
int n=sc.nextInt();

System.out.println("Enter the number of time periods");
int t=sc.nextInt();
```

```

double CI=principal*Math.pow((1.0+(rate/n)),(n*t));
System.out.println("Compound interest Calculated"+CI);
}
}

```

Program5 (Calculate power of a number)

--do it--

Program 6 (Swap two numbers)

```

import java.util.Scanner;

public class SwapTwoNumbers {

    public static void main(String[] args) {

        int x, y, temp;

        System.out.println("Enter x and y");

        Scanner in = new Scanner(System.in);

        x = in.nextInt();

        y = in.nextInt();

        System.out.println("Before Swapping " + x + " " + y);

        temp = x; //moving x value to temp so x is empty

        x = y; //move y to empty x variable

        y = temp; //now finally move temp value to empty y variable

        System.out.println("After Swapping " + x + " " + y);

    }

}

```

Program 7 (Calculate area of a rectangle)

```

import java.util.*;

class Area
{
    public static void main(String args[])
    {
        int l,b;
        System.out.println("Enter Length and Breadth of Rectangle");
        Scanner in = new Scanner(System.in);
    }
}

```

```

        l = in.nextInt();
        b = in.nextInt();
        System.out.println("Area of Rectangle "+(l*b));

    }
}

```

(extra) //factorial using recursion

```

class Factorial
{
    public static void main(String args[])
    {
        int num=7;
        System.out.println(fact(num));
    }

    static int fact(int n)
    {

        if (n==1)
            return(1);
        else
            return(n*fact(n-1)); //5*fact(4) 5*4*fact(3)...1

    }

}

```

(extra)

```

import java.util.*;
public class Array2DEx
{
    public static void main(String args[])
    {
        int array2d[][]=new int[10][10];

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number of rows");
        int m=sc.nextInt();

        System.out.println("Enter the number of columns");
        int n=sc.nextInt();

        System.out.println("Enter the elements of the array");
        for (int i=0;i<m;i++)
            for(int j=0;j<n;j++)
                array2d[i][j]=sc.nextInt();
    }
}

```

```

System.out.println("Array Elements");
for (int i=0;i<m;i++)
    for(int j=0;j<n;j++)

System.out.println(array2d[i][j]);

}
}

```

Program 8 (Calculate area and circumference of a circle)

---do it---

Program 9 (To find ASCII value of a character)

```

import java.util.*;
class ascii
{
    public static void main(String args[])
    {
        char c;
        System.out.println("Enter a character");
        Scanner in = new Scanner(System.in);
        c = in.next().charAt(0);
        int a=c;

        System.out.println("ASCII value of character "+a);

    }
}

```

Program 10

```

import java.util.*;
class defaultvalues
{
    static int i;
    static float f;
    static char c;
    static double d;

    public static void main(String args[])

```

```

{
System.out.println("Default values of primitive data types Integer - "+i+"float -"+f+"char
-"+c+"double -"+d);

}
}

```

Program 11 (Swap two values without using third variables)

class swapping

```

{
public static void main(String args[])
{
int x = 10, y = 50;

x = x + y; // x = 60
y = x - y; // y = 10
x = x - y; // x = 50
System.out.println("After swapping x= "+x+" y= "+y);
}
}

```

Program 12 (Fibonacci Series)

class fibonacci

```

{
public static void main(String args[])
{
int n1=0,n2=1,n3,i,count=10;
System.out.print(n1+" "+n2);//printing 0 and 1

for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed
{
n3=n1+n2;
System.out.print(" "+n3);
n1=n2;
n2=n3;
}

}
}
(extra) (Methods)
class mathsprob
{
public static void main(String args[])
{
int a=100,b=200;
int res=add(a,b); //pass by reference

```



```

Int res=add(34,67); //pass by value

int ressub=sub(a,b);
int resmul=mul(a,b);

System.out.println("Addition"+res);
System.out.println("Subtraction"+ressub);
System.out.println("Multiplication"+resmul);

int c=1000,d=2000;
int res1=add(c,d);
System.out.println(res1);

}

static int add (int num1,int num2) //num1=a(100), num2=b(200)
{
int temp;
temp=num1+num2;
return temp;
}
static int sub (int num1,int num2) //num1=a(100), num2=b(200)
{
int temp;
temp=num1-num2;
return temp;
}
static int mul (int num1,int num2) //num1=a(100), num2=b(200)
{
int temp;
temp=num1*num2;
return temp;
}

}

```

Program 13 (Factorial of a number)

```

class factorial
{
public static void main(String args[])
{
int i, fact=1;
int number=5;//It is the number to calculate factorial
for(i=1;i<=number;i++){
fact=fact*i;
}
}
}

```

```

        System.out.println("Factorial of "+number+" is: "+fact);
    }
}

```

Program 14 (Prime numbers)

```

class prime
{
    public static void main(String args[])
    {
        int flag,m,i;
        for (int n=2;n<=100;n++)
        {
            flag=0;
            m=n/2;
            for(i=2;i<=m;i++){
                if(n%i==0){

                    flag=1;
                    break;
                }
            }
            if(flag==0) { System.out.println(n+" is prime number"); }
        }
    }
}

```

Program 15 (Palindrome)

```

import java.util.*;

class palindrome
{
    public static void main(String args[])
    {
        System.out.println("Enter the number");
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int sum=0,r;
        int temp=n;
        while(n>0){
            r=n%10; //getting remainder 4 3
            sum=(sum*10)+r; 0+3=3 30+4=34 340+3 =343
            n=n/10; 34 3 0
        }
        if(temp==sum)

```

```

        System.out.println("palindrome number ");
    else
        System.out.println("not palindrome");
    }

}

```

Program 16 (square root of a number)

```

import java.lang.Math.*;
import java.util.*;
class sqrt
{
    public static void main(String args[])
    {
        System.out.println("Enter the number");
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        System.out.println(java.lang.Math.pow(n,0.5));
    }
}

```

Program 16 (armstrong of a number)

---do it---

Program 17 (grades of students using their marks)

```

import java.util.*;
class grades
{
    public static void main(String args[])
    {
        int marks[] = new int[6];
        int i;
        float total=0, avg;
        Scanner scanner = new Scanner(System.in);

        for(i=0; i<6; i++) {
            System.out.print("Enter Marks of Subject" +(i+1) + " :");
            marks[i] = scanner.nextInt();
            total = total + marks[i];
        }
        scanner.close();
        //Calculating average here
        avg = total/6;
        System.out.print("The student Grade is: ");
        if(avg>=80)

```

```

    {
        System.out.print("A");
    }
    else if(avg>=60 && avg<80)
    {
        System.out.print("B");
    }
    else if(avg>=40 && avg<60)
    {
        System.out.print("C");
    }
    else
    {
        System.out.print("D");
    }
}
}

```

Program 18 ()

```

public class Switchex {
    public static void main(String[] args)
    {
        int day = 2;
        String dayType;
        String dayString;

        switch (day) {
            case 1:
                dayString = "Monday";
                break;
            case 2:
                dayString = "Tuesday";
                break;
            case 3:
                dayString = "Wednesday";
                break;
            case 4:
                dayString = "Thursday";
                break;
            case 5:
                dayString = "Friday";
                break;
            case 6:
                dayString = "Saturday";
                break;
            case 7:
                dayString = "Sunday";

```

```

        break;
    default:
        dayString = "Invalid day";
    }

    switch (day) {
        // multiple cases without break statements

        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
            dayType = "Weekday";
            break;
        case 6:
        case 7:
            dayType = "Weekend";
            break;

        default:
            dayType = "Invalid daytype";
    }

    System.out.println(dayString + " is a " + dayType);
}
}

```

Program 19 (Average of numbers)

```

class Testarray{

    public static void main(String args[]){

        int a[]=new int[5];//declaration and instantiation

        a[0]=10;//initialization

        a[1]=20;

        a[2]=70;

        a[3]=40;

        a[4]=50;

        int sum=0;
    }
}

```

```
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
{
sum=sum+a[i];
System.out.println(a[i]); }
System.out.println("Average of array values"+(sum/a.length));
}}
```

Program 20 (Reverse an array)

```
class reversearray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
int b[]=new int[5];
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
int sum=0;
//traversing array
for(int i=a.length-1,j=0;i>=0&& j<a.length;i--,j++) //length is the
property of array
{
b[j]=a[i];
System.out.println(b[j]); }
```

```
}}
```

Program21 (Ascending order)

```
public class SortAsc {  
    public static void main(String[] args) {  
  
        //Initialize array  
        int [] arr = new int [] {5, 2, 8, 7, 1};  
        int temp = 0;  
  
        //Displaying elements of original array  
        System.out.println("Elements of original array: ");  
        for (int i = 0; i < arr.length; i++) {  
            System.out.print(arr[i] + " ");  
        }  
  
        //Sort the array in ascending order  
        for (int i = 0; i < arr.length; i++) {  
            for (int j = i+1; j < arr.length; j++) {  
                if(arr[i] > arr[j]) {  
                    temp = arr[i];  
                    arr[i] = arr[j];  
                    arr[j] = temp;  
                }  
            }  
        }  
    }  
}
```

```
    }  
}
```

```
System.out.println();
```

```
//Displaying elements of array after sorting
```

```
System.out.println("Elements of array sorted in ascending  
order: ");
```

```
for (int i = 0; i < arr.length; i++) {  
    System.out.print(arr[i] + " ");  
}  
}  
}
```

Program22 (Ascending order)

***Change it with getting input values for the array

Program22 (matrix addition)

```
class matrixaddition {
```

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

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

```
int b[][]={{1,3,4},{2,4,3},{1,2,4}};
```

```
//creating another matrix to store the sum of two matrices
```



```
int c[][]=new int[3][3]; //3 rows and 3 columns
```

```
//adding and printing addition of 2 matrices
```

```
for(int i=0;i<3;i++){
```

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

```
c[i][j]=a[i][j]+b[i][j]; //use - for subtraction
```

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

```
}
```

```
System.out.println();//new line
```

```
}
```

```
}
```

```
}
```

Program23 (matrix addition)

```
class matrixaddition {
```

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

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

```
int b[][]={{1,3,4},{2,4,3},{1,2,4}};
```

```
    //creating another matrix to store the sum of two matrices
```

```
int c[][]=new int[3][3]; //3 rows and 3 columns
```

```
    //adding and printing addition of 2 matrices
```

```
for(int i=0;i<3;i++){
```

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

```

c[i][j]=a[i][j]+b[i][j];    //use - for subtraction
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}
}
}

```

Program24 (names in alphabetical order)

```

import java.io.*;
class StringSorting {
    public static void main(String[] args)
    {
        // storing input in variable
        int n = 4;

        // create string array called names
        String names[]
            = { "Rahul", "Ajay", "Gourav", "Riya" };
        String temp;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {

                // to compare one string with other strings

```

```

        if (names[i].compareTo(names[j]) > 0) { // if a[i]<a[j]
            // swapping
            temp = names[i];
            names[i] = names[j];
            names[j] = temp;
        }
    }
}

// print output array
System.out.println(
    "The names in alphabetical order are: ");
for (int i = 0; i < n; i++) {
    System.out.println(names[i]);
}
}
}

```

Program25 (names in alphabetical order with input)

```

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

```

```
// storing input in variable
int n ;

// create string array called names
String names[]=new String[10];
System.out.println("Enter number of names");
Scanner sc=new Scanner(System.in);
n=sc.nextInt();
System.out.println("Enter names");
for (int i=0;i<n;i++)
{

names[i]=sc.nextLine();
}

String temp;
for (int i = 0; i < n; i++) {
for (int j = i + 1; j < n; j++) {

// to compare one string with other strings
if (names[i].compareTo(names[j]) > 0) {
// swapping
temp = names[i];
names[i] = names[j];
```

```

        names[j] = temp;
    }
}
}

// print output array
System.out.println(
"The names in alphabetical order are: ");
for (int i = 0; i < n; i++) {
    System.out.println(names[i]);
}
}
}

```

Program26 (Static variable)

```

class Counter2{
    static int count=0;

    Counter2(){
        count++;

        System.out.println(count);
    }

    public static void main(String args[]){

        Counter2 c1=new Counter2();
    }
}

```

```
Counter2 c2=new Counter2();  
  
Counter2 c3=new Counter2();  
  
}  
  
}
```

Program27 (Static method)

```
class Calculate{  
  
    static int cube(int x){  
  
        return x*x*x;  
  
    }  
  
    public static void main(String args[]){  
  
        int result=Calculate.cube(5);  
  
        System.out.println(result);  
  
    }  
  
}
```

Program28 (final variable)

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

```
{  
  
final int count=100;  
  
count=150;  
  
}  
  
}
```

Program for string comparison

```
public class CheckString {  
    public static void main(String[] args) {  
        String firstString = "My Name Is Gaurav!";  
        String secondString = "my name is gaurav!";  
  
        // Case 1  
  
        // Check if the strings are same using the simple equals()  
        method  
  
        System.out.println("checking using equals() method : " +  
firstString.equals(secondString));  
  
        // Case 2  
  
        // Check if the strings are same using the  
equalsIgnoreCase() method  
  
        System.out.println("checking using equalsIgnoreCase()  
method : " + firstString.equalsIgnoreCase(secondString));  
    }  
}
```

```
}
```

(28a) Strings

```
public class StringExample
```

```
{
```

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

```
{
```

```
String s=new String("ACTS");
```

```
String s1=s.concat("JAVA Session");
```

```
StringBuffer s2=new StringBuffer("CDAC");
```

```
s2.append("chennai");
```

```
StringBuilder s3=new StringBuilder(s2);
```

```
StringBuilder s4=s3.reverse();
```

```
System.out.println("S1= "+s1);
```

```
System.out.println("S2= "+s2);
```

```
System.out.println("S3= "+s3);
```

```
System.out.println("S4= "+s4);
```

```
}
```



```
}
```

Program for copying arrays

```
import java.util.Arrays;
```

```
public class CopyArray {
```

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

```
        // original array
```

```
        int arr[] = {10, 20, 30, 40, 50};
```

```
        // copy array using assignment operator
```

```
        int newArr[] = arr;
```

```
        // display array
```

```
        System.out.println("Original Array = " + Arrays.toString(arr));
```

```
        System.out.println("Copied Array = " +  
Arrays.toString(newArr));
```

```
    }
```

```
}
```

program

```
class MaxArray
{
    public static void main(String args[])
    {
        int arr[]={143,122,89,333,2322};
        int maxvalue=largest(arr);
        System.out.println(maxvalue);
    }
    public static int largest(int[] array) {
        // declare a variable max
        int max = 0;
        // assign first element to max
        max = array[0];
        // compare with remaining elements
        // loop
        for (int i = 1; i < array.length; i++) {
            // compare
            if (max < array[i]) max = array[i];
        }
        return max;
    }
}
```

```
}
```

```
class stringex
```

```
{
```

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

```
{
```

```
String s="Java programming";
```

```
char[] s1=s.toCharArray();
```

```
System.out.println(s1[0]);
```

```
}
```

```
}
```

Program for another way of input

```
import javax.swing.*;
```

```
import java.text.*;
```

```
public class swingex
```

```
{
```

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

```
{
```

```
String name= JOptionPane.showInputDialog("Your name :");
```

```
String input= JOptionPane.showInputDialog("Your age :");
```

```
int age = Integer.parseInt(input);
```

```
NumberFormat.getNumberInstance(); // for numbers
```

```
NumberFormat.getCurrencyInstance();// for currency values  
NumberFormat.getPercentInstance();// for percentage values
```

```
double x = 10000.0 / 3.0;  
NumberFormat nf = NumberFormat.getNumberInstance();  
nf.setMaximumFractionDigits(4);  
nf.setMinimumIntegerDigits(6);  
System.out.println(name);  
System.out.println(input);  
System.out.println(nf.format(x)); //003,333.3333  
}  
}
```

Program for Odd numbers

Program for armstrong numbers ($153=1+125+9=153$)

Program to print number of elements in an array

Program to convert to string

```
class stringtoint
```

```
{
```

```
public static void main(String args[])
{
String year="1999";
int year1=Integer.parseInt(year);
int year2=Integer.valueOf(year);
System.out.println(year1);
System.out.println(year2);
int i=10;
System.out.println(Integer.toString(i));

}
}
```

Program

//Create a class account with 3 fields, write respective constructor, methods for display withdraw

//and deposit, create object and invoke the methods by user choice

```
import java.util.*;

public class Account
{
private int acno;
```

```
private String name;
```

```
private double balance;
```

```
public Account(int acno,String name, double balance)
```

```
{
```

```
    this.acno=acno;
```

```
    this.name=name;
```

```
    this.balance=balance;
```

```
}
```

```
public void display()
```

```
{
```

```
    System.out.println("Acno "+acno);
```

```
    System.out.println("Name "+name);
```

```
    System.out.println("Balance "+balance);
```

```
}
```

```
public void deposit(double amt)
```

```
{
```

```
    balance+=amt;
```

```
}
```

```
public void withdraw(double amt)
```

```
{
```

```
    balance-=amt;
```

```
}
```

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

```
{
```

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

```
    System.out.println("Enter the name ");
```

```
    String nm=sc.nextLine();
```

```
    System.out.println("Enter the Account number ");
```

```
    int acnum=sc.nextInt();
```

```
    System.out.println("Enter the balance ");
```

```
    double b=sc.nextDouble();
```

```
    Account account1=new Account(acnum,nm,b);
```

```
    account1.display();
```

```
    System.out.println("Enter your choice (1 for deposit or 2 for  
    withdrawal");
```

```
int choice=sc.nextInt();
```

```
System.out.println("Enter the transaction amount");
```

```
double amount=sc.nextDouble();
```

```
if (choice==1)
```

```
account1.deposit(amount);
```

```
else if (choice ==2)
```

```
account1.withdraw(amount);
```

```
else
```

```
System.out.println("Invalid input");
```

```
account1.display();
```

```
}
```

```
}
```

Program

```
//method overloading
```

```
import java.util.*;
```

```
class testmethods
```

```
{
```

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



```
{  
  
Scanner sc=new Scanner(System.in);  
  
System.out.println("Enter the value of a, b and c");  
  
int a=sc.nextInt();  
  
int b=sc.nextInt();  
  
int c=sc.nextInt();  
  
double e=sc.nextDouble();  
  
double d=sc.nextDouble();  
  
add(a);  
  
add(a,b);  
  
add(a,b,c);  
  
add(e,b,d);  
  
}
```

```
static void add(int var1)  
{  
  
System.out.println("Inside 1st method");  
  
var1+=10;  
  
System.out.println(var1);  
  
}
```

```
static void add(int var1,int var2)
{
System.out.println("Inside 2nd method");
var1=var1+var2;
System.out.println(var1);
}
```

```
static void add(int var1,int var2,int var3)
{
System.out.println("Inside 3rd method");
var1=var2*var3;
System.out.println(var1);
}
```

```
static void add(double var1,int var2,double var3)
{
System.out.println("Inside 4th method");
var1=var2*var3;
System.out.println(var1);
}
}
```

Program of Account class with getter and setter

//Create a class account with 3 fields, write respective constructor, methods for display withdraw

//and deposit, create object and invoke the methods by user choice

```
import java.util.*;
```

```
class Account
```

```
{
```

```
    private int acno;
```

```
    private String name;
```

```
    private double balance;
```

```
    public int getAcno()
```

```
    {
```

```
        return acno;
```

```
    }
```

```
    //setter method
```

```
    public void setAcno(int acno)
```

```
    {
```

```
        this.acno=acno;
```

```
}
```

```
public String getname()
```

```
{
```

```
return name;
```

```
}
```

```
public double getbalance()
```

```
{
```

```
return balance;
```

```
}
```

```
public Account(int acno,String name, double balance)
```

```
{
```

```
this.acno=acno;
```

```
this.name=name;
```

```
this.balance=balance;
```

```
}
```

```
public void display()
```

```
{  
System.out.println("Acno "+acno);  
System.out.println("Name "+name);  
System.out.println("Balance "+balance);  
}
```

```
public void deposit(double amt)  
{  
balance+=amt;  
}
```

```
public void withdraw(double amt)  
{  
balance-=amt;  
}  
}
```

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

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

System.out.println("Enter the name ");

String nm=sc.nextLine();

System.out.println("Enter the Account number ");

int acnum=sc.nextInt();

System.out.println("Enter the balance ");

double b=sc.nextDouble();

Account account1=new Account(acnum,nm,b);

account1.display();

System.out.println("Enter your choice (1 for deposit or 2 for
withdrawal");

int choice=sc.nextInt();

System.out.println("Enter the transaction amount");

double amount=sc.nextDouble();

if (choice==1)

account1.deposit(amount);

else if (choice ==2)

account1.withdraw(amount);

else

System.out.println("Invalid input");

account1.display();
```

```
//System.out.println(account1.acno);  
System.out.println("Account no is "+account1.getAcno());  
account1.setAcno(1000);  
System.out.println("Account no is "+account1.getAcno());  
}  
}
```

Program

```
// static and non static instance fields  
class Employee  
{  
    int id = assignId();  
    static int nextId=1;  
    static int assignId()  
    { int r = nextId;  
      nextId++;  
      return r;  
    }  
  
    public static void main(String args[])
```

```
{  
System.out.println(Employee.nextId);  
Employee e1=new Employee();  
System.out.println(e1.id);  
Employee e2=new Employee();  
System.out.println(e2.id);  
}  
}
```

Program with toString() method

```
class Student{  
    int rollno;  
    String name;  
    String city;  
  
    Student(int rollno, String name, String city){  
        this.rollno=rollno;  
        this.name=name;  
        this.city=city;  
    }  
  
    public String toString(){//overriding the toString() method
```



```
        return rollno+" "+name+" "+city;
    }

    public static void main(String args[]){

        Student s1=new Student(101,"Raj","lucknow");

        Student s2=new Student(102,"Vijay","ghaziabad");


        System.out.println(s1);//compiler writes here s1.toString()

        System.out.println(s2);//compiler writes here s2.toString()

    }

}
```

Program //inheritance

//Inheritance

```
class Emp
{
    protected int empno;

    private String name;

    private String qualification;

    Emp(int empno,String name,String qualification)
```

```
{  
  
this.empno=empno;  
  
this.name=name;  
  
this.qualification=qualification;  
  
}  
  
void show()  
  
{  
  
System.out.println(name);  
  
System.out.println(empno);  
  
System.out.println(qualification);  
  
}  
  
}
```

```
class PartTimeEmp extends Emp  
{  
  
private int noofhrs;  
  
private int rate;  
  
PartTimeEmp(int empno,String name,String qualification,int  
noofhrs,int rate)  
  
{
```

```
super(empno,name,qualification);
```

```
this.noofhrs=noofhrs;
```

```
this.rate=rate;
```

```
}
```

```
void show()
```

```
{
```

```
super.show();
```

```
System.out.println(noofhrs);
```

```
System.out.println(rate);
```

```
}
```

```
}
```

```
class RegularEmp extends Emp
```

```
{
```

```
int basic;
```

```
int da;
```

```
int pf;
```

```
RegularEmp(int empno,String name,String qualification,int basic,int  
da,int pf)
```

```
{
```

```
super(empno,name,qualification);
```

```
this.basic=basic;
```

```
this.da=da;
```

```
this.pf=pf;
```

```
}
```

```
void show()
```

```
{
```

```
super.show();
```

```
System.out.println(empno);
```

```
System.out.println(basic);
```

```
System.out.println(da);
```

```
System.out.println(pf);
```

```
}
```

```
}
```

```
class EmpSalary
```

```
{
```

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

```
{
```

```
PartTimeEmp pt2= new PartTimeEmp(111,"Pratik","MCA",40,300);  
  
pt2.show();  
  
//System.out.println("Salary "+ (pt2.noofhrs * pt2.rate));
```

```
RegularEmp remp2=new  
RegularEmp(222,"Avantika","BTech",40000,10000,8000);  
  
remp2.show();  
  
System.out.println("Salary "+ (remp2.basic + remp2.da - remp2.pf  
));  
  
}  
  
}
```

```
class A //superclass  
  
{  
  
protected int var1;  
  
A()  
  
{  
  
var1=100;  
  
}  
  
public void display()
```

```
{  
  
System.out.println("in A"+ var1);  
  
}  
  
}
```

```
class B extends A //inheriting A //reusability //subclass
```

```
{  
  
int var2;
```

```
B()
```

```
{  
  
var2=200;  
  
}
```

```
public void display() //overriding
```

```
{  
  
System.out.println("var1 "+var1);  
System.out.println("in B"+var2);  
  
}  
  
}
```

```
class testinheritance
```

```
{  
  
public static void main(String args[])  
  
{  
  
A aobj=new A();  
aobj.display();  
  
B bobj=new B();  
bobj.display();  
  
A aobj1=new B();  
aobj1.display();  
  
A aobj2;  
  
aobj2=new B();  
aobj2.display();  
  
B bobj1=new B();  
  
aobj2=bobj1;  
aobj2.display();
```

```
System.out.println(aobj.var1);
```

```
}
```

```
}
```

Program

```
class A //superclass
```

```
{
```

```
private int var1;
```

```
A()
```

```
{
```

```
var1=100;
```

```
}
```

```
public void calc(int r)
```

```
{
```

```
var1*=r;
```

```
}
```

```
public void display()
```

```
{
```

```
System.out.println("in A"+ var1);
```

```
}
```

```
}
```



```
class B extends A //inheriting A //reusability //subclass
```

```
{
```

```
int var2;
```

```
B()
```

```
{
```

```
var2=200;
```

```
}
```

```
public void display() //overriding
```

```
{
```

```
super.display();
```

```
System.out.println("in B"+var2);
```

```
}
```

```
}
```

```
class testinheritance
```

```
{
```

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

```
{
```

```
B bobj=new B();  
bobj.calc(10);  
bobj.display();  
}  
}
```

Program for multilevel inheritance

```
import java.util.*;  
  
class Person  
{  
    int id;  
    String name;  
    Person(int id,String name)  
    {  
        this.id=id;  
        this.name=name;  
    }  
    void displayvaluesPerson()  
    {  
        System.out.println("Id is "+id);  
    }  
}
```

```
System.out.println("Name is "+name);  
  
}  
  
}  
  
class Student extends Person  
{  
  
String collegename;  
  
int duration;  
  
Student(int id,String name, String collegename,int duration)  
{  
  
super(id,name);  
  
this.collegename=collegename;  
  
this.duration=duration;  
  
}  
  
void displayvalues()  
{  
  
displayvaluesPerson();  
  
System.out.println("Collegename is "+collegename);  
  
System.out.println("Duration of study is "+duration);  
  
}  
  
}
```

```
class Pgstudent extends Student
{
String specialisation;

Pgstudent(int id,String name, String collegename,int duration,String
specialisation)
{
super( id, name, collegename, duration);
this.specialisation=specialisation;
}

void displayvalues()
{
super.displayvalues();
System.out.println("Specialisation is "+specialisation);
}
}
```

```
class testmultilevelinheritance
{
public static void main(String args[])
{
Scanner sc =new Scanner (System.in);
```

```
System.out.println("Enter Id");

int id=sc.nextInt();

System.out.println("Enter Name");

String name=sc.next();

System.out.println("Enter collegename");

String collegename=sc.next();

System.out.println("Enter Duration");

int duration=sc.nextInt();

System.out.println("Enter Specialisation");

String spec=sc.next();

Pgstudent pgobj=new
Pgstudent(id,name,collegename,duration,spec);

pgobj.displayvalues();

}

}
```

Program for hierarchical inheritance

```
class Hostel

{

String hostelname;
```

```
Hostel()  
  
{  
  
    hostelname="Sunshine";  
  
}  
}
```

```
class Student extends Hostel  
  
{  
  
    String collegename;  
  
    Student()  
  
    {  
  
        collegename="ABC";  
  
    }  
  
}
```

```
class Employee extends Hostel  
  
{  
  
    String Designation;  
  
    Employee()  
  
    {  
  
        Designation="Manager";  
  
    }  
  
}
```

```
}
```

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

```
{
```

```
Employee e=new Employee();
```

```
Student s=new Student();
```

```
System.out.println(e.hostelname);
```

```
System.out.println(e.Designation);
```

```
System.out.println(s.hostelname);
```

```
System.out.println(s.collegename);
```

```
}
```

```
}
```

Program (Polymorphism)

```
import java.util.*;
```

```
class Shape
```

```
{
```

```
double area;
```

```
double PI=3.14;
```

```
public double area()
```

```
{
```

```
return area;
```

```
}
```

```
}
```

```
class Circle extends Shape
```

```
{
```

```
double radius;
```

```
Circle()
```

```
{
```

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

```
System.out.println("enter the radius of circle");
```

```
radius=sc.nextDouble();
```

```
}
```

```
public double area()
```

```
{
```

```
area=PI*radius*radius;
```

```
return area;
```

```
}
```



```
}
```

```
class Rectangle extends Shape
```

```
{
```

```
double length;
```

```
double width;
```

```
Rectangle()
```

```
{
```

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

```
System.out.println("enter the length of rectangle");
```

```
length=sc.nextDouble();
```

```
System.out.println("enter the width of rectangle");
```

```
width=sc.nextDouble();
```

```
}
```

```
public double area()
```

```
{
```

```
area=length*width;
```

```
return area;
```

```
}
```

}

class Triangle extends Shape

{

double breadth;

double height;

Triangle()

{

Scanner sc=new Scanner(System.in);

System.out.println("enter the breadth of triangle");

breadth=sc.nextDouble();

System.out.println("enter the width of triangle");

height=sc.nextDouble();

}

public double area()

{

area=0.5*breadth*height;

return area;

}

}

class TestShape

{

public static void main(String args[])

{

Shape s=new Shape();

Scanner sc=new Scanner(System.in);

**System.out.println("Enter your choice(1 for Circle, 2 for Rectangle, 3
for Triangle");**

int choice=sc.nextInt();

switch (choice)

{

case 1:

Circle c=new Circle();

s=c;

break;

case 2:

Rectangle r=new Rectangle();

s=r;

break;

case 3:

Triangle t=new Triangle();

s=t;

break;

default:

System.out.println("Invalid input");

}

double areaoutput=s.area();

System.out.println("Area of the shape is "+areaoutput);

System.out.println("Area of the shape is "+s.area());

}

}

Program //getter and setter methods

class Student

{

private int sno;

```
private String name;
```

```
//accessor method
```

```
public int getSno()
```

```
{
```

```
    return sno;
```

```
}
```

```
public String getName()
```

```
{
```

```
    return name;
```

```
}
```

```
//mutator methods
```

```
public void setNo(int sno)
```

```
{
```

```
    this.sno=sno;
```

```
}
```

```
public void setName(String name)
```

```
{
```

```
    this.name=name;
```

```
}
```

```
void methodA()
```

```
{
```

```
System.out.println(sno+" "+name);
```

```
}
```

```
}
```

```
class HostelStudent extends Student
```

```
{
```

```
int roomno;
```

```
public void setRoomno(int roomno)
```

```
{
```

```
this.roomno=roomno;
```

```
}
```

```
public int getRoomno()
```

```
{
```

```
return roomno;
```

```
}
```

```
public void methodB()
```

```
{  
    System.out.println(roomno);  
}  
}  
  
class TestSingleInheritance  
{  
    public static void main(String args[])  
    {  
        Student s1=new Student();  
        s1.setNo(101);  
        s1.setName("Neha");  
        //s1.methodA();  
        System.out.println(s1.getSno());  
        System.out.println(s1.getName());  
  
        HostelStudent hs=new HostelStudent();  
        hs.setNo(102);  
        hs.setName("Karthik");  
        hs.setRoomno(345);  
        System.out.println(hs.getSno());
```

```
System.out.println(hs.getName());  
System.out.println(hs.getRoomno());
```

```
int num=hs.getSno();  
System.out.println(num);
```

```
//hs.methodA();  
//hs.methodB();
```

```
}
```

```
}
```

Program //Abstract class

```
abstract class A
```

```
{
```

```
abstract void display();
```

```
}
```

```
class B extends A
```

```
{
```



```
void display()
{
    System.out.println("Welcome");
}
```

```
public static void main(String args[])
{
    A a1;

    B b1=new B();

    a1=b1;
    a1.display();
}
}
```

//abstract class example

```
abstract class Customer
{
```

```
int billamount;  
  
abstract void getinput();  
  
abstract void bill();  
  
abstract void display();  
  
}
```

```
class Retailcustomer extends Customer  
{  
    int phno;  
    int qty;  
  
    void getinput()  
    {  
        //getting inputs for phnp and qty;  
    }  
  
    void bill()  
    {  
        billamount=qty*1000;  
    }  
  
    void display()  
    {
```

```
//printing billamount;

}

}

class Wholesalecustomer extends Customer
{

int Address;

int qty;

int Maxqty;


void getinput()
{
//getting inputs for Address, qty and Maxqty;
}

void bill()
{
if qty<Maxqty
billamount =qty*950;
}

void display()
{
//printing billamount & supplied to the address;
```

```
}
```

```
}
```

```
class MainClass
```

```
{
```

```
psvm()
```

```
{
```

```
//create objects and invoke the methods
```

```
}
```

```
}
```

```
//abstract another example
```

```
abstract class Account{
```

```
    String Acno;
```

```
    double balance;
```

```
    abstract void withdraw(double amt);
```

```
    abstract double intrest();
```

```
}
```

```
class Saving extends Account{
```

```
    Saving(String Acno, double balance){
```

```

        this.Acno = Acno;

        this.balance = balance;

    }

    void withdraw(double amt){

        if((balance-amt) >5000){

            balance -= amt;

            System.out.println("Your Saving Ac balance is
"+balance);

        }else{

            System.out.println("Your balance is less than 5000
you can not withdraw\n");

        }

    }

    double intrest(){

        return (8.0/100)*balance;

    }

}

```

```

class Current extends Account{
    Current(String Acno, double balance){
        this.Acno = Acno;
        this.balance = balance;
    }
    void withdraw(double amt){
        if((balance-amt) >10000){
            balance -= amt;
            System.out.println("Your Current Ac balance is
"+balance);
        }else{
            System.out.println("Your balance is less than 10000
you can not withdraw");
        }
    }
    double intrest(){
        return (10.0/100)*balance;
    }
}

```

```

public class HelpAbstract{

```

```
public static void main(String [] args){  
    Saving s1 = new Saving("SBI4686166", 10000);  
    Current c1 = new Current("SBIN584656",20000);  
    System.out.println("Savings Account");  
    s1.withdraw(3000);  
    s1.withdraw(3000);  
    System.out.println("S. A. after one year intrest "+  
s1.intrest());  
    System.out.println("Current Account");  
    c1.withdraw(8000);  
    c1.withdraw(12000);  
    System.out.println("S. A. after one year intrest "+ c1.intrest());  
}}
```

```
// interface
```

```
interface I1
```

```
{
```

```
public void methodA();
```

```
}
```

```
class A implements I1
{
    public void methodA()
    {
        System.out.println("Overridden method");
    }

    void methodB()
    {
        System.out.println("Its own method");
    }
}
```

Program for Interface

```
class TestInheritance1
{
    public static void main(String args[])
    {
        I1 obj=new A();

        //obj.methodA();
    }
}
```



```
l1 obj2;
```

```
obj2=new A();
```

```
//obj2.methodA();
```

```
//obj2.methodB(); //this is error
```

```
A obj3=new A();
```

```
//obj3.methodA();
```

```
//obj3.methodB();
```

```
l1 obj4;
```

```
A obj5 =new A();
```

```
obj4=obj5;
```

```
obj4.methodA();
```

```
}
```

```
}
```

```
//loan interface pgm
```

```
class PersonalLoan implements loan
```

```
{
```

```
double loanamount;
```

```
PersonalLoan(double loanamount)
```

```
{
```

```
this.loanamount=loanamount;
```

```
}
```

```
public void emiCalc()
```

```
{
```

```
double emi=(((loanamount*interest*5)/100)+loanamount)/60;
```

```
showemi(emi);
```

```
}
```

```
}
```

```
class HousingLoan implements loan
```

```
{
```

```
double loanamount;
```

```
HousingLoan(double loanamount)
```

```
{  
this.loanamount=loanamount;  
  
}  
public void emiCalc()  
{  
double emi=((loanamount*interest*10)/100)+loanamount*2)/120;  
showEmi(emi);  
}
```

```
class TestLoan  
{  
public static void main(String args[])  
{  
loan l1=new PersonalLoan(100000);  
l1.emiCalc();  
l1=new HousingLoan(1000000);  
l1.emiCalc();  
  
}  
}
```

Program for inheriting interfaces

```
interface Newspaper
```

```
{
```

```
public void news();
```

```
}
```

```
interface Magazine extends Newspaper
```

```
{
```

```
public void colorful();
```

```
}
```

```
class TestInterface implements Magazine
```

```
{
```

```
public void news()
```

```
{
```

```
System.out.println("it gives news");
```

```
}
```

```
public void colorful()
{
    System.out.println("it is colorful");
}

public static void main(String args[])
{
    TestInterface T1=new TestInterface();
    T1.news();
    T1.colorful();
}
}
```

Program for Interface (Achieves multiple inheritance)

```
interface Person
{
    public String PersonIdentity();
}
```

```
interface Location
```

```
{  
public String LocationIdentity();  
}
```

class Employee implements Person, Location

```
{  
String empname;  
String workplace;  
Employee()  
{  
empname="Ashok";  
workplace="CDAC";  
}  
public String PersonIdentity()  
{  
return empname;  
}  
public String LocationIdentity()  
{  
return "works in " + workplace;  
}
```

```
}
```

```
class Student implements Person, Location
```

```
{
```

```
String stuname;
```

```
String collegename;
```

```
Student()
```

```
{
```

```
stuname="John";
```

```
collegename="VIT";
```

```
}
```

```
public String PersonIdentity()
```

```
{
```

```
return stuname;
```

```
}
```

```
public String LocationIdentity()
```

```
{
```

```
return "Studies in "+collegename;
```

```
}
```

```
}
```

```
class TestInheritance
```

```
{
```

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

```
{
```

```
Student s=new Student();
```

```
System.out.println(s.PersonIdentity());
```

```
System.out.println(s.LocationIdentity());
```

```
Employee e=new Employee();
```

```
System.out.println(e.PersonIdentity());
```

```
System.out.println(e.LocationIdentity());
```

```
}
```

```
}
```

Program //interface with default method and static variables

```
interface interface1
```

```
{
```

```
void methodA();
```

```
void methodB();
```

```
default void display()
```



```
{  
  
System.out.println("This is non abstract method of interface");  
  
}  
  
static public final int incrementvalue=0;  
  
}
```

```
class classA implements interface1  
{  
  
static int incrementvalue1=0;  
  
int incrementvalue2;  
  
classA()  
{  
  
++incrementvalue1;  
  
++incrementvalue2;  
  
}  
  
public void methodA()  
{  
  
System.out.println("This is overridden methodA");  
  
}  
  
public void methodB()
```

```
{  
    System.out.println("This is overridden methodB");  
}  
}
```

```
class TestInterface3
```

```
{  
    public static void main(String args[])  
    {  
        interface1 obj=new classA();
```

```
        obj.methodA();
```

```
        obj.methodB();
```

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

```
        System.out.println("Interface static  
value"+interface1.incrementvalue);
```

```
        classA obj1=new classA();
```

```
        System.out.println("class static value "+classA.incrementvalue1);
```

```
        System.out.println("class non static value "+obj1.incrementvalue2);
```

```
        classA obj2=new classA();
```

```
System.out.println("class static value "+classA.incrementvalue1);  
System.out.println("class non static value "+obj2.incrementvalue2);  
}  
}
```

//practice with wrapper

```
class TestWrapper  
{  
    public static void main(String args[])  
    {  
        Integer int1=new Integer(100);  
        Double double1=new Double(100.5);  
        Float float1=new Float(45.22);  
        Character c=new Character('C');  
        Boolean b=new Boolean(true);  
        System.out.println(int1+" "+double1+" "+float1+" "+c+" "+b);  
  
        Integer int2=23;  
        int var1=34;  
        Integer int3=var1;  
        System.out.println(int2);  
        System.out.println(int3);
```

```
var1=int2;  
  
System.out.println("primitive "+var1);
```

```
String s="101";  
  
Integer int4=Integer.parseInt(s);  
  
int4=int4*10;  
  
System.out.println(int4);
```

```
s="23";  
  
Byte byte2=Byte.parseByte(s);  
  
//int4=int4*10;  
  
System.out.println(byte2);
```

```
byte byte3=Byte.parseByte(s);  
  
System.out.println(byte3);
```

```
byte byte4=12;  
  
Byte byte5=Byte.valueOf(byte4);  
  
System.out.println(byte5);
```

```
Integer int5=1556;  
  
byte b1=int5.byteValue();
```

```
System.out.println("byte"+b1);
```

```
int i2=int5.intValue();
```

```
System.out.println(i2);
```

```
double d=int5.doubleValue();
```

```
System.out.println(d);
```

```
}
```

```
}
```

```
//Program for autoboxing- unboxing
```

```
class TestBoxing
```

```
{
```

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

```
{
```

```
byte a=1;
```

```
Byte byteobj =new Byte(a); //primitive to Wrapper object -  
autoboxing
```

```
Integer intobj=new Integer(a); //autoboxing -boxing
```

```
Float floatobj=new Float(a); // widening conversion
```

```
byte b1=byteobj; //wrapper to primitive -unboxing (automatically )
```

```
//Byte b2=(Byte)intobj; // narrowing conversion
```

```
//Integer i3=(Integer) floatobj;
```

```
int a2=a;
```

```
int a1=78;
```

```
byte b=(byte) a1; // explicit conversion
```

```
Character ch='a';
```

```
char a4=ch;
```

```
}
```

```
}
```

```
public class Wrapping {
```

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

```
{
```

```
int a = 50; // Primitive data type value.
```

```
Integer i = Integer.valueOf(a); // Here, we are converting int into Integer explicitly.
```

```
Integer j = a; // Here, Autoboxing is happening. Java compiler will write  
Integer.valueOf(a) internally.
```

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

```

    }
}

public class Unwrapping {

    public static void main(String[] args)

    {

        // For converting Integer to int, create an object of Integer class and pass the value to
        its constructor.

        Integer a = new Integer(50);


        int i = a.intValue();// Here, we are converting Integer to int explicitly.

        int j = a; // Unboxing is happening. Java compiler will write a.intValue() internally.


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

    }

}

public class WrappingUnwrapping {

    public static void main(String[] args)

    {

        char ch = 'a'; // char data type.

        Character chobj = new Character(ch); // Wrapping char type value into Character
        object.


        byte a = 10; // byte data type value.

        Byte byteobj = new Byte(a); // Wrapping byte type value into Byte object.

```

`int b = 20; // int type value.`

`Integer intobj = new Integer(b); // Wrapping int type value into Integer object.`

`float c = 18.6f; // float type value.`

`Float floatobj = new Float(c); // Wrapping float type value into Float object.`

`double d = 250.5; // double data type value.`

`Double doubleobj = new Double(d); // Wrapping double data type value into Double object.`

`// Displaying the values from wrapper class objects.`

`System.out.println("Displaying values of Wrapper class objects:");`

`System.out.println("Character object: " + chrobj);`

`System.out.println("Byte object: " + byteobj);`

`System.out.println("Integer object: " + intobj);`

`System.out.println("Float object: " + floatobj);`

`System.out.println("Double object: " + doubleobj);`

`System.out.println("\n");`

`// Retrieving primitive data type values from objects.`

`// Unwrapping objects to primitive data type values.`


```
char chr = chrobj;

byte by = byteobj;

int in = intobj;

float fl = floatobj;

double db = doubleobj;


// Displaying the values of data types.

System.out.println("Displaying unwrapped values: ");


System.out.println("char value: " + chr);
System.out.println("byte value: " + by);


System.out.println("int value: " + in);
System.out.println("float value: " + fl);
System.out.println("double value: " + db);
}
}
```

Program for Packages

Emp.java

```
package emppkg;

public class Emp
```

```
{  
  
private int id;  
  
private String name;  
  
public String Quali;  
  
protected int exp;  
  
public Emp()  
{  
  
id=101;  
  
name="shubham";  
  
exp=10;  
  
}  
  
public void display()  
{  
  
System.out.println(id+ " "+name);  
  
}  
  
}
```

Salary.java

```
package emppkg;  
  
public class Salary  
{
```

```
double salary;

public Salary()

{

salary=200000;

}

public void display()

{

System.out.println(salary);

}

}
```

Leave.java

```
package emppkg;

public class Leave extends Emp

{

public void displayleave()

{

if (exp<=10)

System.out.println("leave is not permitted");

}

}
```

Setting classpath

export CLASSPATH=../dependency/ -for linux

set CLASSPATH=.;d:\java\packagetesttoday\ - for windows

javac -d d:\java\pkgtest\pkgclasses Emp.java

TestEmp.java

```
import emppkg.*;
```

```
class TestEmp
```

```
{
```

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

```
{
```

```
Emp e1=new Emp();
```

```
Salary s1=new Salary();
```

```
e1.display();
```

```
s1.display();
```

```
Leave l1=new Leave();
```

```
l1.displayleave();
```

```
Bonus b1=new Bonus();
```

```
b1.display();
```

```
e1.Qual="BTech";
```

```
Bonus1 b2=new Bonus1();
```

```
b2.display();
```

```
}
```

```
}
```

```
class Bonus extends Emp
```

```
{
```

```
public void display()
```

```
{
```

```
System.out.println(exp);
```

```
}
```

```
}
```

```
class Bonus1
```

```
{
```

```
public void display()
```

```
{
```

```
Emp e2=new Emp();
```

```
//System.out.println(e2.exp);
```

```
System.out.println(e2.Qualifi);
```

```
//System.out.println(exp);
```

```
}
```

```
}
```

Program for exception

```
class ExceptionEx
```

```
{
```

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

```
{
```

```
try
```

```
{int arr[]={12,34,54,34};
```

```
//System.out.println(arr[9]);
```

```
//int num=100/0;
```

```
//String year="hello";
```

```
//int yr=Integer.parseInt(year);
```

```
throw new Exception("explicit thrown");
```

```
}
```

```
catch(ArrayIndexOutOfBoundsException ex1)
```

```
{
```

```
System.out.println("Error in Array index");
}
catch(NumberFormatException ex1)
{
System.out.println("Number format is wrong");
}
catch(ArithmeticException ex1)
{
System.out.println("Divide by Zero");
}
catch(Exception ex1)
{
System.out.println("thrown exception");
System.out.println(ex1.getMessage());
}

finally
{
System.out.println("Try is done");
}
```

```
}
```

```
}
```

Program for User Exception

```
class MyException extends Exception
```

```
{
```

```
String s;
```

```
MyException(String str)
```

```
{
```

```
s=str;
```

```
}
```

```
public String toString()
```

```
{
```

```
return(s);
```

```
}
```

```
}
```

```
class TestCustomExcption
```

```
{
```

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

```
{
```



```
try
{
    //get a number
    //check a condition with that number mark<40
    throw new MyException("My own exception");
    //else display some msg (pass)
}
catch(MyException ex)
{
    System.out.println(ex);
}
//add a finally block
}
}
//user defined another example
```

```
class UserException extends Exception
{
    UserException(String s)
    {
        super(s);
    }
}
```

```
}
```

```
}
```

```
class TestUserException
```

```
{
```

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

```
{
```

```
String eligibility="BE"; //get user input
```

```
try
```

```
{
```

```
if (!eligibility.equals("BTECH"))
```

```
throw new UserException("Eligibility doesnt match");
```

```
}
```

```
catch(UserException ue)
```

```
{
```

```
System.out.println(ue.getMessage());
```

```
}
```

```
}
```

```
}
```

```
//check exception pgm
```

```

import java.io.*;

class TestCheckedException
{

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

{

Class c1=Class.forName("TestCheckedException");

FileInputStream fs=new FileInputStream("a.txt");

}

}

```

```

import java.util.ArrayList;
public class CustomExceptions
{ public static void main(String[] args)
{
ArrayList<String> arrayList = new ArrayList<>();
arrayList.add("Monday");
arrayList.add("Tuesday");
arrayList.add("Wednesday");
String day = "Sunday";

if (!arrayList.contains(day))
{ try
{ throw new DayNotAvailableException("Day not available",day); }
catch (DayNotAvailableException e)
{ e.getLocalizedMessage();
e.printStackTrace(); } } }

class DayNotAvailableException extends RuntimeException

```

```

{
    private String day;
    public DayNotAvailableException()
    { super(); }

    public DayNotAvailableException(String message, String day)
    { super(message); this.day = day; }

    public DayNotAvailableException(String message, String day, Throwable cause)
    { super(message, cause); this.day = day; }

    @Override
    public String toString()
    { return super.toString(); }

    @Override
    public String getMessage() { return super.getMessage() + " for the day :" + day; }

    @Override
    public String getLocalizedMessage()
    { return "The day "+day + " is not available."; } }
----

```

//User defined checked exception

```

public class EmployeeNotFoundException extends Exception {

    private static final long serialVersionUID =
-2872694086602732648L;

    private int id;

    EmployeeNotFoundException(int i, String message) {

        super (message) ;

        this.id = i;

    }
}

```

```
    EmployeeNotFoundException(int i, String message, String
cause) {

        super(message, new Throwable(cause));

        this.id = i;

    }
```

```
    @Override

    public String toString() {

        return
String.format("EmployeeNotFoundException[%d]", this.id);

    }}
}
```

```
public class TestException2

{

    public static void main(String args[])

    {

        int id=1001;

        try

        {

            if (id!=1000)
```

```
                throw new EmployeeNotFoundException(id,"Employee may not be
available", "resigned");
```

```
        }

        catch (EmployeeNotFoundException exp)

        {

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

            exp.printStackTrace();

        }

    }

}
```

```
}
```

```
}
```

Program for IO classes

```
import java.io.*;

class TestInput
{
    public static void main(String args[]) //throws IOException
    {
        try{
            BufferedReader br=new BufferedReader(new
            InputStreamReader(System.in));

            char c=(char)br.read();

            System.out.println(c);

            //String s=br.readLine();

            //System.out.println(s);
        }
    }
}
```

```
catch(IOException ie)
```

```
{
```

```
ie.printStackTrace();
```

```
}
```

```
}
```

```
}
```

```
//Input stream
```

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

```
class TestInputStream
```

```
{
```

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

```
{
```

```
InputStream input=new
```

```
FileInputStream("/home/boss/Documents/javasep22/file1.txt");
```

```
System.out.println(input.available());
```

```
byte[] array=new byte[100];
```

```
input.read(array);
```

```
String data=new String(array);
```

```
System.out.println(data);
```

```
input.close();
```

```
}
```

```
}
```

```
//pgm2
```

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

```
class TestInputStream1
```

```
{
```

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

```
{
```

```
InputStream input=new
```

```
FileInputStream("/home/boss/Documents/javasep22/file1.txt");
```

```
System.out.println(input.available());
```

```
int i=input.read();
```

```
while(i!=-1)
```

```
{
```

```
System.out.print((char)i);
```

```
i=input.read();
```

```
}
```

```
input.close();
```

```
}
```

```
}
```



```
import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class CopyBytes {

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

        FileInputStream in = null;

        FileOutputStream out = null;

        try {

            in = new FileInputStream("xanadu.txt");

            out = new FileOutputStream("outagain.txt");

            int c;

            while ((c = in.read()) != -1) {

                out.write(c);

            }

        } finally {

            if (in != null) {

                in.close();

            }

            if (out != null) {
```

```

        out.close();
    }
}
}
}

import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyCharacters {

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

        FileReader inputStream = null;

        FileWriter outputStream=null;

        try {

            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");

            int c;

            while ((c = inputStream.read()) != -1) {

                outputStream.write(c);

            }

        } finally {

```

```

        if (inputStream != null) {
            inputStream.close();
        }

        if (outputStream != null) {
            outputStream.close();
        }
    }
}

import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.IOException;

public class CopyLines {

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

        BufferedReader inputStream = null;

        PrintWriter outputStream = null;

        try {

            inputStream = new BufferedReader(new FileReader("xanadu.txt"));

            outputStream = new PrintWriter(new FileWriter("characteroutput.txt"));

```

```

String l;

while ((l = inputStream.readLine()) != null) {

    outputStream.println(l);

}

} finally {

    if (inputStream != null) {

        inputStream.close();

    }

    if (outputStream != null) {

        outputStream.close();

    }

}

}

}

```

```

//ByteArrayStream

import java.io.*;

class TestByteStream

{

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

    {

        byte[] array={1,2,3,4,5};

        ByteArrayInputStream input= new ByteArrayInputStream(array);
    }
}

```

```
System.out.println(input.available());

for (int i=0;i<array.length;i++)
{
    byte data=(byte)input.read();
    System.out.println(data+" ");
}

input.close();

}

}

//ByteOutput
import java.io.*;

class TestByteOutputStream
{
    public static void main(String args[]) throws IOException
    {
        String str="Welcome to Chennai";

        ByteArrayOutputStream out= new ByteArrayOutputStream();

        byte[] array=str.getBytes();

        out.write(array);

        String s=out.toString();

        System.out.println(s);
    }
}
```

```
import java.io.*;

import java.util.Scanner;


public class ScanXan {

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

        Scanner s = null;

        try {

            s = new Scanner(new BufferedReader(new FileReader("xanadu.txt")));

            while (s.hasNext()) {

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

            }

        } finally {

            if (s != null) {

                s.close();

            }

        }

    }

}
```

```
public class Root2 {  
  
    public static void main(String[] args) {  
  
        int i = 2;  
  
        double r = Math.sqrt(i);  
  
        System.out.format("The square root of %d is %f.%n", i, r);  
  
    }  
}
```

Program for NIO Classes

```
import java.io.IOException;  
  
import java.io.RandomAccessFile;  
  
import java.nio.ByteBuffer;  
  
import java.nio.channels.FileChannel;  
  
  
public class ChannelDemo {  
  
    public static void main(String args[]) throws IOException {  
  
        RandomAccessFile file = new  
RandomAccessFile("/home/boss/Documents/temp.txt", "r");  
  
        FileChannel fileChannel = file.getChannel();  
  
        ByteBuffer byteBuffer = ByteBuffer.allocate(512);  
  
        while (fileChannel.read(byteBuffer) > 0) {
```

```

        // flip the buffer to prepare for get operation
        byteBuffer.flip();
        while (byteBuffer.hasRemaining()) {
            System.out.print((char) byteBuffer.get());
        }
    }
    file.close();
}

//String Tokenizer
import java.util.*;

class StringTokenEx
{
    public static void main(String args[]) throws IOException
    {
        String str="this, is , an,example, for, string,tokenizer";
        StringTokenizer st=new StringTokenizer(str,"");

        while (st.hasMoreTokens())
            System.out.println(st.nextToken());
    }
}

```



```
}
```

Program for Collections

```
import java.util.*;

class TestCollection

{

public static void main(String args[])

{

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

intlist.add(100);

intlist.add(200);

System.out.println(intlist);

System.out.println(intlist.get(1));

for (Integer i : intlist) //for each

System.out.println(i);

Iterator itr=intlist.iterator();

while(itr.hasNext()) //iterating

{

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

}

}
```

```
}
```

```
//conversion
```

```
import java.util.*;
```

```
class ListToArray
```

```
{
```

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

```
{
```

```
ArrayList<String> subjects=new ArrayList();
```

```
subjects.add("Physics");
```

```
subjects.add("Physics");
```

```
subjects.add("Physics");
```

```
subjects.add("Physics");
```

```
String sub[]=subjects.toArray(new String[5]);
```

```
//String str[]=a1.toArray(new String[subjects.size()]);
```

```
for (String s:sub)
```

```
System.out.println(s);
```

```
}
```

```
}
```

```
//conversion
```

```
import java.util.*;
```

```
class ArraytoList
```

```
{
```

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

```
{
```

```
Integer[] num={23,45,562,23,45};
```

```
List<Integer> list=Arrays.asList(num);
```

```
System.out.println(list.get(1));
```

```
System.out.println(list);
```

```
ArrayList<Integer> a1=new ArrayList<Integer>();
```

```
Collections.addAll(a1,num);
```

```
System.out.println(a1);
```

```
}
```

```
}
```

```
// Program for Queue collection
```

```
import java.util.Queue;

import java.util.LinkedList;

class TestQueue {

    public static void main(String[] args) {

        // Creating Queue using the LinkedList class
        Queue<Integer> numbers = new LinkedList<>();

        // offer elements to the Queue
        numbers.offer(1);
        numbers.offer(2);
        numbers.offer(3);

        System.out.println("Queue: " + numbers);

        // Access elements of the Queue
        int accessedNumber = numbers.peek();

        System.out.println("Accessed Element: " + accessedNumber);

        System.out.println("Updated Queue: " + numbers);

        // Remove elements from the Queue
        int removedNumber = numbers.poll();
```

```
        System.out.println("Removed Element: " + removedNumber);

        System.out.println("Updated Queue: " + numbers);
    }
}
```

Program for collection with user defined class objects

```
import java.util.*;

class Student
{
    int id;

    String name;

    Student(int id,String name)
    {
        this.id=id;

        this.name=name;
    }

    public String toString()
    {
        return(id+" "+name);
    }
}
```

```
}
```

```
}
```

```
class TestCollection1
```

```
{
```

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

```
{
```

```
ArrayList<Student> stlist=new ArrayList<Student>();
```

```
Student s1=new Student(101,"Abhi");
```

```
Student s2=new Student(102,"Neha");
```

```
Student s3=new Student(103,"Tushar");
```

```
stlist.add(s1);
```

```
stlist.add(s2);
```

```
stlist.add(s3);
```

```
Iterator itr=stlist.iterator();
```

```
while(itr.hasNext()) //iterating
```

```
{
```

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

```
}
```

```
for (Student s:stlist)
```

```
System.out.println(s);
```

```
}
```

```
}
```

//Comparator interface, overriding compare method

```
import java.util.*;
```

```
class TestObjectCollection
```

```
{
```

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

```
{
```

```
ArrayList<Student> a1=new ArrayList<Student>();
```

```
a1.add(new Student(101,"Balaji"));
```

```
a1.add(new Student(105,"Abhishek"));
```

```
a1.add(new Student(104,"Suhas"));
```

```
a1.add(new Student(102,"Nikhil"));
```

```
System.out.println(a1);
```

```
Collections.sort(a1,Student.sturollnoComp);
```

```
System.out.println(a1);
```

```
Collections.sort(a1,Student.stunameComp);
```

```
System.out.println(a1);
```

```
}
```

```
}
```

```
class Student
```

```
{
```

```
int rollno;
```

```
String name;
```

```
Student(int rollno,String name)
```

```
{
```

```
this.rollno=rollno;
```

```
this.name=name;
```

```
}
```

```
int getRollno()
```

```
{
```

```
return rollno;
```

```
}
```

```
String getName()
```

```
{
```



```
return name;

}

public String toString()

{

return rollno+" "+name;

}
```

@Override

```
public static Comparator<Student> sturollnoComp=new
Comparator<Student>(){

public int compare(Student s1,Student s2) {

int rno1=s1.getRollno();

int rno2=s2.getRollno();


return rno1-rno2;

}

};
```

```
public static Comparator<Student> stunameComp=new
Comparator<Student>(){

public int compare(Student s1,Student s2) {

String name1=s1.getName();
```

```
String name2=s2.getName();
```

```
return name1.compareTo(name2);
```

```
}
```

```
};
```

```
}
```

```
// copying, reversing, merging
```

```
import java.util.*;
```

```
public class TestJavaCollection{
```

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

```
ArrayList<Integer> list2=new ArrayList<>();
```

```
list2.add(100);
```

```
list2.add(300);
```

```
list2.add(400);
```

```
//list2.add(1500);
```

```
//list2.add(2500);
```

```
Collections.reverse(list2);
```

```
System.out.println(list2);
```

```
LinkedList<Integer> list1=new LinkedList<>();
```

```
list1.add(1000);
```

```
list1.add(3000);
```

```
list1.add(4000);
```

```
list1.add(15000);
```

```
Iterator i1=list1.descendingIterator();
```

```
while(i1.hasNext())
```

```
System.out.println(i1.next());
```

```
System.out.println(list1);
```

```
ArrayList<Integer> list3=new ArrayList<>(10);
```

```
//list3.ensureCapacity(15);
```

```
Collections.copy(list1,list2);
```

```
System.out.println(list1);
```

```
list3.addAll(list1);
```

```
list3.addAll(list2);
```

```
System.out.println(list3);
```

```
}
```

```
}
```

// Hashmap and tree map

```
import java.util.*;

public class TestHashMap{

    public static void main(String[] args) {

        /*HashMap<Integer,String> h1=new HashMap<>();

        h1.put(12,"Red");

        h1.put(11,"Green");

        h1.put(15,"Orange");

        h1.put(13,"Yellow");

        h1.put(15,"Orange");

        System.out.println(h1);

        Map<String,Integer> h2=new TreeMap<>();

        h2.put("Red",12);

        h2.put("Green",11);

        h2.put("Orange",15);

        h2.put("Yellow",13);

        h2.put("Orange",15);

        System.out.println(h2);*/

        Map<String,Integer> h3=new HashMap<>();
```

```
h3.put("Red",12);
h3.put("Green",11);
h3.put("Orange",15);
h3.put("Yellow",13);
h3.put("Orange",15);
System.out.println(h3);

Map<String,Integer> h4=new TreeMap<>(h3);
System.out.println(h4);

}

}
```

Program with Map Collection

Pgm1

```
import java.util.*;

public class TestHashMapfun {
```

```
public static void main(String args[]) {  
  
    // Creating a HashMap of int keys and String values  
  
    HashMap<Integer, String> hashmap = new HashMap<Integer,  
String>();  
  
    // Adding Key and Value pairs to HashMap  
  
    hashmap.put(22,"A");  
    hashmap.put(55,"B");  
    hashmap.put(33,"Z");  
    hashmap.put(44,"M");  
    hashmap.put(99,"I");  
    hashmap.put(88,"X");  
  
    System.out.println(hashmap.keySet());  
    System.out.println(hashmap.values());  
    System.out.println(hashmap.entrySet());  
    }  
    }  
  
Pgm2  
  
import java.util.*;
```

```
public class HashMapSortByKeyExample {  
    public static void main(String args[]) {  
  
        // Creating a HashMap of int keys and String values  
        HashMap<Integer, String> hashmap = new HashMap<Integer,  
String>();  
  
        // Adding Key and Value pairs to HashMap  
        hashmap.put(22,"A");  
        hashmap.put(55,"B");  
        hashmap.put(33,"Z");  
        hashmap.put(44,"M");  
        hashmap.put(99,"I");  
        hashmap.put(88,"X");  
  
        System.out.println(hashmap.get(99));  
  
        System.out.println("Before Sorting:");  
        System.out.println(hashmap);  
  
        Set set = hashmap.entrySet();
```

```
Iterator iterator = set.iterator();

while(iterator.hasNext()) {

    Map.Entry pair = (Map.Entry)iterator.next();

    System.out.print(pair.getKey() + ": ");

    System.out.println(pair.getValue());

}
```

```
Map<Integer, String> map = new TreeMap<Integer,
String>(hashmap);  System.out.println("After Sorting:");

    System.out.println(map);
```

```
Set set2 = map.entrySet();

Iterator iterator2 = set2.iterator();

while(iterator2.hasNext()) {

    Map.Entry pair = (Map.Entry)iterator2.next();

    System.out.print(pair.getKey() + ": ");

    System.out.println(pair.getValue());

}
```



```
}
```

```
}
```

```
//pgm3
```

```
import java.util.*;
```

```
class Student
```

```
{
```

```
int id;
```

```
String name;
```

```
String address;
```

```
Student(int id,String name,String address)
```

```
{
```

```
this.id=id;
```

```
this.name=name;
```

```
this.address=address;
```

```
}
```

```
public int getId()
```

```
{
```

```
return id;
```

```
}
```

```
public String toString()
```

```
{
```

```
return(id+name+address);  
  
}  
  
}
```

```
class TestLinkedhashmap  
{  
  
    public static void main(String args[])  
    {  
  
        LinkedHashMap<Integer,Student> map1=new  
        LinkedHashMap<Integer,Student>();  
  
        Student s1=new Student(1,"Aloki","Chennai");  
        Student s2=new Student(2,"Krishnan","Indore");  
        Student s3=new Student(3,"Santhosh","Delhi");  
  
        map1.put(1,s1);  
        map1.put(2,s2);  
        map1.put(3,s3);  
  
  
        Student s4=new Student(4,"Rameshi","TVM");  
        map1.put(s4.getId(),s4);  
        System.out.println(map1);  
  
        Set set2 = map1.entrySet();
```

```
Iterator iterator2 = set2.iterator();

while(iterator2.hasNext()) {

    Map.Entry pair = (Map.Entry)iterator2.next();

    System.out.print(pair.getKey() + ": ");

    System.out.println(pair.getValue());

    Student tmp=(Student) pair.getValue();

    System.out.println(tmp.name);

}

}
```

//Program for random number generation

```
import java.util.*;

class TestRandom

{

public static void main(String args[])

{

Random rand=new Random();

for (int i=1;i<100;i++)

System.out.println(rand.nextInt(1000));

}
```

```
for (int i=1;i<100;i++)  
  
System.out.println(rand.nextInt(0+10)-10); //nextInt(max-min)+min  
-10 to 0  
  
}  
  
}
```

//Map with objects

```
import java.util.*;  
  
class TestHashMap  
{  
  
public static void main(String args[])  
{  
  
Map<Integer,String> newmap=new HashMap<>();  
  
newmap.put(1,"Rose");  
  
newmap.put(2,"Jasmine");  
  
newmap.put(3,"Lotus");  
  
System.out.println(newmap);  
  
System.out.println(newmap.keySet());  
  
System.out.println(newmap.values());  
  
System.out.println(newmap.entrySet());  
  
System.out.println(newmap.get(3));  
  
}
```

```
System.out.println(newmap.remove(3));
```

```
System.out.println(newmap.entrySet());
```

```
A a1=new A(10,"Chennai","South");
```

```
A a2=new A(20,"Mumbai","West");
```

```
A a3=new A(30,"Delhi","North");
```

```
Map<Integer,A> objmap=new HashMap();
```

```
objmap.put(a1.k,a1);
```

```
objmap.put(a2.k,a2);
```

```
objmap.put(a3.k,a3);
```

```
System.out.println(objmap);
```

```
}
```

```
}
```

```
class A
```

```
{
```

```
int k;
```

```
String n;
```

```
String n1;
```

```
A(int k,String n,String n1)
```

```
{  
  
this.k=k;  
  
this.n=n;  
  
this.n1=n1;  
  
}  
  
public String toString()  
  
{  
  
return(" key is " + k + " record is " +n + " "+n1);  
  
}  
  
}
```

//Program with comparable interface

```
import java.util.*;  
  
class TestComparable  
  
{  
  
public static void main(String args[])  
  
{  
  
ArrayList<clsA> arrlist=new ArrayList<clsA>();  
arrlist.add(new clsA(10,20,"chennai",19.2F));  
arrlist.add(new clsA(20,40,"delhi",15.2F));  
arrlist.add(new clsA(30,30,"pune",14.2F));
```

```
arrlist.add(new clsA(30,15,"kolkata",12.2F));
```

```
Collections.sort(arrlist); //compareTo() -Comparable
```

```
System.out.println(arrlist);
```

```
}
```

```
}
```

```
class clsA implements Comparable<clsA>
```

```
{
```

```
int s;
```

```
int s1;
```

```
String s2;
```

```
float s3;
```

```
clsA(int s,int s1,String s2,float s3)
```

```
{
```

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

```
this.s1=s1;
```

```
this.s2=s2;
```

```
this.s3=s3;
```

```
}
```

```
public int getS1()
```

```
{
```

```
    return s1;
```

```
}
```

```
public String toString()
```

```
{
```

```
    return("s="+s +" s1="+s1+" s2="+s2+" s3="+s3);
```

```
}
```

```
public int compareTo(clsA cl1)
```

```
{
```

```
    int temp=cl1.getS1();
```

```
    return this.s1-temp;
```

```
}
```

```
}
```

```
//Array to List
```

```
import java.util.*;
```



```
class CopyArraytoList
{
    public static void main(String args[])
    {
        Integer arr[]=new Integer[10];

        int i;
        for( i=0;i<arr.length;i++)
            arr[i]=i;

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

        for( i=0;i<arr.length;i++)
        {
            arrlist.add(i,arr[i]);
        }

        System.out.println(arrlist);

    }
}
```

```
//List to Array

import java.util.*;

class CopyListtoArray

{

public static void main(String args[])

{

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


arrlist.add(10);

arrlist.add(20);

arrlist.add(30);

arrlist.add(40);


Integer arr[]=new Integer[arrlist.size()];

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

{

arr[i]=arrlist.get(i);

}

for (Integer i1:arr)
```

```
System.out.println(i1);  
  
}  
  
}
```

Program for Generic Class

```
class Main {  
  
    public static void main(String[] args) {  
  
        GenericsClass<Integer> intObj = new GenericsClass<>(5);  
  
        System.out.println("Generic Class returns: " + intObj.getData());  
  
        GenericsClass<String> stringObj = new GenericsClass<>("Java  
Programming");  
  
        System.out.println("Generic Class returns: " + stringObj.getData());  
  
    }  
  
}  
  
class GenericsClass<T> {  
  
    private T data;
```

```
public GenericsClass(T data) {  
    this.data = data;  
}
```

```
public T getData() {  
    return this.data;  
}  
}
```

Program for multithreading

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

Program with Runnable interface

```
class MultithreadingDemo implements Runnable{

    public void run(){

        System.out.println("My thread is in running state.");

    }

    public static void main(String args[]){

        MultithreadingDemo obj=new MultithreadingDemo();

        Thread tobj =new Thread(obj);

        tobj.start();

    }

}
```

Program for Multithreading

```
class Count extends Thread

{

    Count()

    {

        super("my extending thread");

        System.out.println("my thread created" + this);

        start();

    }

    public void run()

    {

        try

        {

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

```

        {
            System.out.println("Printing the count " + i);
            Thread.sleep(1000);
        }
    }

    catch (InterruptedException e)
    {
        System.out.println("my thread interrupted");
    }

    System.out.println("My thread run is over" );
}

}

class ExtendingExample
{
    public static void main(String args[])
    {
        Count cnt = new Count();

        try
        {
            while(cnt.isAlive())
            {
                System.out.println("Main thread will be alive till the
child thread is live");

                Thread.sleep(1500);
            }
        }

        catch (InterruptedException e)

```

```

        {
            System.out.println("Main thread interrupted");
        }

        System.out.println("Main thread's run is over" );
    }
}
}

```

Program for Prioirties

```

public class AThread implements Runnable
{
    public void run()
    {
        System.out.println(Thread.currentThread()); // This method is
        static.
    }

    public static void main(String[] args)
    {
        AThread a = new AThread();

        Thread t1 = new Thread(a, "First Thread");
        Thread t2 = new Thread(a, "Second Thread");
        Thread t3 = new Thread(a, "Third Thread");

        t1.setPriority(4); // Setting the priority of first thread.
        t2.setPriority(2); // Setting the priority of second thread.
        t3.setPriority(8); // Setting the priority of third thread.
    }
}

```

```
t1.start();  
t2.start();  
t3.start();  
}  
}
```

Program for thread priorities

```
class X implements Runnable  
{  
public void run()  
{  
    System.out.println("Thread X started");  
    for(int i = 1; i<=4; i++)  
    {  
        System.out.println("Thread X: " +i);  
    }  
    System.out.println("Exit from X");  
}  
}  
  
class Y implements Runnable  
{  
public void run()  
{  
    System.out.println("Thread Y started");  
    for(int j = 0; j <= 4; j++)  
    {  
        System.out.println("Thread Y: " +j);  
    }  
}
```



```

        System.out.println("Exit from Y");
    }
}

class Z implements Runnable
{
    public void run()
    {
        System.out.println("Thread Z started");
        for(int k = 0; k <= 4; k++)
        {
            System.out.println("Thread Z: " +k);
        }
        System.out.println("Exit from Z");
    }
}

public class ThreadPriority1 {
    public static void main(String[] args)
    {
        X x = new X();
        Y y = new Y ();
        Z z = new Z ();

        Thread t1 = new Thread(x);
        Thread t2 = new Thread(y);
        Thread t3 = new Thread(z);

        t1.setPriority(Thread.MAX_PRIORITY);
    }
}

```

```
t2.setPriority(t2.getPriority() + 4);  
t3.setPriority(Thread.MIN_PRIORITY);  
  
t1.start();  
t2.start();  
t3.start();  
  
}  
}
```

Program with Synchronized

```
public class Synchronization implements Runnable  
{  
    int tickets = 3;  
    static int i = 1, j = 2, k = 3;  
    synchronized void bookticket (String name, int wantedtickets)  
    {  
        if (wantedtickets <= tickets)  
        {  
            System.out.println (wantedtickets + " booked to " +  
name);  
            tickets = tickets - wantedtickets;  
        }  
        else  
        {  
            System.out.println ("No tickets to book");  
        }  
    }  
}
```

```

}

public void run ()
{
    String name = Thread.currentThread ().getName ();

    if (name.equals ("t1"))
    {
        bookticket (name, i);
    }
    else if (name.equals ("t2"))
    {
        bookticket (name, j);
    }
    else
    {
        bookticket (name, k);
    }
}

public static void main (String[]args)
{
    Synchronization s = new Synchronization ();

    Thread t1 = new Thread (s);
    Thread t2 = new Thread (s);
    Thread t3 = new Thread (s);

    t1.setName ("t1");
    t2.setName ("t2");
    t3.setName ("t3");

    t1.start ();

```

```
        t2.start ();

        t3.start ();

    }

}
```

Program inner classes

(Nested inner class)

```
class outer

{

    int data;

    class inner

    {

        public void display()

        {

            System.out.println("Inner class"+data);

        }

    }

}

class TestInner

{

    public static void main(String args[])

    {

        outer.inner obj=new outer().new inner();

        obj.display();

    }

}
```

Program local inner class

```
class outer

{

int data;

public void methodA()

{


class inner1

{

public void methodB()

{

System.out.println("inner class method");

}

}

inner1 obj2=new inner1();

obj2.methodB();


}

}

class TestInner1

{

public static void main(String args[])

{
```

```
outer obj2=new outer();  
obj2.methodA();  
}  
}
```

Program with variables access of outer class

```
class Outer  
{  
    int sum1=100;  
    void outerMethod()  
    {  
        int sum=10;  
        System.out.println("Outer class method");  
        class Inner  
        {  
            int i=200;  
            void innerMethod()  
            {  
  
                System.out.println("Inner class method"+sum+" "+sum1);  
            }  
        }  
  
        Inner in=new Inner();  
        in.innerMethod();  
        System.out.println(in.i);  
    }  
}
```

```
class MainClass
{
public static void main(String args[])
{
Outer out=new Outer();
out.outerMethod();
//System.out.println(out.i);
}
}
```

Program for Static nested class

```
import java.util.*;

// Class 1
// Outer class
class Outer {

    // Method
    private static void outerMethod()
    {

        // Print statement
        System.out.println("inside outerMethod");
    }

    // Class 2
```

```
// Static inner class

static class Inner {

    public static void display()

    {

        // Print statement

        System.out.println("inside inner class Method");

        // Calling method inside main() method

        outerMethod();

    }

}

// Class 3

// Main class

class GFG {

    // Main driver method

    public static void main(String args[])

    {

        // Calling method static display method rather than an
instance of that class.

        Outer.Inner.display();

    }

}
```



```
}
```

Program Anonymous class

(as a subclass)

```
class animal
```

```
{
```

```
public void sound()
```

```
{
```

```
System.out.println("makes sound");
```

```
}
```

```
}
```

```
class TestAnonymous
```

```
{
```

```
static animal a1=new animal()
```

```
{
```

```
public void sound()
```

```
{
```

```
super.sound();
```

```
System.out.println("inside anonymous");
```

```
}
```

```
};
```

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

```
{
```

```
a1.sound();
```

```
}  
  
}
```

Program Anonymous class (interface)

```
interface animal  
{  
    public void sound();  
}  
  
class TestAnonymous1  
{  
  
    static animal a1=new animal()  
    {  
        public void sound()  
        {  
            //super.sound();  
            System.out.println("inside anonymous");  
        }  
    };  
  
    public static void main(String args[])  
    {  
        a1.sound();  
    }  
}
```

```
}
```

Program for reflection

(Pgm1)

```
import java.lang.Class;
```

```
import java.lang.reflect.*;
```

```
class Animal {
```

```
}
```

```
// put this class in different Dog.java file
```

```
public class Dog extends Animal {
```

```
    public void display() {
```

```
        System.out.println("I am a dog.");
```

```
    }
```

```
}
```

```
// put this in Main.java file
```

```
class Main {
```

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

```
        try {
```

```
            // create an object of Dog
```

```
            Dog d1 = new Dog();
```

```
            // create an object of Class
```

```
            // using getClass()
```

```
            Class obj = d1.getClass();
```

```
            // get name of the class
```

```

String name = obj.getName();

System.out.println("Name: " + name);


// get the access modifier of the class
int modifier = obj.getModifiers();


// convert the access modifier to string
String mod = Modifier.toString(modifier);
System.out.println("Modifier: " + mod);


// get the superclass of Dog
Class superClass = obj.getSuperclass();

System.out.println("Superclass: " + superClass.getName());
}


catch (Exception e) {
    e.printStackTrace();
}
}
}

```

(Pgm2)

```

import java.lang.reflect.*;

class Test
{

```

```
private int data;

public Test()

{
data=10;

}

public Test(int a,int b)

{
data=a;

}

public void method1()

{
System.out.println("welcome");

}


public void method2()

{

}


}


class TestReflection

{

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

{

Test t1=new Test();
```

```
Class cls = t1.getClass();

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


//Constructor constructor = cls.getConstructor();

//      System.out.println("The name of constructor is " +
//
//                      constructor.getName());


//Constructor c1=cls.getConstructor();


Constructor[] constr=cls.getConstructors();


for (Constructor c:constr)
{
System.out.println(c.getName());
}


Method[] methodlist=cls.getMethods();


for (Method m1:methodlist)
System.out.println(m1.getName());


//System.out.println(c1.getName());


Field f1=cls.getDeclaredField("data");
f1.setAccessible(true);
f1.set(t1,100);
```

```
Method m1=cls.getDeclaredMethod("method1");  
  
m1.invoke(t1);  
  
}  
  
}
```

(pgm3)

```
import java.lang.Class;  
import java.lang.reflect.*;
```

```
class Dog {  
    private String color;  
}
```

```
class Main {  
    public static void main(String[] args) {  
        try {  
            // create an object of Dog  
            Dog d1 = new Dog();  
  
            // create an object of Class  
            // using getClass()  
            Class obj = d1.getClass();  
  
            // access the private field color  
            Field field1 = obj.getDeclaredField("color");  
  
            // allow modification of the private field
```

```

        field1.setAccessible(true);

        // set the value of color
        field1.set(d1, "brown");

        // get the value of field color
        String colorValue = (String) field1.get(d1);
        System.out.println("Value: " + colorValue);

        // get the access modifier of color
        int mod2 = field1.getModifiers();

        // convert the access modifier to string
        String modifier2 = Modifier.toString(mod2);
        System.out.println("Modifier: " + modifier2);
    }

    catch (Exception e) {
        e.printStackTrace();
    }
}

(pgm4)
import java.lang.Class;
import java.lang.reflect.*;

```



```
class Dog {

    // public constructor without parameter
    public Dog() {

    }

    // private constructor with a single parameter
    private Dog(int age) {

    }

}

class Main {

    public static void main(String[] args) {

        try {

            // create an object of Dog

            Dog d1 = new Dog();

            // create an object of Class
            // using getClass()

            Class obj = d1.getClass();

            // get all constructors of Dog

            Constructor[] constructors = obj.getDeclaredConstructors();
```

```

        for (Constructor c : constructors) {

            // get the name of constructors

            System.out.println("Constructor Name: " + c.getName());

            // get the access modifier of constructors

            // convert it into string form

            int modifier = c.getModifiers();

            String mod = Modifier.toString(modifier);

            System.out.println("Modifier: " + mod);

            // get the number of parameters in constructors

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

            System.out.println("");

        }

    }

    catch (Exception e) {

        e.printStackTrace();

    }

}

```

Program for Deadlock

```

class A implements Runnable{

    public void run() {

```

```

        synchronized (String.class) {

            try {

                Thread.sleep(100);

            } catch (InterruptedException e)
            {e.printStackTrace();}

            System.out.println(Thread.currentThread().getName() + "has
            acquired lock "

                                + "on String class and waiting to
            acquire lock on Object class...");

            synchronized (Object.class) {

                System.out.println(Thread.currentThread().getName() +

                                " has acquired lock on
            Object class");

            }

        }

        System.out.println(Thread.currentThread().getName()+"
        has ENDED");

    }

}

class B extends Thread{

    public void run() {

```

```

        synchronized (Object.class) {

System.out.println(Thread.currentThread().getName() + " has
acquired "

                + "lock on Object class and waiting to
acquire lock on String class...");

        try {

                Thread.sleep(100);

        } catch (InterruptedException e)
{e.printStackTrace();}

        synchronized (String.class) {

System.out.println(Thread.currentThread().getName() +

                " has acquired lock on
String class");

        }

    }

    System.out.println(Thread.currentThread().getName()+ "
has ENDED");

    }

}

public class DeadlockCreation {

    public static void main(String[] args) {

        Thread thread1 = new Thread(new A(), "Thread-1");

        Thread thread2 = new Thread(new B(), "Thread-2");

```

```
        thread1.start();

        thread2.start();

    } }
```

Program Create multiple threads using lambda expressions.

```
class RunnableLambdaExample {

    public static void main(String[] args) {

        System.out.println(Thread.currentThread().getName() + ":
RunnableTest");

        // Anonymous Runnable

        Runnable task1 = new Runnable() {

            @Override

            public void run() {

                System.out.println(Thread.currentThread().getName() +
" is running");

            }

        };

        // Passing a Runnable when creating a new thread

        Thread thread2 = new Thread(new Runnable() {

            @Override

            public void run() {

                System.out.println(Thread.currentThread().getName() + " is
running");

            }

        });

    }

}
```

```

        }

    });

    // Lambda Runnable

    Runnable task3 = () -> {

        System.out.println(Thread.currentThread().getName() +
" is running");

    };

    Thread thread1 = new Thread(task1);

    thread1.start();

    thread2.start();

    new Thread(task3).start();

}

}

```

//Producer-consumer problem

```

import java.util.LinkedList;

public class Threadexample {

    public static void main(String[] args)

        throws InterruptedException

    {

```

```
// Object of a class that has both produce()  
// and consume() methods  
final PC pc = new PC();
```

```
// Create producer thread  
Thread t1 = new Thread(new Runnable() {  
    @Override  
    public void run()  
    {  
        try {  
            pc.produce();  
        }  
        catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
});
```

```
// Create consumer thread  
Thread t2 = new Thread(new Runnable() {  
    @Override  
    public void run()  
    {  
        try {  
            pc.consume();  
        }  
        catch (InterruptedException e) {
```

```

        e.printStackTrace();
    }
}

});

// Start both threads

t1.start();

t2.start();

// t1 finishes before t2

t1.join();

t2.join();
}

// This class has a list, producer (adds items to list
// and consumer (removes items).

public static class PC {

    // Create a list shared by producer and consumer
    // Size of list is 2.

    LinkedList<Integer> list = new LinkedList<>();

    int capacity = 2;

    // Function called by producer thread

    public void produce() throws InterruptedException
    {

        int value = 0;

```



```

while (true) {
    synchronized (this)
    {
        // producer thread waits while list
        // is full
        while (list.size() == capacity)
            wait();

        System.out.println("Producer produced-"
                           + value);

        // to insert the jobs in the list
        list.add(value++);

        // notifies the consumer thread that
        // now it can start consuming
        notify();

        // makes the working of program easier
        // to understand
        Thread.sleep(1000);
    }
}
}

```

```

// Function called by consumer thread
public void consume() throws InterruptedException

```

```
{
    while (true) {
        synchronized (this)
        {
            // consumer thread waits while list
            // is empty
            while (list.size() == 0)
                wait();

            // to retrieve the first job in the list
            int val = list.removeFirst();

            System.out.println("Consumer consumed-"
                               + val);

            // Wake up producer thread
            notify();

            // and sleep
            Thread.sleep(1000);
        }
    }
}
}
```

Program for Enumeration
class TestEnum1

```

{
public enum Season {WINTER,SPRING,SUMMER}
public static void main(String args[])
{
for (Season s: Season.values())
System.out.println(s);
System.out.println(Season.valueOf("WINTER"));
System.out.println(Season.valueOf("WINTER").ordinal());

System.out.println(Season.valueOf("SPRING").ordinal());
System.out.println(Season.valueOf("SUMMER").ordinal());
}

}

```

//lambda expressions

(pgm0)

@FunctionalInterface

interface I1

```

{
void simpleMethod();

}

```

class TestFunctional

```

{
public static void main(String args[])
{
I1 obj1= () -> System.out.println("Interface abstract method");

obj1.simpleMethod();

```

```

I1 obj2= () -> { int sum=100; System.out.println("Interface overriding "+sum);};

```

```

obj2.simpleMethod();
}
}

```

(pgm1)

@FunctionalInterface

interface Shape

```

{
public void draw();

```

```
}
```

```
class ShapeEx
```

```
{
```

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

```
    {
```

```
        Shape s1=new Shape() //Anonymous class
```

```
        {
```

```
            public void draw()
```

```
            { System.out.println("Circle is drawn using radius");
```

```
            }
```

```
        };
```

```
        s1.draw();
```

```
        Shape s2=()-> //lambda expression
```

```
        {
```

```
            System.out.println("Rectangle is drawn using length & width");
```

```
        };
```

```
        s2.draw();
```

```
        Shape s3=new Shape()
```

```
        {
```

```
            public void draw() { }
```

```
        };
```

```
        s3.draw();
```

```
    }
```

```
}
```

```
(pgm2)
```

```
interface myinterface
```

```
{
```

```
    int getvalue();
```

```
}
```

```
class TestFuncInterface
```

```
{
```

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

```
    {
```

```
        myinterface ref;
```

```
        ref=()->{return(100);};
```

```
        System.out.println(ref.getvalue());
```

```
}  
}
```

// lambda expressions

```
class TestLambda  
{  
    public static void main(String args[])  
    {  
        new Thread(new Runnable()  
        {  
            public void run(){  
                System.out.println("run method");  
            }  
        }).start();  
    }  
}
```

```
class TestLampdaThread  
{  
    public static void main(String args[])  
    {  
        Runnable R1 = () -> System.out.println("Run method");  
  
        Thread t1=new Thread(R1);  
        t1.start();  
  
    }  
}
```

//Lambda- string example
interface StringInt
{
 public String reverse(String str);
}

```
class TestLambda4  
{  
    public static void main(String args[])  
    {  
        StringInt obj=(str)-> {  
            String result="";  
            for (int i=str.length()-1;i>=0;i--)  
                result+=str.charAt(i);  
            return(result);  
        };  
  
        System.out.println(obj.reverse("CDAC"));
```

```
}  
}
```

(pgm-lambda with return type)

@FunctionalInterface

interface I1

```
{  
    int simpleMethod(int a, int b);  
  
}
```

class TestFunctional

```
{  
    public static void main(String args[])  
    {  
        int a=10,b=100;
```

```
I1 obj1= (var1,var2) -> var1+var2;
```

```
int sum=obj1.simpleMethod(a,b);  
System.out.println(sum);
```

```
I1 obj2= (var1,var2) -> { int sum1=100; return(var1*var2) ;};
```

```
System.out.println(obj2.simpleMethod(100,200));  
}  
}
```

//Synchronized methods

class Line

```
{  
  
    // if multiple threads(trains) trying to access  
    // this synchronized method on the same Object  
    // but only one thread will be able  
    // to execute it at a time.  
    synchronized public void getLine()  
    {  
        for (int i = 0; i < 3; i++)  
        {  
            System.out.println(i);  
            try  
            {  
                Thread.sleep(400);  
            }  
            catch (Exception e)
```

```

        {
            System.out.println(e);
        }
    }
}

```

```

class Train extends Thread
{
    // Reference variable of type Line.
    Line line;

    Train(Line line)
    {
        this.line = line;
    }

    @Override
    public void run()
    {
        line.getLine();
    }
}

```

```

class GFG
{
    public static void main(String[] args)
    {
        Line obj = new Line();

        // we are creating two threads which share
        // same Object.
        Train train1 = new Train(obj);
        Train train2 = new Train(obj);

        // both threads start executing .
        train1.start();
        train2.start();
    }
}

```

//sender-receiver problem

```

public class Data {

    private String packet;

    private boolean transfer = true;
}

```



```
public synchronized String receive() {  
    while (transfer) {  
        try {  
            wait();  
        } catch (InterruptedException e) {  
            Thread.currentThread().interrupt();  
            System.out.println("Thread Interrupted");  
        }  
    }  
    transfer = true;  
  
    String returnPacket = packet;  
    notifyAll();  
    return returnPacket;  
}
```

```
public synchronized void send(String packet) {  
    while (!transfer) {  
        try {  
            wait();  
        } catch (InterruptedException e) {  
            Thread.currentThread().interrupt();  
            System.out.println("Thread Interrupted");  
        }  
    }  
}
```

```
transfer = false;
```

```
this.packet = packet;
```

```
notifyAll();
```

```
}
```

```
}
```

//NIO Program

```
import java.nio.*;
```

```
import java.nio.channels.*;
```

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

```
class TestNio
```

```
{
```

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

```
{
```

```
FileInputStream input = new FileInputStream  
("/home/boss/Documents/javasep22/Testin.txt"); // Path of Input text file
```

```
ReadableByteChannel source = input.getChannel();
```

```
FileOutputStream output = new FileOutputStream  
("/home/boss/Documents/javasep22/Testout.txt"); //Path of Output text file
```

```
WritableByteChannel destination = output.getChannel();
```

```
copyData(source, destination);
```

```
source.close();
```

```
destination.close();
```

```
}
```

```
static void copyData(ReadableByteChannel src, WritableByteChannel dest) throws  
IOException
```

```

{
ByteBuffer buffer = ByteBuffer.allocateDirect(20 * 1024);

while (src.read(buffer) != -1)
{
// The buffer is used to drained

buffer.flip();

// keep sure that buffer was fully drained

while (buffer.hasRemaining())
{

dest.write(buffer);

}

buffer.clear(); // Now the buffer is empty, ready for the filling

}

}
}

```

// Association

//Aggregation

class TestAssoication

```

{
public static void main(String args[])
{
Student s1=new Student(101);
Student s2=new Student(102);

Student s3=new Student(103);
Student stu[]={s1,s2,s3};
Course c=new Course("BSc",stu);
System.out.println(c.getCourseName());

Student[] sarray=c.getStudents();
for (Student s:sarray)
{

```

```

System.out.println(s.getId());
}

}
}
class Course
{
String coursename;
Student[] stu;
Course (String coursename, Student[] stu)
{
this.coursename=coursename;
this.stu=stu;
}
String getCourseName()
{
return coursename;
}
Student[] getStudents()
{
return stu;
}

}

class Student
{
int id;
Student(int id)
{
this.id=id;
}
int getId()
{
return id;
}
}

```

//compostion

```

class Engine
{
String model;

Engine(String model)
{
this.model=model;
}
String getModel()
{

```

```
return model;  
}
```

```
}
```

```
class Car  
{  
String carname;  
Engine engine;  
Car(String carname)  
{  
this.carname=carname;  
}  
void addEngine()  
{  
engine=new Engine("VXI");  
}  
void print()  
{  
System.out.println(carname);  
System.out.println(engine.getModel());  
  
}  
}
```

```
class TestComposition  
{  
public static void main(String args[])  
{  
Car c=new Car("WagonR");  
c.addEngine();  
c.print();  
}  
}
```

//Functional interface

```
import java.util.*;  
interface Person //Functional interface  
{  
public void show(int id,int age);  
}
```

```
class TestLamda1  
{  
public static void main(String args[])  
{  
Scanner sc=new Scanner(System.in);
```

```

System.out.println("Enter Student id");
int sid=sc.nextInt();
System.out.println("Enter Student Age");
int sage=sc.nextInt();

Person p= (id,age) -> System.out.println("Id "+id+"Age =" +age);

p.show(sid,sage);

System.out.println("Enter Employee id");
int eid=sc.nextInt();
System.out.println("Enter Employee Age");
int eage=sc.nextInt();

p.show(eid,eage);

Person p1= (id,age) -> {age=age+10;System.out.println("Age is " +age);};

p1.show(eid,eage);

```

```

}
}

```

//Backed Set

```

import java.util.*;
class SortedSetDemo
{
    public static void main (String args[])
    {
        TreeSet < String > set = new TreeSet < String > ();
        set.add ("A");
        set.add ("B");
        set.add ("C");
        set.add ("D");
        set.add ("E");
        System.out.println ("Intial Set: " + set);
        System.out.println ("Head Set: " + set.headSet ("C"));
        System.out.println ("SubSet: " + set.subSet ("A", "E"));
        System.out.println ("TailSet: " + set.tailSet ("C"));
        TreeSet < String > set1 = (TreeSet) set.subSet ("A", "C");
        // set1.add("F");
        System.out.println ("Intial Set: " + set+" Altered Set "+set1);
    }
}

```

```
import java.util.*;
class SortedSetDemo
{
    public static void main (String args[])
    {
        Integer[] integerArray = new Integer[3];
integerArray[0] = 1;
integerArray[1] = 2;
List<Integer> integerList = Arrays.asList(integerArray);
integerArray[2] = 3;
System.out.println(integerList);
    }
}
```