**1.Accept i/ps from User , till user enters "quit" or any other option.**

**I/P : operation(add|sub|mult|div) , number1(double) ,**

**number2(double)**

**Display the result of the operation.**

**It should be done in a loop , till user enters "quit"**

**1)Calculator.java**

import java.util.Scanner;

class Calculator

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

double a=0,b=0;

String ch="";

boolean flag = true;

System.out.println("List of Operations:->\n 1. Add\n 2. Subtract\n 3. Multiply\n 4. Division\n 5. Modulus\n 6. Exit");

while(flag != false)

{

System.out.print("\n------------------------------------------\nEnter the choice : ");

ch=sc.next();

switch(ch)

{

case "1":

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

System.out.print("\nPlease enter first number : ");

a=sc.nextDouble();

System.out.print("Please enter second number : ");

b=sc.nextDouble();

System.out.print("Addition : "+(a+b));

break;

case "2":

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

System.out.print("\nPlease enter first number : ");

a=sc.nextDouble();

System.out.print("Please enter second number : ");

b=sc.nextDouble();

System.out.print("Subtraction : "+(a-b));

break;

case "3":

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

System.out.print("\nPlease enter first number : ");

a=sc.nextDouble();

System.out.print("Please enter second number : ");

b=sc.nextDouble();

System.out.print("Multiplication : "+(a\*b));

break;

case "4":

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

System.out.print("\nPlease enter first number : ");

a=sc.nextDouble();

System.out.print("Please enter second number : ");

b=sc.nextDouble();

System.out.print("Division : "+(a/b));

break;

case "5":

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

System.out.print("\nPlease enter first number : ");

a=sc.nextDouble();

System.out.print("Please enter second number : ");

b=sc.nextDouble();

System.out.print("Modulus : "+(a%b));

break;

case "quit":

flag = false;

break;

default:

System.out.println("Please enter valid choice.");

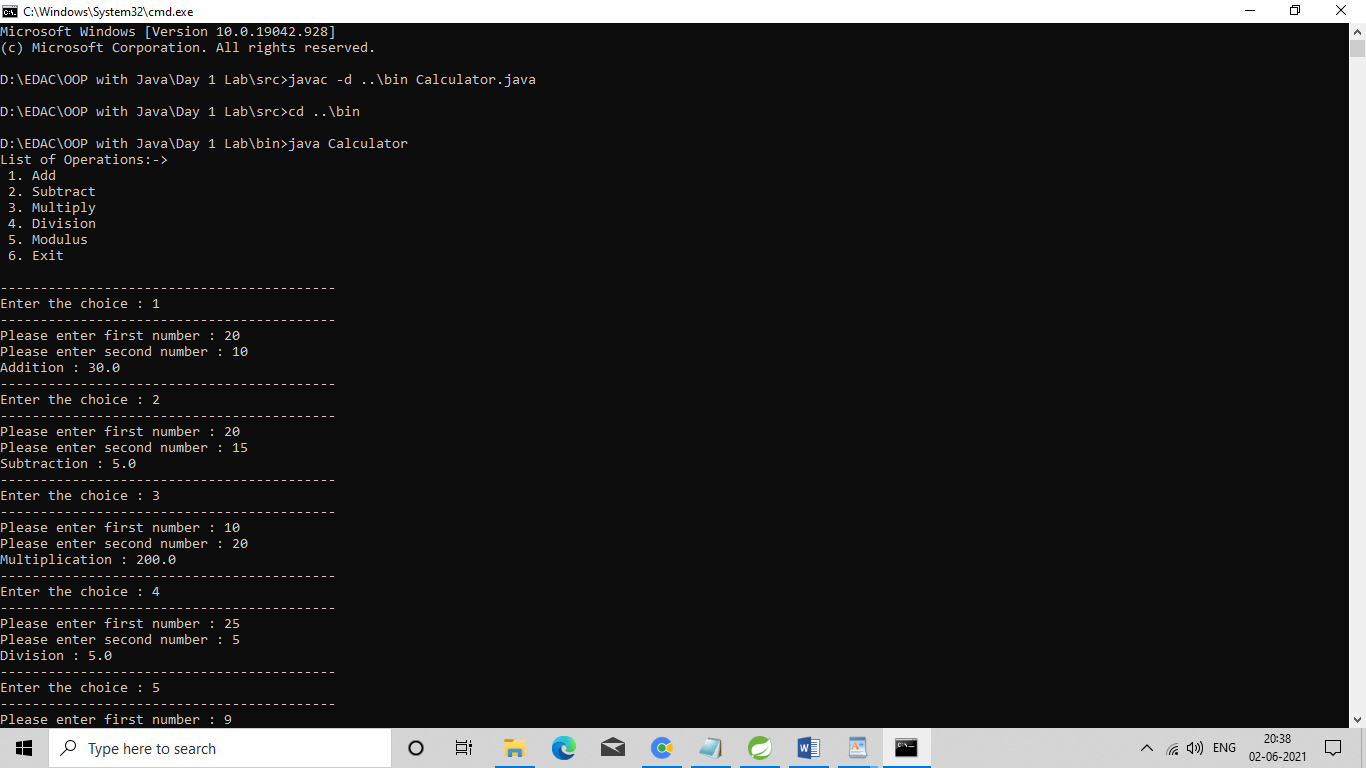
}

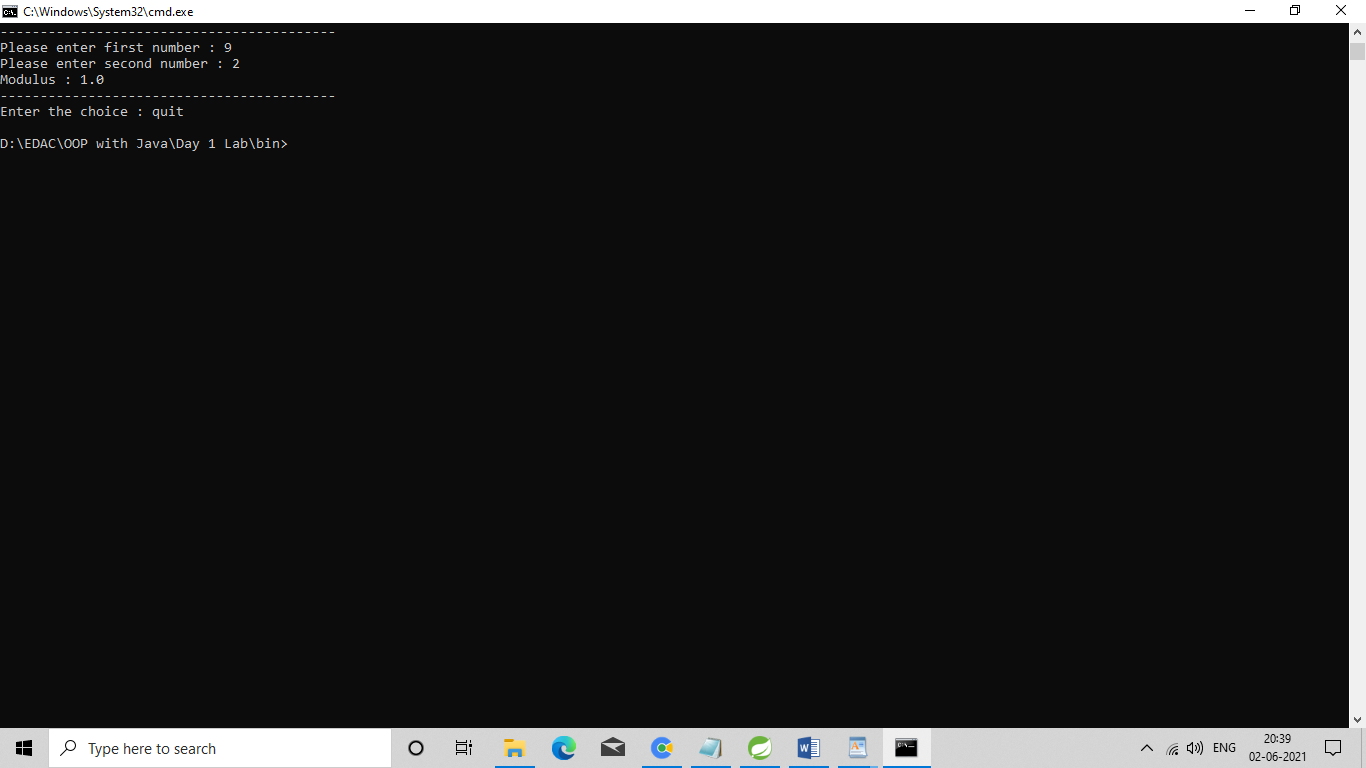
}

}

}

**Output :**





**2. Accept 2 double values from User (using Scanner). Check data type.**

**If arguments are not doubles , supply suitable error message &**

**terminate. If numbers are double values , print its average.**

**1)DoubleAverage.java**

import java.util.Scanner;

public class DoubleAverage {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter two double numbers:");

if(sc.hasNextDouble())

{

double d1=sc.nextDouble();

if(sc.hasNextDouble())

{

double d2=sc.nextDouble();

System.out.println("Avg ="+((d1+d2)/2));

}

else

{

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

}

}

else

{

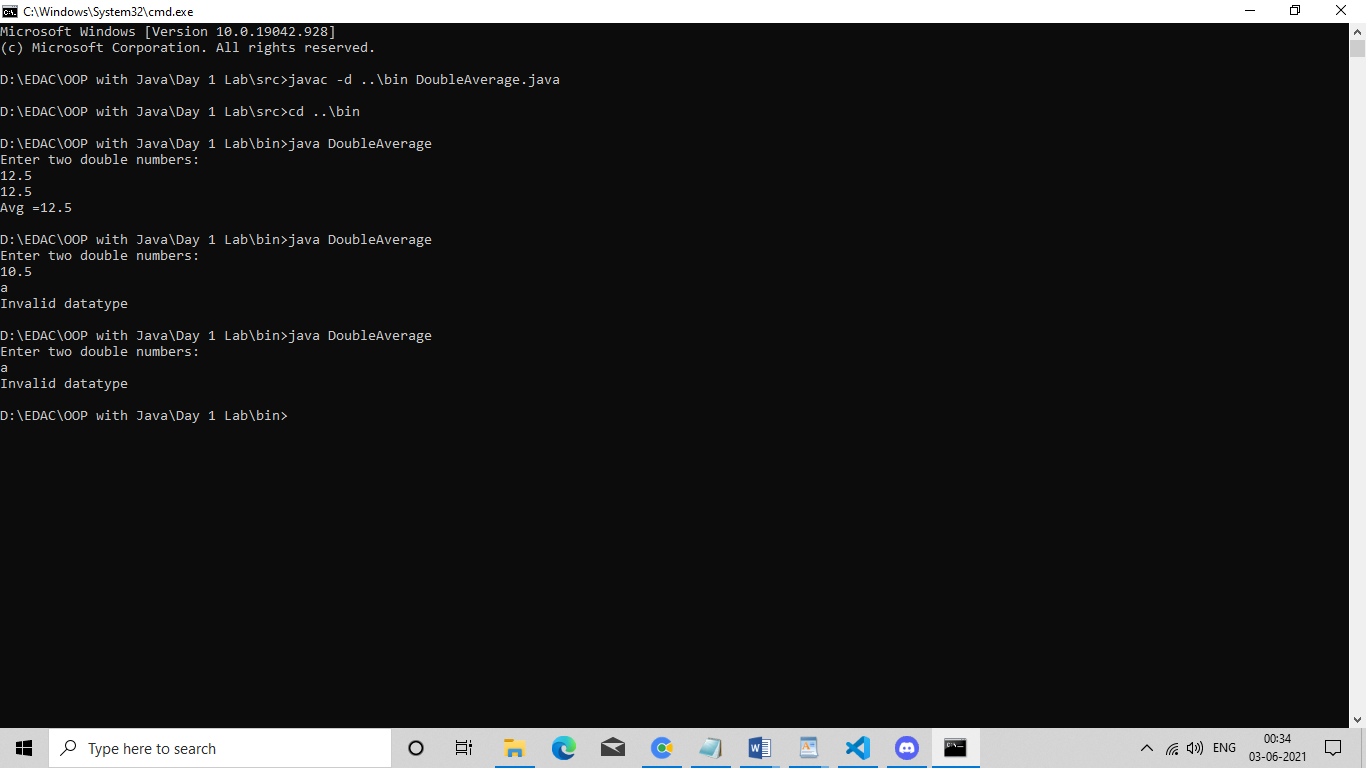
System.out.println("Invalid datatype");

}

}

}

**Output :**



**3. Display food menu to user. User will select items from menu along with the quantity. (eg 1. Dosa 2. Samosa .......10 . Generate Bill ) Assign fixed prices to food items .When user enters 'Generate Bill' option(10) , display total bill & exit.**

**1)FoodMenu.java**

import java.util.Scanner;

public class FoodMenu

{

private static double billTotal;

public static void main(String[] args)

{

System.out.println("Welcome to Food Cafe.");

System.out.println("Please select one of the Food Item below.");

System.out.println("Please select Exit after your order.");

menu();

}

public static void menu()

{

Scanner sc = new Scanner(System.in);

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

System.out.println("1.Dosa \t\t\t- \t\t40Rs\n2.Idli \t\t\t- \t\t20Rs\n3.Vada \t\t\t- \t\t30Rs\n4.Samosa \t\t- \t\t10Rs\n5.Pasta \t\t- \t\t100Rs");

System.out.println("6.Pizza \t\t- \t\t200Rs\n7.Burger \t\t- \t\t150Rs\n8.French Fries \t\t-\t\t100Rs\n9.Uttapa \t\t- \t\t50Rs\n10.Generate Bill");

boolean flag = false;

while (flag != true)

{

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

System.out.println("Enter Your Choice : ");

int choice = sc.nextInt();

switch(choice)

{

case 1:

System.out.println("Enter Quantity of Dosa : ");

int dosa = sc.nextInt();

billTotal = billTotal + (40 \* dosa);

break;

case 2:

System.out.println("Enter Quantity of Idli : ");

int idli = sc.nextInt();

billTotal = billTotal + (20 \* idli);

break;

case 3:

System.out.println("Enter Quantity of Vada : ");

int vada = sc.nextInt();

billTotal = billTotal + (30 \* vada);

break;

case 4:

System.out.println("Enter Quantity of Samosa : ");

int samosa = sc.nextInt();

billTotal = billTotal + (10 \* samosa);

break;

case 5:

System.out.println("Enter Quantity of Pasta : ");

int pasta = sc.nextInt();

billTotal = billTotal + (100 \* pasta);

break;

case 6:

System.out.println("Enter Quantity of Pizza : ");

int pizza = sc.nextInt();

billTotal = billTotal + (200 \* pizza);

break;

case 7:

System.out.println("Enter Quantity of Burger : ");

int burger = sc.nextInt();

billTotal = billTotal + (150 \* burger);

break;

case 8:

System.out.println("Enter Quantity of French Fries : ");

int fries = sc.nextInt();

billTotal = billTotal + (100 \* fries);

break;

case 9:

System.out.println("Enter Quantity of Uttapa : ");

int uttapa = sc.nextInt();

billTotal = billTotal + (50 \* uttapa);

break;

case 10:

Bill(billTotal);

break;

default:

System.out.println("Sorry!!! Currently not available.");

break;

}

}

}

public static void Bill(double x)

{

System.out.println("Do you want anything else?\n1.Yes\n2.No");

Scanner sc = new Scanner(System.in);

int reply = sc.nextInt();

if (reply == 1)

{

menu();

sc.close();

}

else

{

System.out.println("Your total bill is : "+x);

System.out.println("Thank You. Visit Again.");

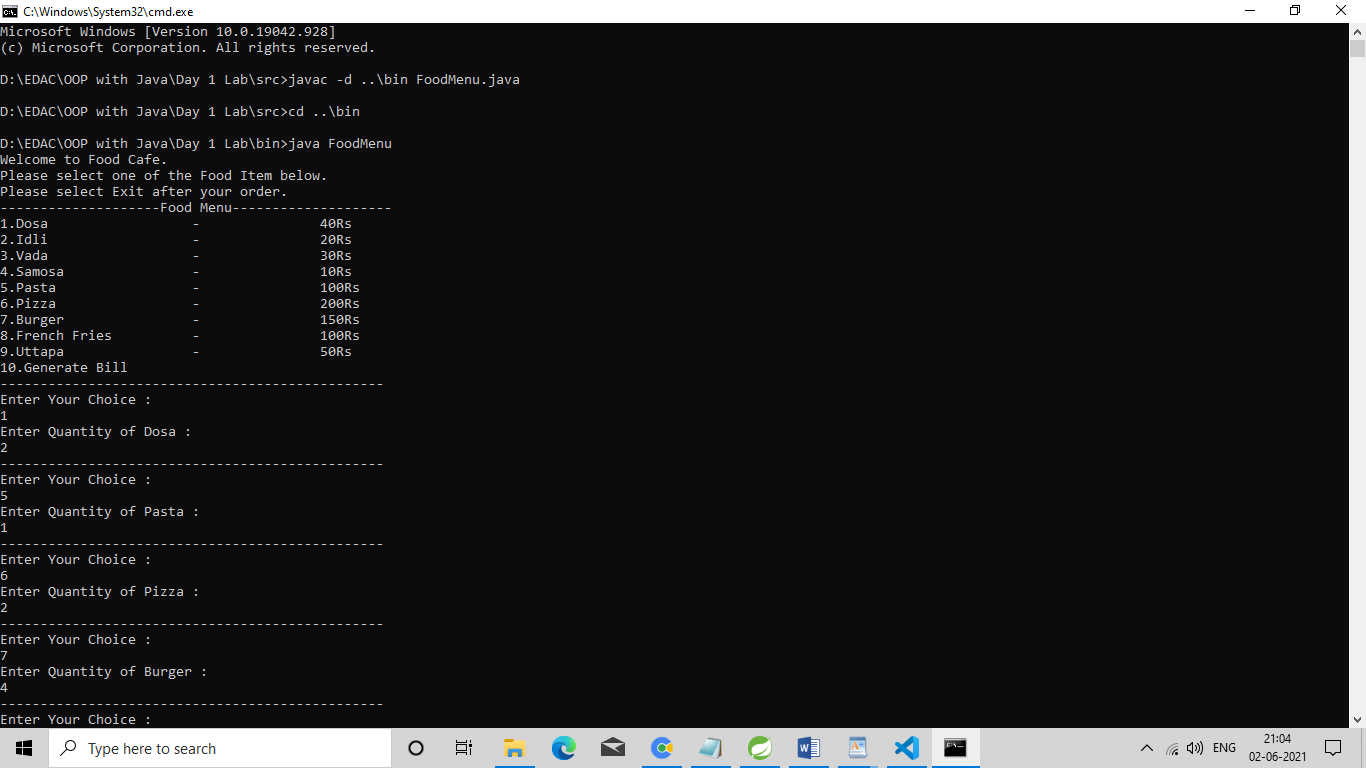
System.exit(0);

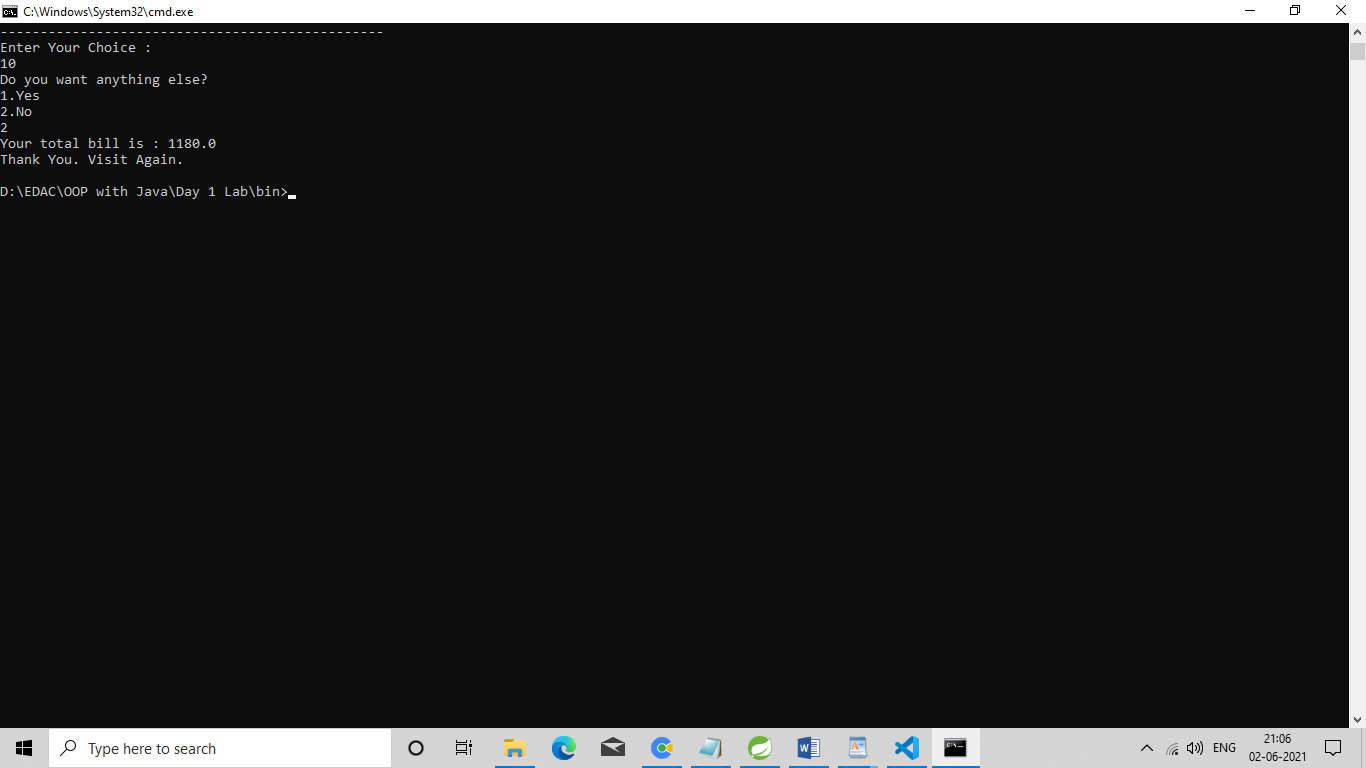
}

}

}

**Output :**





**1. Solve Tank assignment along with memory picture.**

**What will be the output ?**

**1)Tank.java**

class Tank

{

private int level;

Tank(int l)

{

level=l;

}

public void setLevel(int level1)

{

level=level1;

}

public int getLevel()

{

return level;

}

}

**2)Assignment.java**

public class Assignment {

public static void main(String[] args) {

Tank t1 = new Tank(10);

Tank t2 = new Tank(20);

System.out.println("1: t1.level: " + t1.getLevel() + ", t2.level: " + t2.getLevel());

t1 = t2;

System.out.println("2: t1.level: " + t1.getLevel() + ", t2.level: " + t2.getLevel());

t1.setLevel(27);

System.out.println("3: t1.level: " + t1.getLevel() + ", t2.level: " + t2.getLevel());

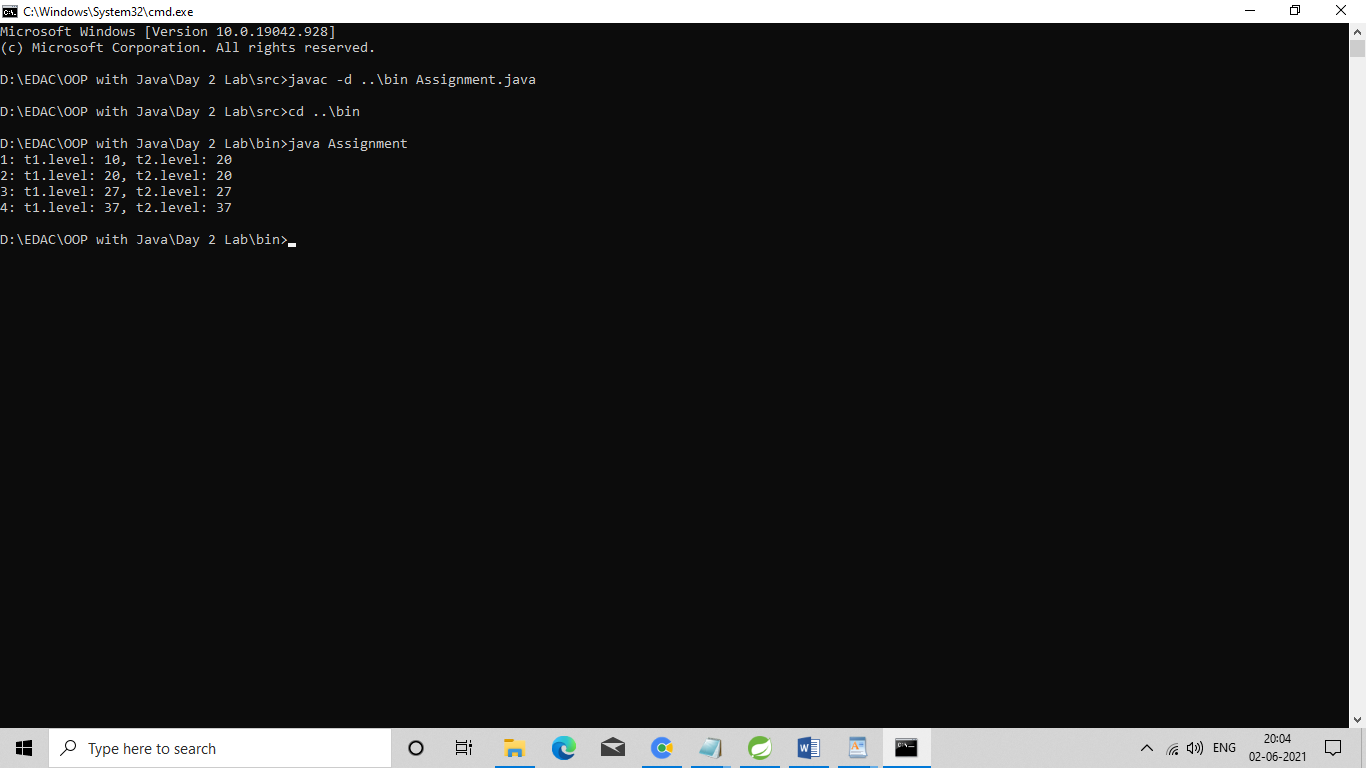
t2.setLevel(t1.getLevel()+10);

System.out.println("4: t1.level: " + t1.getLevel() + ", t2.level: " + t2.getLevel());

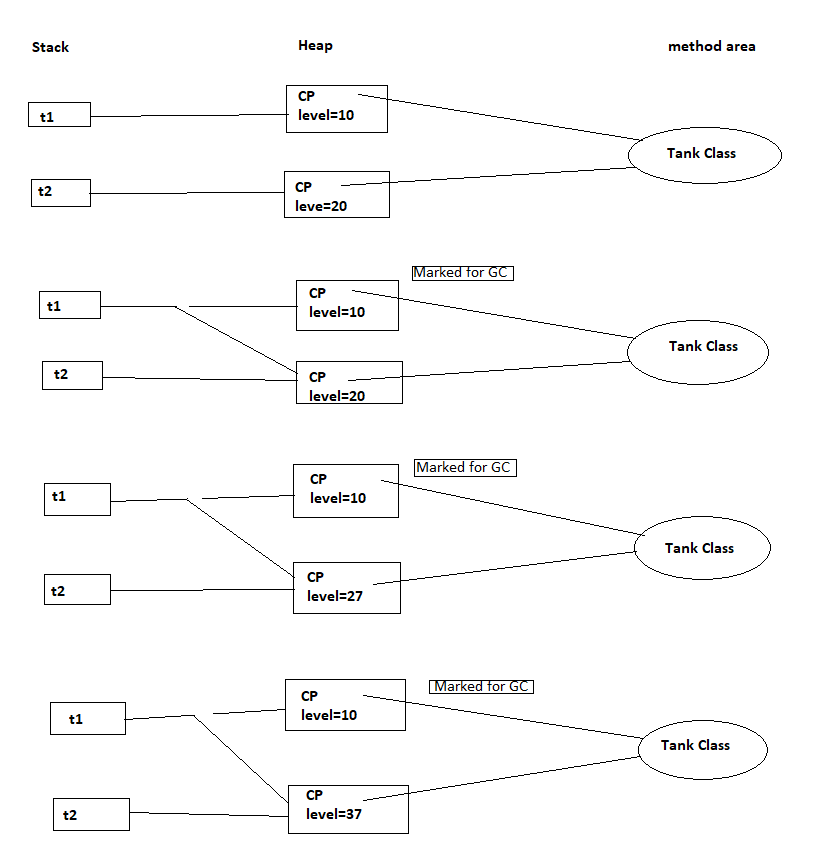
}

}

**Output :**



**Memory Diagram :**



**2. Solve**

**Create Point class Point2D : for representing a point in x-y co-ordinate system.**

**2.1 Create a parameterized constructor to accept x & y co-ords.**

**2.2 Add public String getDetails()) --to return string form point's x & y**

**co-ords**

**2.3 Add isEqual method to Point2D class : boolean returning method :**

**must return true if both points are having same x,y co-ords or false**

**otherwise.**

**2.4 Add a method to Point2D class -- to create and return new point having**

**given x & y offset.**

**eg : Point2D p1=new Point2D(10,20);**

**Point2d p3=p1.createNewPoint(5,-2);//p3 : 15,18**

**If user supplies offset of (5, -2) : your method should return a new**

**point**

**object placed at (15,18)**

**eg : Point2D createNewPoint(int xOff,iny yOff){...}**

**2.5 Add calculateDistance method to calculate distance between current**

**point & specified point & return the distance to the caller.**

**(eg double calcDistance(Point2D p2))**

**Hint : Use distance formula . Use java.lang.Math class methods --sqrt,**

**pow etc.**

**2.6 Write TestPoint class , under "com.app.tester" package with a main**

**method**

**Accept co ordinates of 2 points from user (Scanner) --p1 & p2**

**2.7 Use getDetails method to display point details.(p1's details & p2's**

**details)**

**2.8 Invoke isEqual & display if points are same or different (i.e p1 & p2 are**

**located at the same position)**

**2.9 Create new point p3 , with the dimensions offset from p1.**

**I/P --x offset & y offset**

**2.10 Display distance between 2 points.(between p1 & p2)**

**1)Point2D.java**

import java.util.\*;

class Point2D

{

private int x , y;

Point2D(int x, int y)

{

this.x = x;

this.y = y;

}

public String getDetails()

{

return "Point ( "+this.x+" , "+this.y+" ) " ;

}

boolean isEqual(Point2D anotherPoint)// anotherPoint2D=p2 : copy of refs

{

return this.x == anotherPoint.x && this.y == anotherPoint.y;

}

Point2D createNewPoint(int xOff, int yOff)

{

Point2D newPoint = new Point2D(this.x + xOff, this.y + yOff);

return newPoint;

}

double calculateDistance(Point2D anotherPoint)

{

double diff1 = anotherPoint.x-this.x;

double diff2 = anotherPoint.y-this.y;

double pow1 = Math.pow(diff1, 2);

double pow2 = Math.pow(diff2, 2);

double sum = pow1 + pow2;

double squareroot = Math.sqrt(sum);

return squareroot;

}

}

**2)TestPoint.java**

import java.util.Scanner;

class TestPoint

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter x and y co-ordinates of point p1 : ");

Point2D p1 = new Point2D(sc.nextInt(), sc.nextInt());

System.out.println("Enter x and y co-ordinates of point p2 : ");

Point2D p2 = new Point2D(sc.nextInt(), sc.nextInt());

System.out.println("First Point : "+p1.getDetails());

System.out.println("Second Point : "+p2.getDetails());

System.out.print("Point p1 and point p2 are ");

System.out.println(p1.isEqual(p2) ? "Same" : "Different");

System.out.println("Enter offset value from p1 : ");

Point2D p3 = p1.createNewPoint(sc.nextInt(), sc.nextInt());

System.out.println("Old Point : "+p1.getDetails());

System.out.println("New Point : "+p3.getDetails());

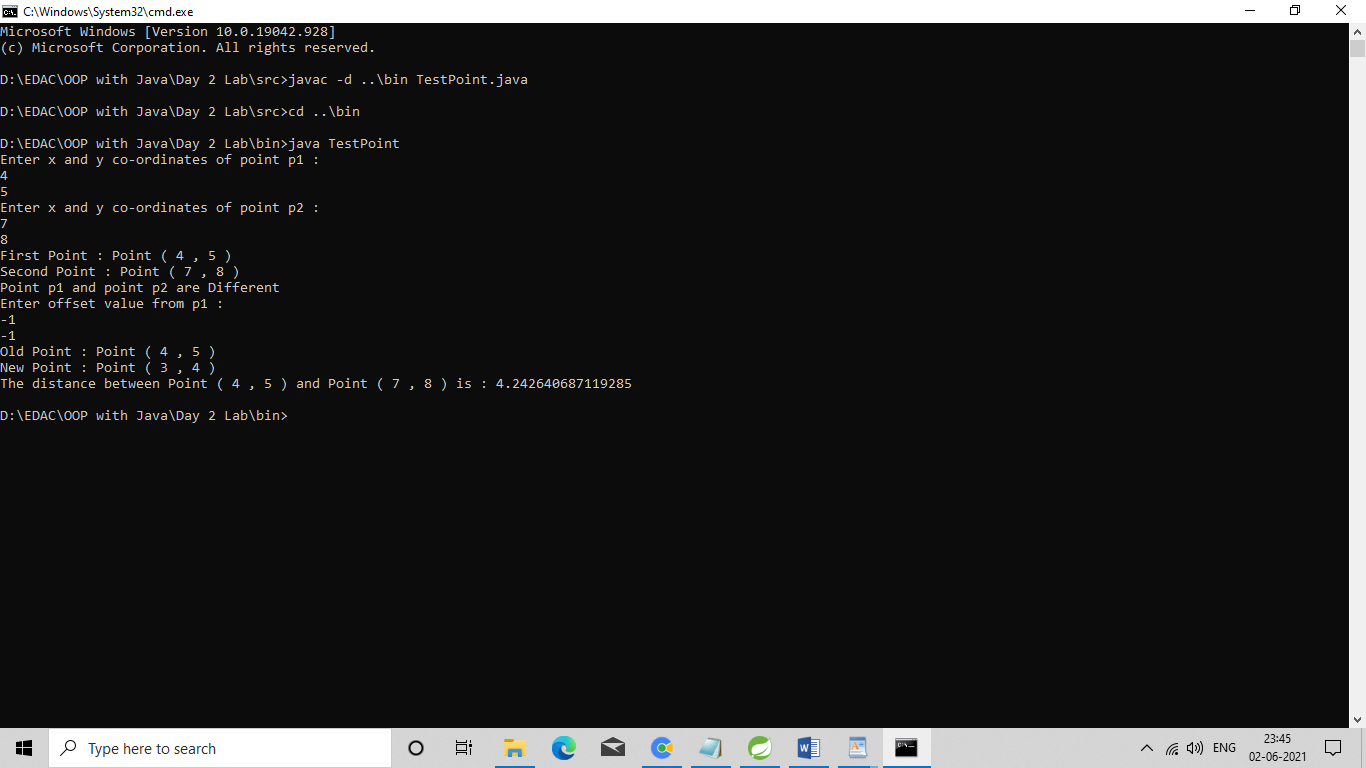
System.out.println("The distance between "+p1.getDetails()+"and "+p2.getDetails() +"is : "+p1.calculateDistance(p2));

sc.close();

}

}

**Output :**



**1.Copy earlier created Point2D class n place it under the package "com.geometry"**

**Copy earlier created TestPoint class n place it under "tester" package.**

**Solve compiler errors n run it without IDE (set CLASSPATH , so that you can run it from anywhere)**

**1)Point2D.java**

package com.geometry;

import java.util.\*;

public class Point2D

{

private int x , y;

public Point2D(int x, int y)

{

this.x = x;

this.y = y;

}

public String getDetails()

{

return "Point ( "+this.x+" , "+this.y+" ) " ;

}

public boolean isEqual(Point2D anotherPoint)// anotherPoint2D=p2 : copy of refs

{

return this.x == anotherPoint.x && this.y == anotherPoint.y;

}

public Point2D createNewPoint(int xOff, int yOff)

{

Point2D newPoint = new Point2D(this.x + xOff, this.y + yOff);

return newPoint;

}

public double calculateDistance(Point2D anotherPoint)

{

double diff1 = anotherPoint.x-this.x;

double diff2 = anotherPoint.y-this.y;

double pow1 = Math.pow(diff1, 2);

double pow2 = Math.pow(diff2, 2);

double sum = pow1 + pow2;

double squareroot = Math.sqrt(sum);

return squareroot;

}

}

**2)TestPoint.java**

package com.tester;

import com.geometry.Point2D;

import java.util.Scanner;

class TestPoint

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter x and y co-ordinates of point p1 : ");

Point2D p1 = new Point2D(sc.nextInt(), sc.nextInt());

System.out.println("Enter x and y co-ordinates of point p2 : ");

Point2D p2 = new Point2D(sc.nextInt(), sc.nextInt());

System.out.println("First Point : "+p1.getDetails());

System.out.println("Second Point : "+p2.getDetails());

System.out.print("Point p1 and point p2 are ");

System.out.println(p1.isEqual(p2) ? "Same" : "Different");

System.out.println("Enter offset value from p1 : ");

Point2D p3 = p1.createNewPoint(sc.nextInt(), sc.nextInt());

System.out.println("Old Point : "+p1.getDetails());

System.out.println("New Point : "+p3.getDetails());

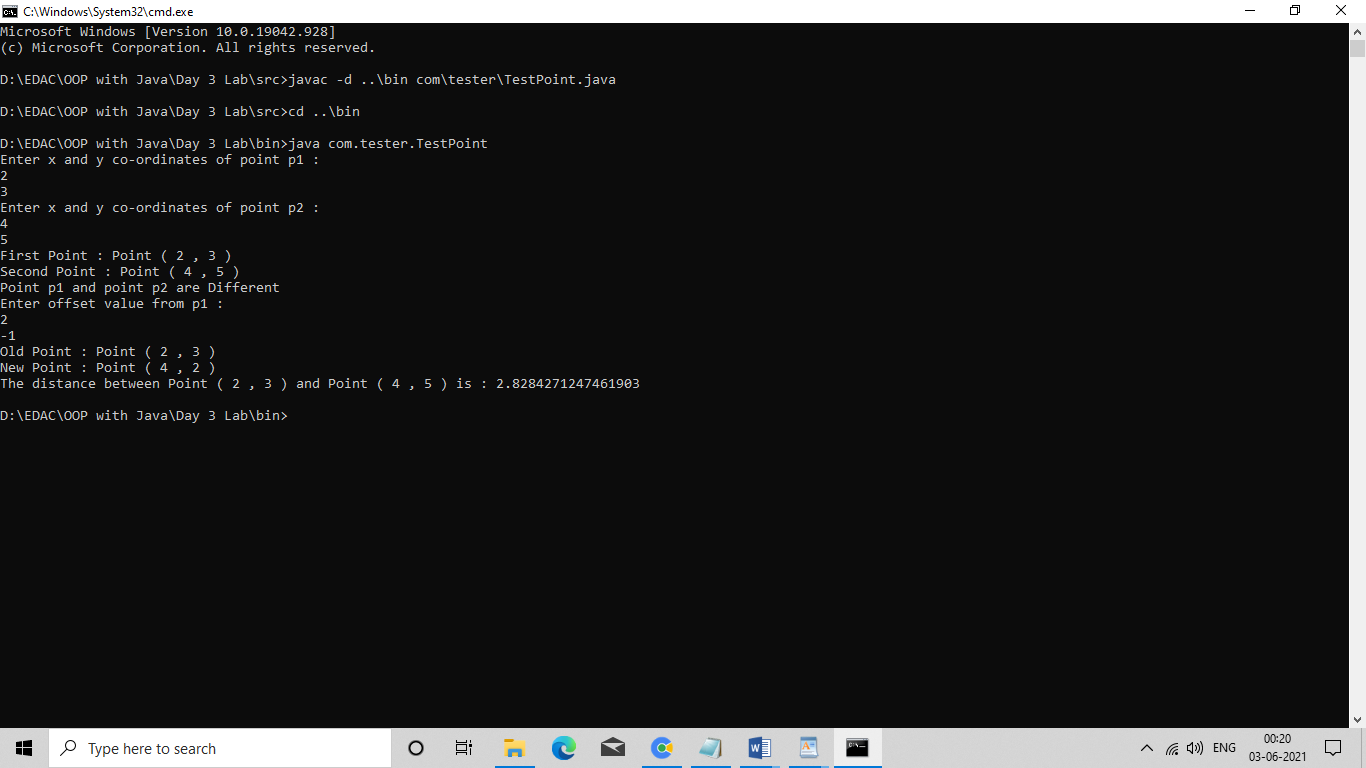
System.out.println("The distance between "+p1.getDetails()+"and "+p2.getDetails() +"is : "+p1.calculateDistance(p2));

sc.close();

}

}

**Output :**



**2. Solve**

**Create new eclipse project "day3\_assgnment"**

**Copy Point2D class n TestPoints classalong with the packages in IDE.**

**2.1 Prompt user , how many points to plot?**

**Create suitable array , to store Point2D type of references.**

**2.2 Add a menu , Run the application till user chooses option 10**

**(exit)**

**1. Plot a new point**

**I/P --index , x & y**

**eg : If user supplies 5 50 100**

**Create a Point2D with x, y of (50,100) & its reference should be**

**stored at the 5th index position in array.**

**2. Display all points plotted so far. (Use for-each)**

**Will null checking be required?**

**3. Test equality of 2 points**

**I/P : index1 , index 2**

**eg If user enters 1 7 , you have to check if point at 1st n 7th**

**index is same.**

**If same , print "SAME" , otherwise print "DIFFERENT"**

**4. Calculate distance**

**I/P strt , end point indexes.**

**eg : If user enters 2 6**

**Find out the distance between 2nd & 6th point. (array indexes**

**start from 0)**

**10. Exit**

**Note : Boundary condition checking & null checking is expected.**

**(Meaning : if user supplies invalid index , then give error message : invalid index --out of range**

**If no point is plotted at user specified index , then give error message : invalid index -- no point plotted here.)**

**1)Point2D.java**

package com.geometry;

import java.util.\*;

public class Point2D

{

private int x , y;

public Point2D(int x, int y)

{

this.x = x;

this.y = y;

}

public String getDetails()

{

return "Point ( "+this.x+" , "+this.y+" ) " ;

}

public boolean isEqual(Point2D anotherPoint)// anotherPoint2D=p2 : copy of refs

{

return this.x == anotherPoint.x && this.y == anotherPoint.y;

}

public Point2D createNewPoint(int xOff, int yOff)

{

Point2D newPoint = new Point2D(this.x + xOff, this.y + yOff);

return newPoint;

}

public double calculateDistance(Point2D anotherPoint)

{

double diff1 = anotherPoint.x-this.x;

double diff2 = anotherPoint.y-this.y;

double pow1 = Math.pow(diff1, 2);

double pow2 = Math.pow(diff2, 2);

double sum = pow1 + pow2;

double squareroot = Math.sqrt(sum);

return squareroot;

}

}

**2)TestPointArray.java**

package com.tester;

import com.geometry.Point2D;

import java.util.Arrays;

import java.util.Scanner;

class TestPointArray

{

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of points to create : ");

Point2D[] points;

points = new Point2D[sc.nextInt()];

System.out.println("Points are : "+Arrays.toString(points));

boolean flag = true;

while(flag != false)

{

System.out.println("1.Plot a new point");

System.out.println("2.Display all points plotted");

System.out.println("3.Test equality of 2 points");

System.out.println("4.Calculate distance between two points");

System.out.println("10.Exit");

System.out.println("Enter your choice : ");

int choice = sc.nextInt();

switch(choice)

{

case 1:

System.out.println("Enter index where point want to plot : ");

int index = sc.nextInt();

if (index >= 0 && index < points.length )

{

System.out.println("Enter x and y co-ordinates : ");

points[index] = new Point2D(sc.nextInt(), sc.nextInt());//parameterized constructor of class Point2D

System.out.println("Point is : "+points[index].getDetails());

}

else

{

System.out.println("Please enter valid index.");

}

break;

case 2:

System.out.println("Points are : ");

for (Point2D p : points)

{

if(p != null)

{

System.out.print(p.getDetails()+" ");

System.out.println();

}

}

break;

case 3:

System.out.println("Enter index1 and index2 : ");

int index1=sc.nextInt();

int index2=sc.nextInt();

if(index1 >= 0 && index1 < points.length && index2 >= 0 && index2 < points.length)

{

if (points[index1] != null && points[index2] != null)

{

System.out.print("Points are ");

System.out.println(points[index1].isEqual(points[index2]) ? "SAME" : "DIFFERENT");

}

else

{

System.out.println("No point plotted here.");

}

}

else

{

System.out.println("Please enter valid index.");

}

break;

case 4:

System.out.println("Enter index1 and index2 : ");

int i1=sc.nextInt();

int i2=sc.nextInt();

if(i1 >= 0 && i1 < points.length && i2 >= 0 && i2 < points.length)

{

if(points[i1] != null && points[i2] != null)

{

System.out.println("The distance between "+points[i1].getDetails()+"and "+points[i2].getDetails() +"is : "+points[i1].calculateDistance(points[i2]));

}

else

{

System.out.println("No point plotted here.");

}

}

else

{

System.out.println("Please enter valid index.");

}

break;

default:

System.out.println("Invalid choice!! Please enter valid choice.");

break;

case 10:

flag = false;

break;

}

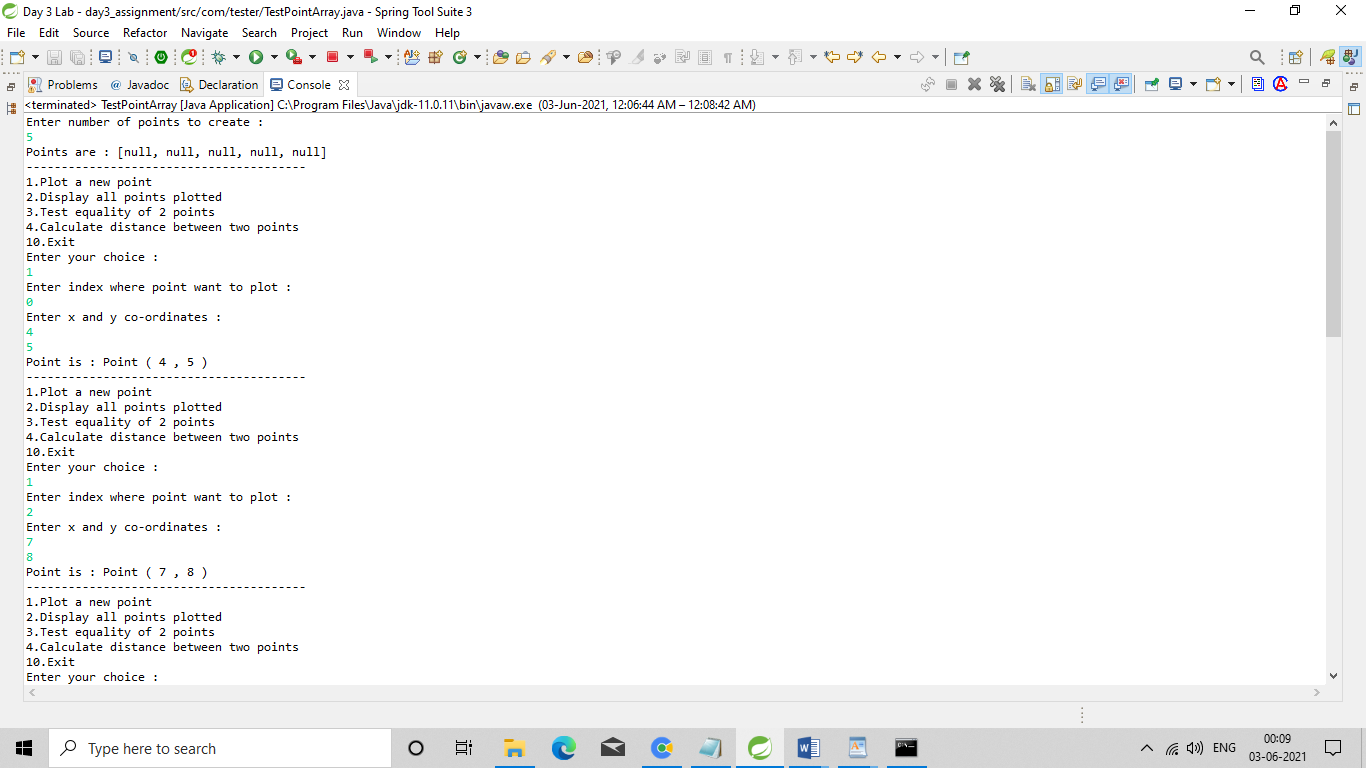
}

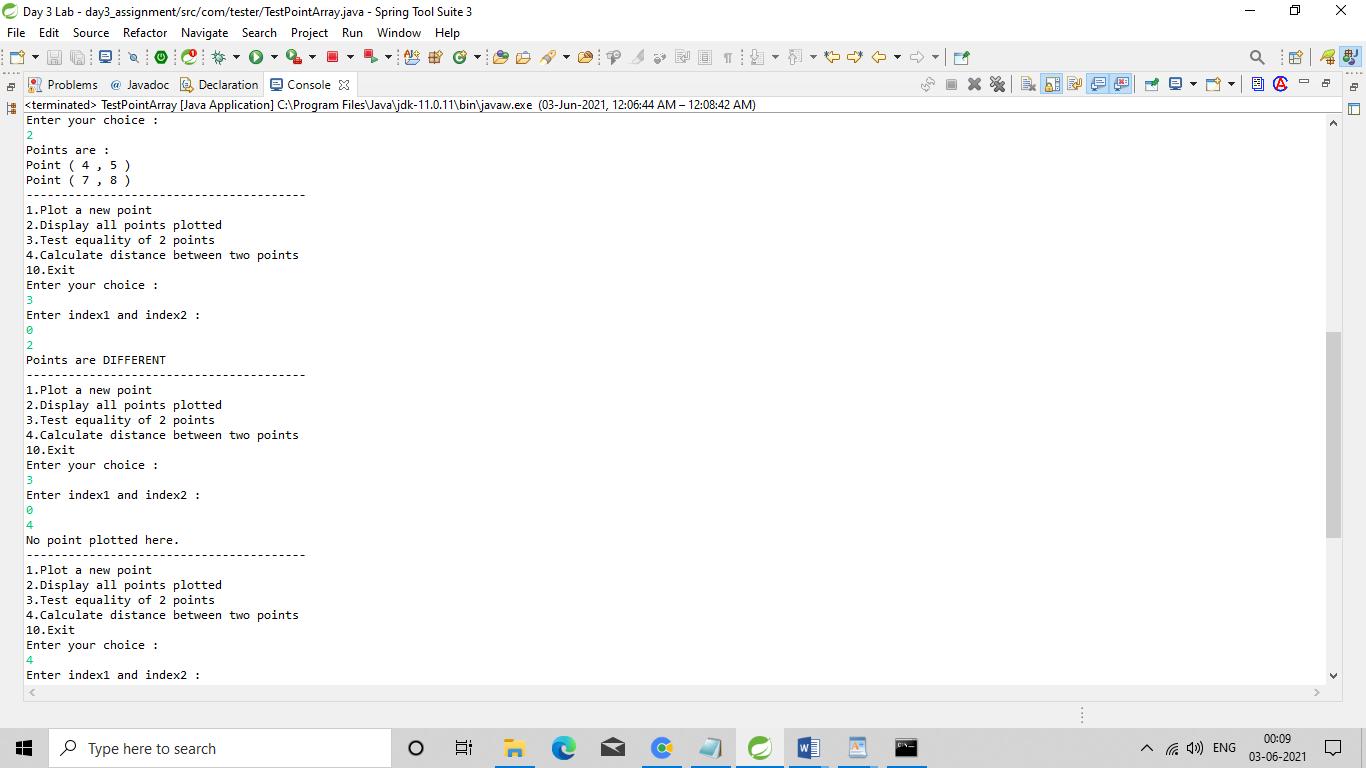
sc.close();

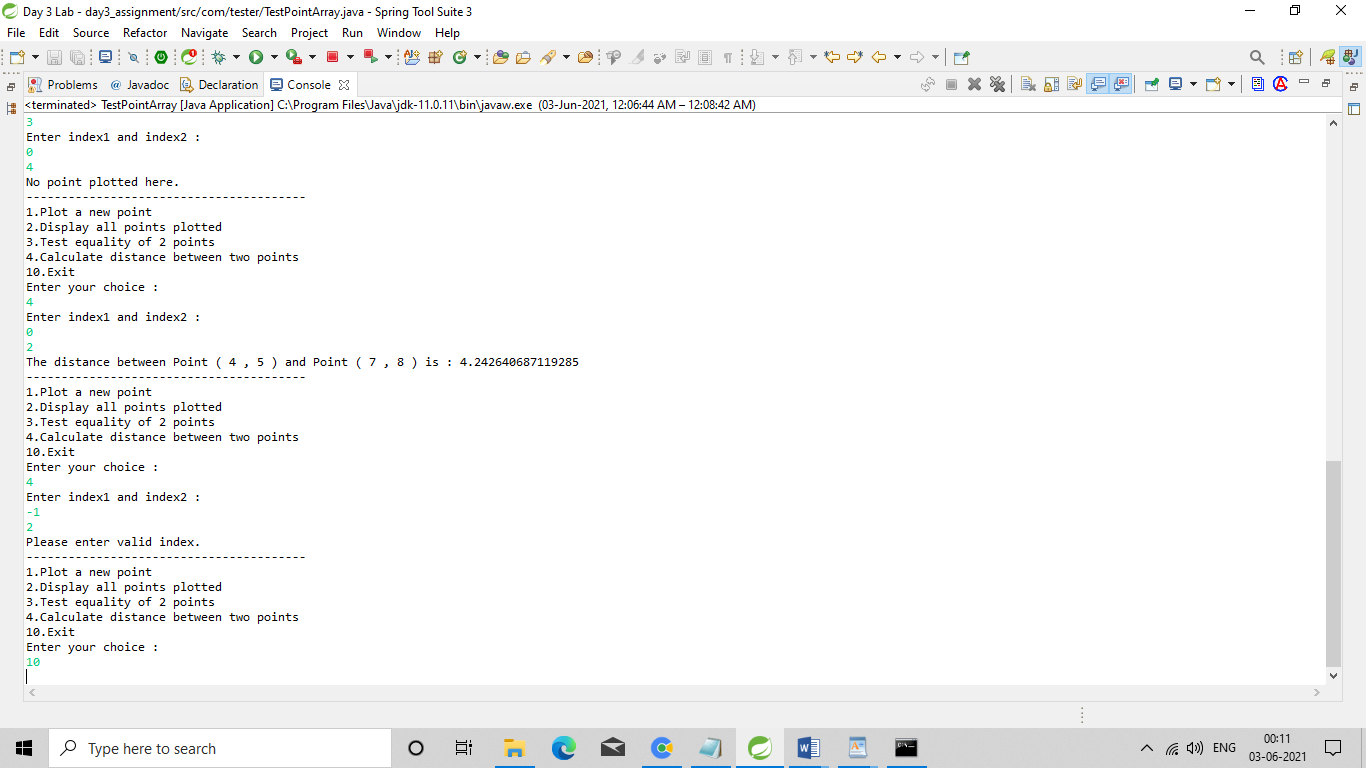
}

}

**Output :**







**1)Employee.java**

/\*2. Apply inheritance & polymorphism to emp based organization scenario.

Create Emp based organization structure --- Emp , Mgr , Worker

All of above classes must be in package--com.app.org

2.1 Emp state--- id(int), name, deptId(string) , basic(double)

Accept all of above in constructor arguments.

Behaviour ---1. get emp details -- override toString.

2. compute net salary ---return 0

(eg : public double computeNetSalary(){return 0;})\*/

**package** com.app.org;

**public** **class** Employee {

**private** **int** id , deptId;

**private** String name;

**private** **double** basicSalary;

**public** Employee(**int** id, String name, **int** deptId, **double** basicSalary)

{

**super**();

**this**.id = id;

**this**.name = name;

**this**.deptId = deptId;

**this**.basicSalary = basicSalary;

}

@Override

**public** String toString()

{

**return** "Employee : id=" + id + ", name=" + name + ", deptId=" + deptId +", basicSalary=" + basicSalary ;

}

**public** **double** computeNetSalary()

{

**return** 0;

}

**public** **double** getBasicSalary()

{

**return** basicSalary;

}

}

**2)Manager.java**

/\*2.2 Mgr state ---id,name,basic,deptId , perfmonceBonus

Behaviour ----1. get mgr details : override toString.

2. compute net salary (formula: basic+perfmonceBonus) -- override computeNetSalary

3. get performance bonus. --add a new method to return bonus.(getter)\*/

**package** com.app.org;

**public** **class** Manager **extends** Employee{

**private** **double** perfmonceBonus;

**public** Manager(**int** id, String name, **int** deptId, **double** basicSalary, **double** perfmonceBonus)

{

**super**(id, name, deptId, basicSalary);

**this**.perfmonceBonus = perfmonceBonus;

}

@Override

**public** String toString() {

**return** "Manager ["+**super**.toString()+ " , perfmonceBonus=" + perfmonceBonus + "]";

}

@Override

**public** **double** computeNetSalary()

{

**double** computeNetSalary = getBasicSalary() + perfmonceBonus;

**return** computeNetSalary;

}

**public** **double** getPerfmonceBonus()

{

**return** perfmonceBonus;

}

}

**3)Worker.java**

/\*

2.3 Worker state --id,name,basic,deptId,hoursWorked,hourlyRate

Behaviour---

1. get worker details -- : override toString.

2. compute net salary (formula: = basic+(hoursWorked\*hourlyRate) --override computeNetSalary

3. get hrlyRate of the worker -- add a new method to return hourly rate of a worker.(getter)

\*/

**package** com.app.org;

**public** **class** Worker **extends** Employee {

**private** **double** hoursWorked, hourlyRate;

**public** Worker(**int** id, String name, **int** deptId, **double** basicSalary, **double** hoursWorked, **double** hourlyRate)

{

**super**(id, name, deptId, basicSalary);

**this**.hoursWorked = hoursWorked;

**this**.hourlyRate = hourlyRate;

}

@Override

**public** String toString()

{

**return** "Worker ["+**super**.toString()+" , hoursWorked=" + hoursWorked + ", hourlyRate=" + hourlyRate + "]";

}

@Override

**public** **double** computeNetSalary()

{

**double** computeNetSalary = getBasicSalary() + (hoursWorked \* hourlyRate);

**return** computeNetSalary;

}

**public** **double** getHourlyRate()

{

**return** hourlyRate;

}

}

**4)TestOrganization.java**

/\*2.4 Write TestOrganization in "tester" package.

Create suitable array to store organization details

Provide following options. Run the application till "exit"

1. Hire Manager

2. Hire Worker

3. Display information of all employees including net salary using single for-each loop.

Display from the same for-each loop, performance bonus if it's a manager or if it's a worker , display hourly rate of the worker .10 Exit\*/

package tester;

import java.util.Scanner;

import com.app.org.Employee;

import com.app.org.Manager;

import com.app.org.Worker;

public class TestOrganization {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of employees you want to recruit : ");

Employee[] emps = new Employee[sc.nextInt()];

int counter = 0;

boolean exit = false;

while(exit != true)

{

System.out.println("1. Hire Manager\n2. Hire Worker\n3. Display Information\n10. Exit");

System.out.println("Enter your choice : ");

switch(sc.nextInt())

{

case 1 :

if (counter >= 0 && counter < emps.length)

{

System.out.println("Enter details of manager : id, name, deptId, basicSalary, perfmonceBonus ");

emps[counter++] = new Manager(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextDouble(), sc.nextDouble());

}

else

{

System.out.println("Cannot recruit.");

}

break;

case 2 :

if (counter >= 0 && counter < emps.length)

{

System.out.println("Enter details of worker : id, name, deptId, basicSalary, hoursWorked, hourlyRate ");

emps[counter++] = new Worker(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextDouble(), sc.nextDouble(), sc.nextDouble());

}

else

{

System.out.println("Cannot recruit.");

}

break;

case 3 :

for (Employee e : emps)

{

if(e != null)

{

System.out.println(e);

System.out.println("Net Salary : "+e.computeNetSalary());

if (e instanceof Manager)

{

System.out.println("Performance Bonus : "+((Manager)e).getPerfmonceBonus());

}

else if (e instanceof Worker)

{

System.out.println("Hourly Rate : "+((Worker)e).getHourlyRate());

}

}

}

break;

case 10 :

exit = true;

break;

}

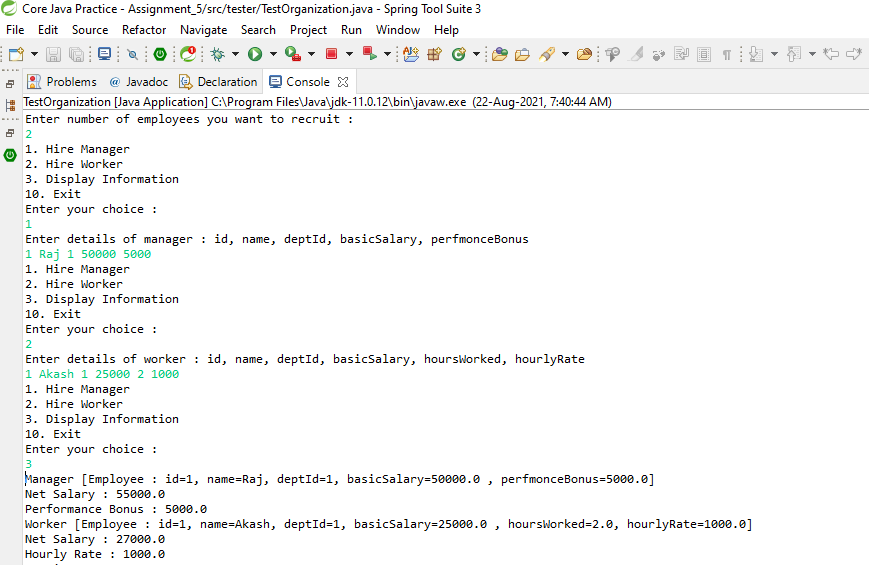
}

sc.close();

}

}

**Output :**

****

****

**1)Employee.java**

/\*2. Apply inheritance & polymorphism to emp based organization scenario.

Create Emp based organization structure --- Emp , Mgr , Worker

All of above classes must be in package--com.app.org

2.1 Emp state--- id(int), name, deptId(string) , basic(double)

Accept all of above in constructor arguments.

Behaviour ---1. get emp details -- override toString.

2. compute net salary ---return 0

(eg : public double computeNetSalary(){return 0;})\*/

package com.app.org;

public abstract class Employee {

private int id , deptId;

private String name;

private double basicSalary;

public Employee(int id, String name, int deptId, double basicSalary)

{

super();

this.id = id;

this.name = name;

this.deptId = deptId;

this.basicSalary = basicSalary;

}

@Override

public String toString()

{

return "Employee : id=" + id + ", name=" + name + ", deptId=" + deptId +", basicSalary=" + basicSalary ;

}

public abstract double computeNetSalary();

public double getBasicSalary()

{

return basicSalary;

}

}

**2)Manager.java**

/\*2.2 Mgr state ---id,name,basic,deptId , perfmonceBonus

Behaviour ----1. get mgr details : override toString.

2. compute net salary (formula: basic+perfmonceBonus) -- override computeNetSalary

3. get performance bonus. --add a new method to return bonus.(getter)\*/

package com.app.org;

public class Manager extends Employee{

private double perfmonceBonus;

public Manager(int id, String name, int deptId, double basicSalary, double perfmonceBonus)

{

super(id, name, deptId, basicSalary);

this.perfmonceBonus = perfmonceBonus;

}

@Override

public String toString() {

return "Manager ["+super.toString()+ " , perfmonceBonus=" + perfmonceBonus + "]";

}

@Override

public double computeNetSalary()

{

double computeNetSalary = getBasicSalary() + perfmonceBonus;

return computeNetSalary;

}

public double getPerfmonceBonus()

{

return perfmonceBonus;

}

}

**3)Worker.java**

/\*

2.3 Worker state --id,name,basic,deptId,hoursWorked,hourlyRate

Behaviour---

1. get worker details -- : override toString.

2. compute net salary (formula: = basic+(hoursWorked\*hourlyRate) --override computeNetSalary

3. get hrlyRate of the worker -- add a new method to return hourly rate of a worker.(getter)

\*/

package com.app.org;

public class Worker extends Employee {

private double hoursWorked, hourlyRate;

public Worker(int id, String name, int deptId, double basicSalary, double hoursWorked, double hourlyRate)

{

super(id, name, deptId, basicSalary);

this.hoursWorked = hoursWorked;

this.hourlyRate = hourlyRate;

}

@Override

public String toString()

{

return "Worker ["+super.toString()+" , hoursWorked=" + hoursWorked + ", hourlyRate=" + hourlyRate + "]";

}

@Override

public double computeNetSalary()

{

double computeNetSalary = getBasicSalary() + (hoursWorked \* hourlyRate);

return computeNetSalary;

}

public double getHourlyRate()

{

return hourlyRate;

}

}

**4)TestOrganization.java**

/\*2.4 Write TestOrganization in "tester" package.

Create suitable array to store organization details

Provide following options. Run the application till "exit"

1. Hire Manager

2. Hire Worker

3. Display information of all employees including net salary using single for-each loop.

Display from the same for-each loop, performance bonus if it's a manager or if it's a worker , display hourly rate of the worker .10 Exit\*/

package tester;

import java.util.Scanner;

import com.app.org.Employee;

import com.app.org.Manager;

import com.app.org.Worker;

public class TestOrganization {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of employees you want to recruit : ");

Employee[] emps = new Employee[sc.nextInt()];

int counter = 0;

boolean exit = false;

while(exit != true)

{

System.out.println("1. Hire Manager\n2. Hire Worker\n3. Display Information\n10. Exit");

System.out.println("Enter your choice : ");

switch(sc.nextInt())

{

case 1 :

if (counter >= 0 && counter < emps.length)

{

System.out.println("Enter details of manager : id, name, deptId, basicSalary, perfmonceBonus ");

emps[counter++] = new Manager(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextDouble(), sc.nextDouble());

}

else

{

System.out.println("Cannot recruit.");

}

break;

case 2 :

if (counter >= 0 && counter < emps.length)

{

System.out.println("Enter details of worker : id, name, deptId, basicSalary, hoursWorked, hourlyRate ");

emps[counter++] = new Worker(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextDouble(), sc.nextDouble(), sc.nextDouble());

}

else

{

System.out.println("Cannot recruit.");

}

break;

case 3 :

for (Employee e : emps)

{

if(e != null)

{

System.out.println(e);

System.out.println("Net Salary : "+e.computeNetSalary());

if (e instanceof Manager)

{

System.out.println("Performance Bonus : "+((Manager)e).getPerfmonceBonus());

}

else if (e instanceof Worker)

{

System.out.println("Hourly Rate : "+((Worker)e).getHourlyRate());

}

}

}

break;

case 10 :

exit = true;

break;

}

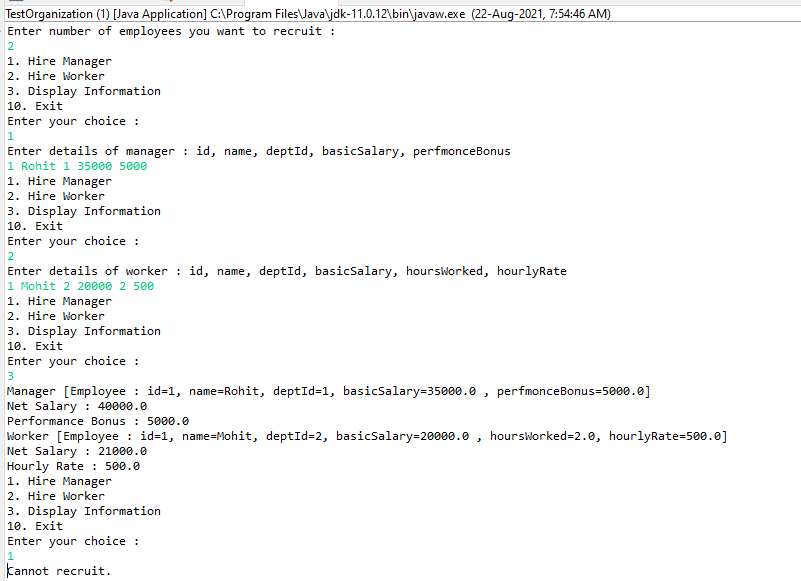
}

sc.close();

}

}

**Output.java**

****

**1)Vehicle.java**

/\* Create a class Vehicle , under the package : com.app.vehicles , to represent any vehicle.

tight encapsulation

state(data members) : registrationNo : int , color : String , price : double

registrationNo MUST be unique (non duplicate) for any vehicle (primary Key : unique ID)

Override toString to return complete state.

DO NOT override equals

\*/

package com.app.vehicles;

public class Vehicle {

private int registrationNo;

private String color;

private double price;

public Vehicle(int registrationNo, String color, double price)

{

super();

this.registrationNo = registrationNo;

this.color = color;

this.price = price;

}

@Override

public String toString() {

return "Vehicle [registrationNo=" + registrationNo + ", color=" + color + ", price=" + price + "]";

}

public boolean equals(Object anotherObject)

{

//unique ID for testing equality : regNo (Primary key)

System.out.println("in vehicle equals");

if(anotherObject instanceof Vehicle)

return this.registrationNo == ((Vehicle)anotherObject).registrationNo;

return false;

}

}

**2)TestVehicle.java**

/\*Create a class TestVehicle under the package "tester"

Accept details of 2 vehicles from user .

Display "SAME" or "DIFFERENT" , in case of same or different reg nos.

(try to invoke equals : inherited from Object class : v1.equals(v2))

\*/

package tester;

import java.util.Scanner;

import com.app.vehicles.Vehicle;

public class TestVehicle {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter 1st vehicle details : registrationNo, color, price");

Vehicle v1 = new Vehicle(sc.nextInt(), sc.next(), sc.nextDouble());

System.out.println("Enter 2nd vehicle details : registrationNo, color, price");

Vehicle v2 = new Vehicle(sc.nextInt(), sc.next(), sc.nextDouble());

System.out.println(v1.equals(v2) ? "Vehicles are SAME" : "Vehicles are DIFFERENT");

// System.out.println(v1.hashCode()+" "+v2.hashCode());

sc.close();

}

}

**3)TestVehicle2.java**

package tester;

import java.util.Scanner;

import com.app.vehicles.Vehicle;

public class TestVehicle2 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter 1st vehicle details: registrationNo, color, price");

Object v1=new Vehicle(sc.nextInt(),sc.next(),sc.nextDouble());

System.out.println("Enter 2nd vehicle details: registrationNo, color, price");

Object v2=new Vehicle(sc.nextInt(),sc.next(),sc.nextDouble());

System.out.println(v1.equals(v2)?"SAME":"DIFFERENT");

// System.out.println(v1.hashCode()+" "+v2.hashCode());

String s = new String("hello");

System.out.println(v1.equals(s));

sc.close();

}

}

**4)TestVehicle3.java**

package tester;

import java.util.Scanner;

import com.app.vehicles.Vehicle;

public class TestVehicle3 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter 1st vehicle details: registrationNo, color, price");

Object v1=new Vehicle(sc.nextInt(),sc.next(),sc.nextDouble());

System.out.println("Enter 2nd vehicle details: registrationNo, color, price");

Object v2=new Vehicle(sc.nextInt(),sc.next(),sc.nextDouble());

System.out.println(v1.equals(v2)?"SAME":"DIFFERENT");

// System.out.println(v1.hashCode()+" "+v2.hashCode());

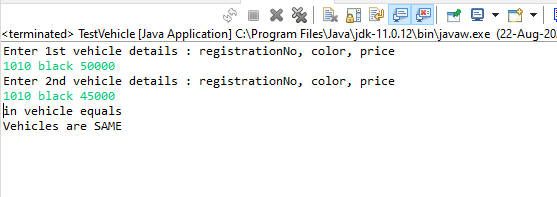
System.out.println(v1.equals(null));

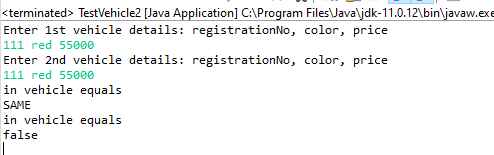
sc.close();

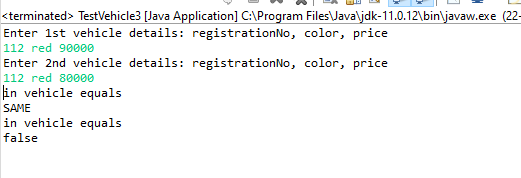
}

}

**Output :**

****

****

****

**1)CovariantReturn.java**

**package** covariance\_demo;

**class** Grain

{

**public** String toString()

{

**return** "Grain";

}

}

**class** Wheat **extends** Grain

{

**public** String toString()

{

**return** "Wheat";

}

}

**class** Mill

{

Grain process()

{

**return** **new** Grain();

}

}

**class** WheatMill **extends** Mill

{

Wheat process()

{

**return** **new** Wheat();

}

}

**public** **class** CovariantReturn {

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

Mill m = **new** Mill();

// System.out.println(m);

Grain g = m.process();

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

m = **new** WheatMill();

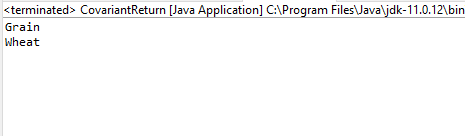
g = m.process();

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

}

}

**Output :**



**1)Vehicle.java**

/\* Create a class Vehicle , under the package : com.app.vehicles , to represent any vehicle.

tight encapsulation

state(data members) : registrationNo : int , color : String , price : double

registrationNo MUST be unique (non duplicate) for any vehicle (primary Key : unique ID)

Override toString to return complete state.

DO NOT override equals

\*/

package com.app.vehicles;

public class Vehicle {

private int registrationNo;

private String color;

private double price;

public Vehicle(int registrationNo, String color, double price)

{

super();

this.registrationNo = registrationNo;

this.color = color;

this.price = price;

}

public Vehicle(int registrationNo, String color) {

super();

this.registrationNo = registrationNo;

this.color = color;

}

@Override

public String toString() {

return "Vehicle [registrationNo=" + registrationNo + ", color=" + color + ", price=" + price + "]";

}

public boolean equals(Object anotherObject)

{

// unique ID for testing equality : regNo (Primary key) & clr

System.out.println("in vehicle equals...");

if (anotherObject instanceof Vehicle)

{

//Vehicle v = ((Vehicle) anotherObject);//down casting

// return this.registrationNo == v.registrationNo && color.equals(v.color);

return this.registrationNo == ((Vehicle)anotherObject).registrationNo && this.color.equals(((Vehicle)anotherObject).color);

}

return false;

}

}

**2)** **VehicleHandlingException.java**

**package** custom\_exception;

@SuppressWarnings("serial")

**public** **class** VehicleHandlingException **extends** Exception{

**public** VehicleHandlingException(String errMsg)

{

**super**(errMsg);

}

}

**3)** **ValidationsRules.java**

package utils;

import com.app.vehicles.Vehicle;

import custom\_exception.VehicleHandlingException;

public class ValidationRules {

//add static method to check for dup vehicle

public static Vehicle checkForDuplicate(Vehicle[] vehicleData, int registrationNo, String color, double price ) throws VehicleHandlingException

{

//create vehicle type of object holding registrationNo n color

Vehicle newVehicle = new Vehicle(registrationNo, color, price);//allfields

//for-each : equals of Vehicle class : Vehicle ref

for(Vehicle v : vehicleData)

{

if(v != null)

{

if(v.equals(newVehicle))

{

throw new VehicleHandlingException("Duplicate Vehicle Detected");

}

}

}

System.out.println("No Duplicate Vehicle Detected");

return newVehicle;

}

}

**4)** **VehicleShowroom.java**

/\*Create a Tester : VehicleShowroom

Create suitable data structure to store vehicle information.

Add Options

1. Add new vehicle :

It should add a vehicle iff it's not duplicate.

2. Display all vehicles

3. Exit\*/

**package** tester;

**import** java.util.Scanner;

**import** com.app.vehicles.Vehicle;

**import** custom\_exception.VehicleHandlingException;

**import** **static** utils.ValidationRules.*checkForDuplicate*;

**public** **class** VehicleShowroom {

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

**try**(Scanner sc = **new** Scanner(System.***in***))

{

System.***out***.println("Enter Size of Show-room : ");

Vehicle[] vehicles = **new** Vehicle[sc.nextInt()];

**boolean** exit = **false**;

**int** counter = 0;

**while**(exit != **true**)

{

**try**

{

System.***out***.println("1. Add new vehicle\n2. Display all vehicles\n3. Exit");

System.***out***.println("Enter your choice : ");

**switch**(sc.nextInt())

{

**case** 1 :

**if**(counter >=0 && counter < vehicles.length)

{

System.***out***.println("Enter Vehicle details: Registration\_Number Color Price");

//Tester has to invoke method of ValidationRules : static

Vehicle v = *checkForDuplicate*(vehicles,sc.nextInt(),sc.next(),sc.nextDouble());

//not dup

vehicles[counter++] = v;

System.***out***.println("Vehicle details added....");

}

**else**

**throw** **new** VehicleHandlingException("Showroom is full!!!!");

**break**;

**case** 2 :

System.***out***.println("Showroom Details");

**for**(Vehicle v : vehicles)

**if**(v != **null**)

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

**break**;

**case** 3 :

exit = **true**;

**break**;

}

}

**catch**(Exception e)

{

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

e.printStackTrace();

}

}

}

**finally**

{

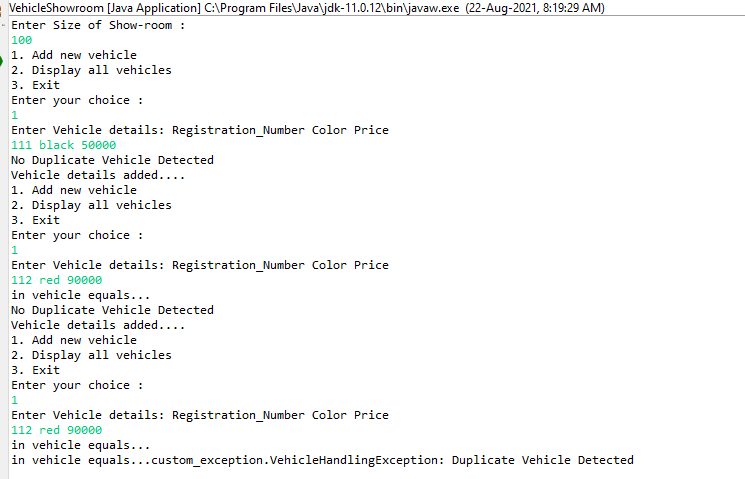
System.***out***.println("main over");

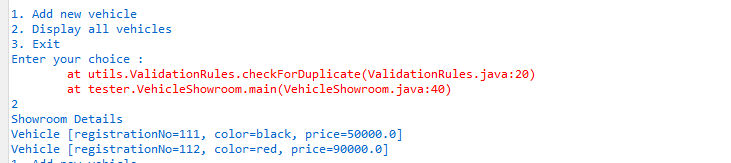
}

}

}

**Output :**

****

****

**1)Customer.java**

/\*Create Java application for customer management system.

4.1 Customer : name(string),email(string),password(string),registrationAmount(double),dob(Date)

Add/generate suitable constructor & toString

Unique ID : email (where will you use this fact ?????????????????) : equals

Will you add any other data member in Customer class for parsing n formatting? YES

HOW ? : static data member : SDF

\*/

package com.app.customers;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Customer {

private String name, email, password;

private double registrationAmount;

private Date dob;

public static SimpleDateFormat sdf;//concrete class of date

static

{

sdf = new SimpleDateFormat("dd-MM-yyyy");//dd-MMM-yyyy//MMM for printing Jan , Feb...

}

public Customer(String name, String email, String password, double registrationAmount, Date dob)

{

super();

this.name = name;

this.email = email;

this.password = password;

this.registrationAmount = registrationAmount;

this.dob = dob;

}

@Override

public String toString()

{

return "Customer [name=" + name + ", email=" + email + ", password=" + password + ", registrationAmount="

+ registrationAmount + ", dob=" + sdf.format(dob) + "]";

}

@Override

public boolean equals(Object anotherObject)

{

System.out.println("in customer equal method");

if(anotherObject instanceof Customer)

{

//Customer c1=(Customer)cust;

//return this.emailId.equals(c1.emailId);

return this.email.equals(((Customer) anotherObject).email);

}

return false;

}

}

**2)** **CustomerHandlingException.java**

**package** custom\_exception;

@SuppressWarnings("serial")

**public** **class** CustomerHandlingException **extends** Exception {

**public** CustomerHandlingException(String errMsg)

{

**super**(errMsg);

}

}

**3)** **ValidationRules.java**

package utils;

import java.text.ParseException;

import java.util.Date;

import custom\_exception.CustomerHandlingException;

import static com.app.customers.Customer.sdf;

public class ValidationRules {

public static final int MIN\_LENGTH, MAX\_LENGTH;

public static Date thresholdDate;

static

{

MIN\_LENGTH = 4;

MAX\_LENGTH = 10;

try

{

thresholdDate = sdf.parse("1-1-1995"); //String ----> date conversion

}

catch(ParseException e)

{

System.out.println("Error in static init block"+e);

}

}

public static String validateEmail(String email) throws CustomerHandlingException

{

if(email.contains("@") && email.endsWith(".com"))

return email;

else

throw new CustomerHandlingException("Invalid email format !!!");

}

public static String validatePassword(String password) throws CustomerHandlingException

{

if(password.length() >= MIN\_LENGTH && password.length() <= MAX\_LENGTH)

return password;

else

throw new CustomerHandlingException("Invalid Password Format !!!");

}

public static double validateRegistrationAmount(double registrationAmount) throws CustomerHandlingException

{

if(registrationAmount % 500 == 0)

return registrationAmount;

else

throw new CustomerHandlingException("Amount must be multiple of 500");

}

public static Date convertDate(String dob) throws CustomerHandlingException, ParseException

{

Date d = sdf.parse(dob); //String ----> date conversion

if(d.before(thresholdDate))

return d;

else

throw new CustomerHandlingException("Invalid date");

}

}

**2)** **TestCustomer.java**

package tester;

import java.util.Scanner;

import com.app.customers.Customer;

import static utils.ValidationRules.\*;

public class TestCustomer {

public static void main(String[] args) {

try(Scanner sc = new Scanner(System.in))

{

System.out.println("Enter customer details : name, email, password, registrationAmount, dob(dd-MM-yyyy)");

Customer c = new Customer(sc.next(), validateEmail(sc.next()), validatePassword(sc.next()),validateRegistrationAmount(sc.nextDouble()),convertDate(sc.next()));

System.out.println(c.toString());

}

catch(Exception e)

{

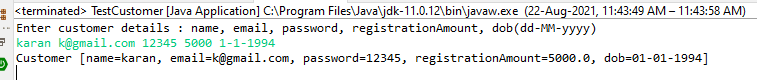
e.printStackTrace();

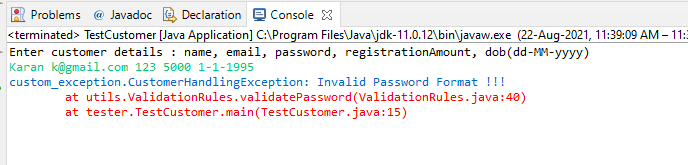
}

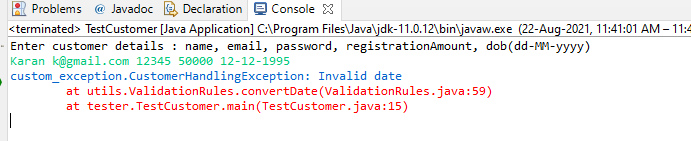
}

}

**Output :**

****

****

****

**1)Customer.java**

/\*Create Java application for customer management system.

4.1 Customer : name(string),email(string),password(string),registrationAmount(double),dob(Date)

Add/generate suitable constructor & toString

Unique ID : email (where will you use this fact ?????????????????) : equals

Will you add any other data member in Customer class for parsing n formatting? YES

HOW ? : static data member : SDF

\*/

package com.app.customers;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Customer {

private String name, email, password;

private double registrationAmount;

private Date dob;

//HAS-A : enum (weaker form of association)

private CustomerCategory category;

public static SimpleDateFormat sdf;//concrete class of date

static

{

sdf = new SimpleDateFormat("dd-MM-yyyy");//dd-MMM-yyyy//MMM for printing Jan , Feb...

}

public Customer(String name, String email, String password, double registrationAmount, Date dob, CustomerCategory category)

{

super();

this.name = name;

this.email = email;

this.password = password;

this.registrationAmount = registrationAmount;

this.dob = dob;

this.category = category;

}

@Override

public String toString()

{

return "Customer [name=" + name + ", email=" + email + ", password=" + password + ", registrationAmount="

+ registrationAmount + ", dob=" + sdf.format(dob) + "customer category="+category+"]";

}

@Override

public boolean equals(Object anotherObject)

{

System.out.println("in customer equal method");

if(anotherObject instanceof Customer)

{

//Customer c1=(Customer)cust;

//return this.emailId.equals(c1.emailId);

return this.email.equals(((Customer) anotherObject).email);

}

return false;

}

}

**2)CustomerCategory.java**

**package** com.app.customers;

**public** **enum** CustomerCategory //extends java.lang.Enum

{

***ILVER***, ***GOLD***, ***DIAMOND***, ***PLATINUM***;

//Can you override toString/equals/compareTo : toString

@Override

**public** String toString()

{

**return** name().toLowerCase();

}

}

**3)CustomerHandlingException.java**

**package** custom\_exception;

@SuppressWarnings("serial")

**public** **class** CustomerHandlingException **extends** Exception {

**public** CustomerHandlingException(String errMsg)

{

**super**(errMsg);

}

}

**4)ValidationRules.java**

package utils;

import java.text.ParseException;

import java.util.Date;

import com.app.customers.CustomerCategory;

import custom\_exception.CustomerHandlingException;

import static com.app.customers.Customer.sdf;

public class ValidationRules {

public static final int MIN\_LENGTH, MAX\_LENGTH;

public static Date thresholdDate;

static

{

MIN\_LENGTH = 4;

MAX\_LENGTH = 10;

try

{

thresholdDate = sdf.parse("1-1-1995"); //String ----> date conversion

}

catch(ParseException e)

{

System.out.println("Error in static init block"+e);

}

}

public static String validateEmail(String email) throws CustomerHandlingException

{

if(email.contains("@") && email.endsWith(".com"))

return email;

else

throw new CustomerHandlingException("Invalid email format !!!");

}

public static String validatePassword(String password) throws CustomerHandlingException

{

if(password.length() >= MIN\_LENGTH && password.length() <= MAX\_LENGTH)

return password;

else

throw new CustomerHandlingException("Invalid Password Format !!!");

}

public static double validateRegistrationAmount(double registrationAmount) throws CustomerHandlingException

{

if(registrationAmount % 500 == 0)

return registrationAmount;

else

throw new CustomerHandlingException("Amount must be multiple of 500");

}

public static Date convertDate(String dob) throws CustomerHandlingException, ParseException

{

Date d = sdf.parse(dob); //String ----> date conversion

if(d.before(thresholdDate))

return d;

else

throw new CustomerHandlingException("Invalid date");

}

public static CustomerCategory validateCategory(String category) throws CustomerHandlingException

{

try {

return CustomerCategory.valueOf(category.toUpperCase());

}

catch (IllegalArgumentException e) {

StringBuilder sb = new StringBuilder();

sb.append("Enter valid customer category : ");

throw new CustomerHandlingException("Invalid customer category ");

}

}

}

**5)TestCustomer.java**

package tester;

import java.util.Scanner;

import com.app.customers.Customer;

import static utils.ValidationRules.\*;

public class TestCustomer {

public static void main(String[] args) {

try(Scanner sc = new Scanner(System.in))

{

System.out.println("Enter customer details : name, email, password, registrationAmount, dob(dd-MM-yyyy), customerCategory");

Customer c = new Customer(sc.next(), validateEmail(sc.next()), validatePassword(sc.next()),validateRegistrationAmount(sc.nextDouble()),convertDate(sc.next()), validateCategory(sc.next()));

System.out.println(c.toString());

}

catch(Exception e)

{

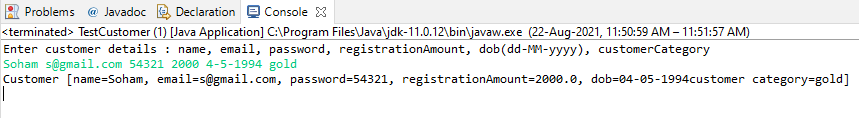
e.printStackTrace();

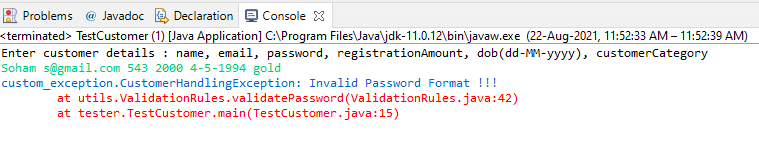
}

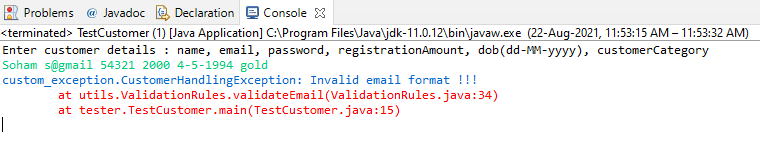
}

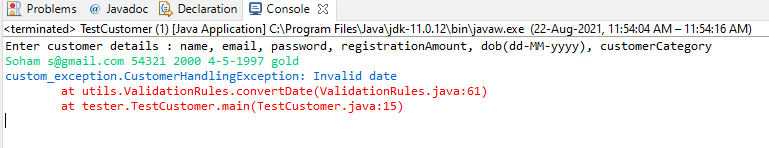
}

**Output :**

****

****

****

****

**Day 10.1**

**1) Customer.java**

**package** customer\_management;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

/\*Customer : name(string),email(string),password(string),registrationAmount(double),dob(Date)

Add/generate suitable constructor & toString

Unique ID : email (where will you use this fact ?????????????????) : equals

Will you add any other data member in Customer class for parsing n formatting? YES

HOW ? : static data member : SDF

\*/

**public** **class** Customer {

**private** String name,email,password;

**private** **double** registrationAmount;

**private** Date dateOfBirth;

**private** CustomerCategory category;

**private** AdharCard adharCard;

//add single copy of the SDF

**public** **static** SimpleDateFormat *sdf*; //converter (string--->Date and Date---> string)

**static**

{

*sdf*=**new** SimpleDateFormat("dd-MM-yyyy");

}

//constructor

**public** Customer(String name,String email,String password,**double** registrationAmount,

Date dateOfBirth,CustomerCategory category)

{

**super**();

**this**.name=name;

**this**.email=email;

**this**.password=password;

**this**.registrationAmount=registrationAmount;

**this**.dateOfBirth=dateOfBirth;

**this**.category=category;

}

//toString

@Override

**public** String toString() {

StringBuilder sb=**new** StringBuilder("");

**if**(adharCard==**null**)

sb.append(" Adhar card not yet linked");

**else**

sb.append(adharCard);

**return** "Customer Details: name=" + name + ", email=" + email + ", password=" + password + ", registrationAmount="

+ registrationAmount + ", dateOfBirth=" + *sdf*.format(dateOfBirth)+

" category="+category + ""+sb.toString();

}

//equals

@Override

**public** **boolean** equals(Object O)

{

System.***out***.println("in Customer equals");

**if**(O **instanceof** Customer)

{

//Customer c= (Customer) O; //downcast

**return** **this**.email.equals(((Customer)O).email);

}

**return** **false**;

}

//add nonstactic method to link adhar card details

**public** **void** linkAdharCard(String uid,String createDate) **throws** ParseException {

**this**.adharCard=**new** AdharCard(uid,*sdf*.parse(createDate));

}

//nested inner class

**class** AdharCard

{

**private** String adharNo;

**private** Date createdOn;

**public** AdharCard(String adharNo, Date createdOn) {

**super**();

**this**.adharNo = adharNo;

**this**.createdOn = createdOn;

}

@Override

**public** String toString() {

**return** " AdharCard [adharNo=" + adharNo + ", createdOn=" + *sdf*.format(createdOn) + "]";

}

}

}

**2) CustomerCategory.java**

**package** customer\_management;

/\*Create CustomerCategory as an enum.

Add these constants : SILVER,GOLD,DIAMOND,PLATINUM

\*/

**public** **enum** CustomerCategory {

***SILVER***,***GOLD***,***DIAMOND***,***PLATINUM***;

**public** String toString()

{

**return** name().toLowerCase();

}

}

**3) CustomerHandlingException.java**

/\*Create custom exception class in a separate package

CustomerHandlingException

\*/

**package** CustomerHandlingException;

@SuppressWarnings("serial")

**public** **class** CustomException **extends** Exception{

**public** CustomException(String errMessage)

{

**super**(errMessage);

}

}

**4) ValidationRules.java**

/\*Rules

email must contain "@" & should be from ".com" domain

password must be min 4 max 10 chars long

Add 2 different static methods : validateEmail n validatePassword

Add a static method here to parse String---> Date

eg : convertDate

\*/

**package** utils;

**import** **static** customer\_management.Customer.*sdf*;

**import** java.text.ParseException;

**import** java.util.Arrays;

**import** java.util.Date;

**import** CustomerHandlingException.CustomException;

**import** customer\_management.CustomerCategory;

**public** **class** ValidationRules {

**public** **static** **final** **int** ***MIN\_LENGTH***;

**public** **static** **final** **int** ***MAX\_LENGTH***;

**static** {

***MIN\_LENGTH***=4;

***MAX\_LENGTH***=10;

}

//validate email---> add static method

**public** **static** String validateEmail(String email) **throws** CustomException{

**if**(email.contains("@") && email.endsWith(".com"))

**return** email; //returns validated email

**else**

//throw throwable

**throw** **new** CustomException("invalid email !!!");

}

//validate password

**public** **static** String validatePassword(String password) **throws** CustomException{

**if**(password.length()>=***MIN\_LENGTH*** && password.length()<=***MAX\_LENGTH***)

**return** password;

**else**

**throw** **new** CustomException("Password must contain minimum 4 character and maximum 10 character");

}

//add static method to convert string to date

**public** **static** Date converteDate(String dateOfBirth) **throws** ParseException

{

**return** *sdf*.parse(dateOfBirth);

}

// validate customerCategory

**public** **static** CustomerCategory validateCategory(String category) **throws** CustomException

{

**try** {

//public static CustomerCategory valueOf(String name) throw IllegalargumentExc

**return** CustomerCategory.*valueOf*(category.toUpperCase());

}**catch** (IllegalArgumentException e) {

StringBuilder sb=**new** StringBuilder("Invalid category chosen \n");

sb.append("Valid category");

sb.append(Arrays.*toString*(CustomerCategory.*values*()));

**throw** **new** CustomException(sb.toString());

}

}

}

**5) CustomerTester.java**

**package** tester1;

**import** java.util.Scanner;

**import** customer\_management.Customer;

**import** **static** utils.ValidationRules.\*;

**public** **class** CustomerTester {

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

**try**(Scanner sc=**new** Scanner(System.***in***))

{

System.***out***.println("Enter Customer details: name,email,password,registration amount,"

+ "date(dd-MM-yyyy),category");

Customer cust=**new** Customer(sc.next(),*validateEmail*(sc.next()),*validatePassword*(sc.next()),

sc.nextDouble(),*converteDate*(sc.next()),*validateCategory*(sc.next()));

System.***out***.println("Customer details: "+cust);

}**catch** (Exception e) {

//e.printStackTrace();

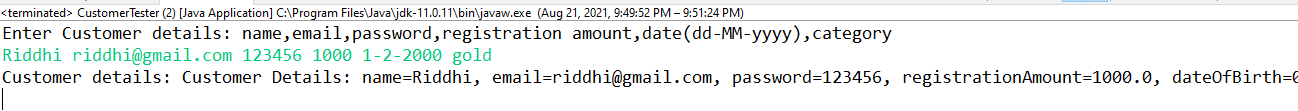
e.getMessage();

}

}

}

**Output:**

****

**7) Tester2.java**

**package** tester1;

**import** **static** utils.ValidationRules.*converteDate*;

**import** **static** utils.ValidationRules.*validateCategory*;

**import** **static** utils.ValidationRules.*validateEmail*;

**import** **static** utils.ValidationRules.*validatePassword*;

**import** java.util.Scanner;

**import** customer\_management.Customer;

**public** **class** Tester2 {

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

**try**(Scanner sc=**new** Scanner(System.***in***))

{

System.***out***.println("Enter Customer details: name,email,password,registration amount,"

+ "date(dd-MM-yyyy),category");

Customer cust=**new** Customer(sc.next(),*validateEmail*(sc.next()),*validatePassword*(sc.next()),

sc.nextDouble(),*converteDate*(sc.next()),*validateCategory*(sc.next()));

System.***out***.println("Enter adharcard details: adhar no,created date(day-month-year)");

cust.linkAdharCard(sc.next(),sc.next());

System.***out***.println("Customer details: "+cust);

}**catch** (Exception e) {

//e.printStackTrace();

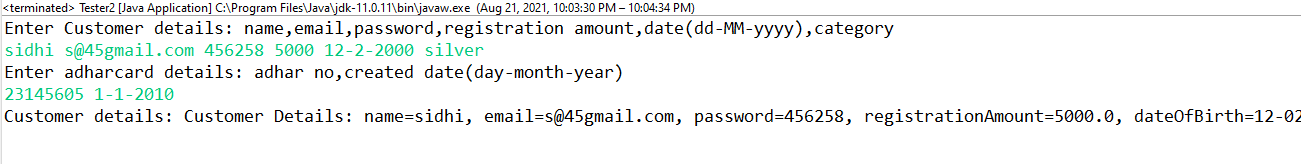
e.getMessage();

}

}

}

**Output:**

****

Create java application for Student admission management.

1.1 Student : prn (string : unique id) , name , email , password, course (enum) ,GPA(double 1---10) , dob(Date)

Add suitable constructor/s , toString , equals (as per the requirement)

1.2 Course (enum) : DBT,REACT,ANGULAR,REST,SPRING,HIBERNATE

Assign course capacity , along with enum constants.

Student HAS-A Course (one-to-one)

1.3 Student HAS-A ContactDetails (one-to-one)

Use aggregation (i.e a separate class to store ContactDetails)

ContactDetails : city , state , phoneNo

1.4 Create custom exception to alert in case of validation errors/ B.L failures.

1.5 Add validation rules

email : min 5 max 10 chars

password min 5 chars & must contain at least one of the special chars [@,#,$,%,\*...]

course : a valid course name having available capacity , to admit a new student

min GPA : 7

dob : Not exceeding 1st Jan 1999.

dup student validation

1.6 Create a tester : StudentAdmissionSystem

Choose suitable , growable data structure (ArrayList) for storing student information

Com.app.core

**Department.java**

**package** com.app.core;

**public** **enum** Department {

***RND***, ***HR***, ***MARKETING***, ***FINANCE***, ***SALES***

}

**Emp.java**

package com.app.core;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Emp {

//id , name , salary , deptId(enum),joinDate (Date)

private int id;

private String name;

private double salary;

private Department dept;

private Date joinDate;

public static SimpleDateFormat sdf;

static {

sdf = new SimpleDateFormat("dd-MM-yyyy");

}

public Emp(int id, String name, double salary, Department dept, Date joinDate) {

super();

this.id = id;

this.name = name;

this.salary = salary;

this.dept = dept;

this.joinDate = joinDate;

}

public Emp(int id) {

super();

this.id = id;

}

@Override

public String toString() {

return "Emp id=" + id + ", name=" + name + ", salary=" + salary

+ ", dept=" + dept + ", joinDate="

+ sdf.format(joinDate);

}

@Override

public boolean equals(Object o)

{

System.out.println("in emp equals");

if(o instanceof Emp)

return this.id==((Emp)o).id;

return false;

}

}

**EmpHandlingException**

**package** custom\_exception;

**public** **class** EmpHandlingException **extends** Exception {

**public** EmpHandlingException(String mesg) {

**super**(mesg);

}

}

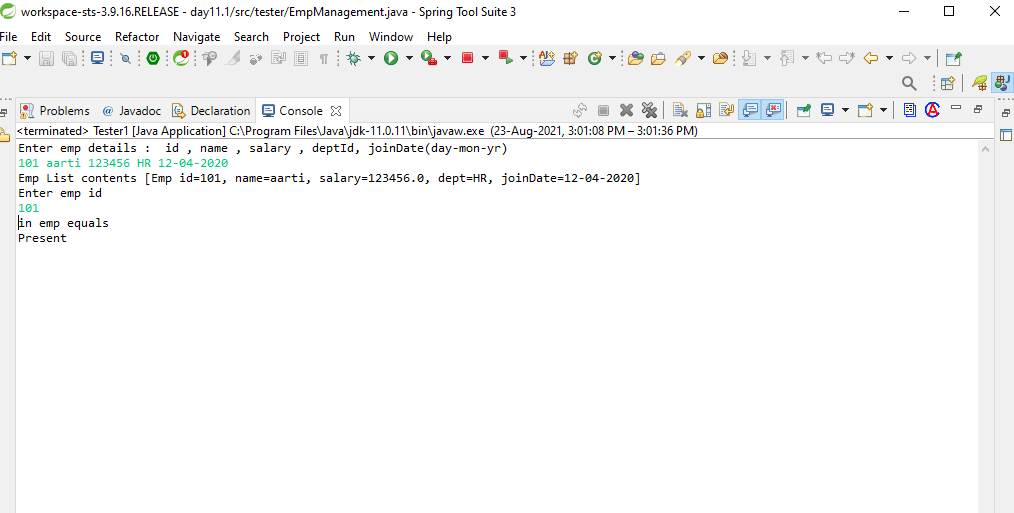
**EmpManagement.java**

**package** tester;

**public** **class** EmpManagement {

}

**Output:**



**Tester1**

package tester;

import java.util.ArrayList;

import java.util.Scanner;

import com.app.core.Emp;

import static com.app.core.Department.\*;

import static com.app.core.Emp.sdf;

/\*

\* 1. Create empty AL to store emp info.

Emp : id , name , salary , deptId(enum),joinDate (Date)

\*/

public class Tester1 {

public static void main(String[] args) {

try (Scanner sc = new Scanner(System.in)) {

// Create empty AL to store emp info.

ArrayList<Emp> empList = new ArrayList<>();// size=0,capa=10

// 2. Accept emp details from user n store it in AL

System.out.println("Enter emp details : id , name , salary , deptId, joinDate(day-mon-yr)");

// AL API : add

empList.add(new Emp(sc.nextInt(), sc.next(), sc.nextDouble(), valueOf(sc.next().toUpperCase()),

sdf.parse(sc.next())));

//AL : toString

System.out.println("Emp List contents "+empList);

System.out.println("Enter emp id");

Emp e=new Emp(sc.nextInt());

System.out.println(empList.contains(e)?"Present":"Absent");

} catch (Exception e) {

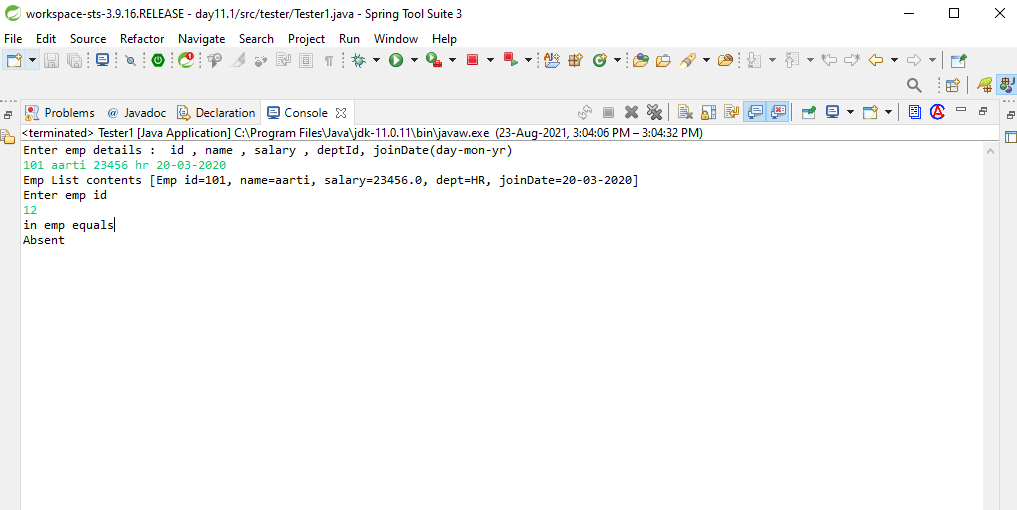
e.printStackTrace();

}

}

}

**Output:**



**tester2.java**

package tester;

import java.util.ArrayList;

import java.util.Scanner;

import com.app.core.Emp;

import custom\_exception.EmpHandlingException;

import static com.app.core.Department.\*;

import static com.app.core.Emp.sdf;

/\*

\* 1. Create empty AL to store emp info.

Emp : id , name , salary , deptId(enum),joinDate (Date)

\*/

public class Tester2 {

public static void main(String[] args) {

try (Scanner sc = new Scanner(System.in)) {

// Create empty AL to store emp info.

ArrayList<Emp> empList = new ArrayList<>();// size=0,capa=10

// 2. Accept emp details from user n store it in AL

System.out.println("Enter emp details : id , name , salary , deptId, joinDate(day-mon-yr)");

// AL API : add

empList.add(new Emp(sc.nextInt(), sc.next(), sc.nextDouble(), valueOf(sc.next().toUpperCase()),

sdf.parse(sc.next())));

//AL : toString

System.out.println("Emp List contents "+empList);

System.out.println("Enter emp id");

Emp e=new Emp(sc.nextInt());

System.out.println(empList.contains(e)?"Present":"Absent");

int index=empList.indexOf(e);

if(index==-1)

throw new EmpHandlingException("Emp not found");

System.out.println("emp details"+empList.get(index));

} catch (Exception e) {

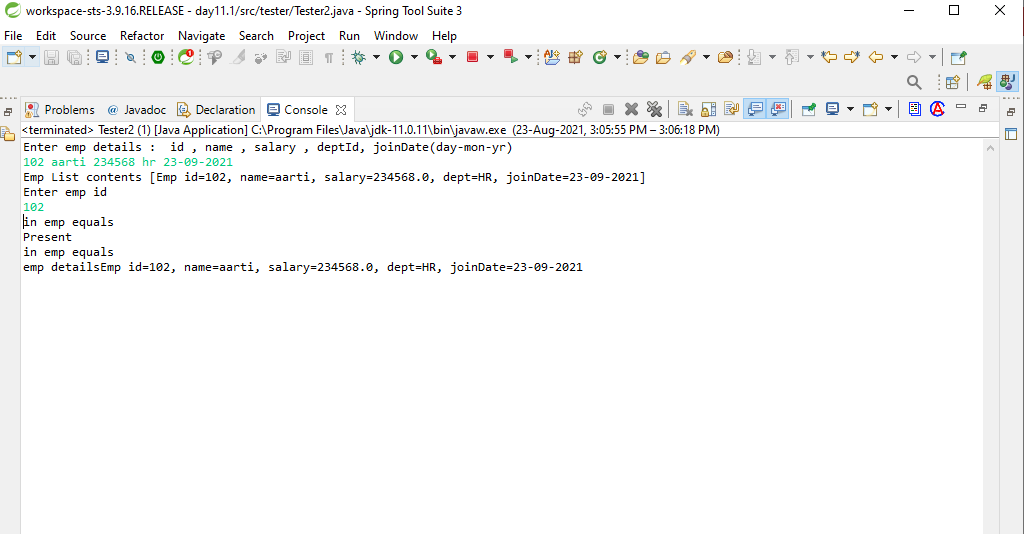
e.printStackTrace();

}

}

}

**Output:**



**1) Book.java**

**package** com.app.core;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**public** **class** Book **implements** Comparable<Book>{

**private** String title;

**private** String author;

**private** **double** price;

**private** Category category;

**private** **int** quantity;

**private** Date publish\_date;

**public** **static** SimpleDateFormat *sdf*;

**static** {

*sdf*=**new** SimpleDateFormat("dd-MM-yyyy");

}

//constructor

**public** Book(String title,String author,**double** price,Category category,**int** quantity,Date publish\_date)

{

**super**();

**this**.title=title;

**this**.author=author;

**this**.category=category;

**this**.price=price;

**this**.quantity=quantity;

**this**.publish\_date=publish\_date;

}

//constructor

**public** Book(String title)

{

**super**();

**this**.title=title;

}

//tostringMethod

@Override

**public** String toString() {

**return** "title=" + title + ", author=" + author + ", price=" + price + ", category=" + category

+ ", quantity=" + quantity + ", publish\_date=" + *sdf*.format(publish\_date);

}

//equals method

@Override

**public** **boolean** equals(Object o)

{

System.***out***.println("In equals method.........");

**if**(o **instanceof** Book)

**return** **this**.title.equals(((Book)o).title); //typecasting

**return** **false**;

}

@Override

**public** **int** compareTo(Book anotherBook)

{

System.***out***.println("in compareTo");

//sorting criteria : emp id

**int** s=**this**.title.compareTo(anotherBook.getTitle());

**if**(s<0)

**return** -1;

**if**(**this**.title == anotherBook.getTitle())

**return** 0;

**return** 1;

}

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

**public** String getAuthor() {

**return** author;

}

**public** **void** setAuthor(String author) {

**this**.author = author;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

**public** Category getCategory() {

**return** category;

}

**public** **void** setCategory(Category category) {

**this**.category = category;

}

**public** **int** getQuantity() {

**return** quantity;

}

**public** **void** setQuantity(**int** quantity) {

**this**.quantity = quantity;

}

**public** Date getPublish\_date() {

**return** publish\_date;

}

**public** **void** setPublish\_date(Date publish\_date) {

**this**.publish\_date = publish\_date;

}

**public** **static** SimpleDateFormat getSdf() {

**return** *sdf*;

}

**public** **static** **void** setSdf(SimpleDateFormat sdf) {

Book.*sdf* = sdf;

}

}

**2) Category.java**

**package** com.app.core;

**public** **enum** Category {

***SCIENCE***,***NOVELS***,***HISTORY***,***COMICS***,***POETRY***,***BUSINESS***,***PROGRAMMING***;

}

**3) BookHandlingException.java**

**package** custom\_exception;

**public** **class** BookHandlingException **extends** Exception{

**public** BookHandlingException(String mesg) {

**super**(mesg);

}

}

**4) CollectionUtils.java**

**package** utils;

**import** java.text.ParseException;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** **static** com.app.core.Book.*sdf*;

**import** com.app.core.Book;

**import** **static** com.app.core.Category.\*;

**public** **class** CollectionUtils {

**public** **static** ArrayList<Book> populateBooksData() **throws** ParseException

{

ArrayList<Book> list=**new** ArrayList<>();

//String title,String author,double price,Category category,int quantity,Date publish\_date

Book b1=**new** Book("Let\_Us\_C","Y.Kanetkar",200,***PROGRAMMING***,10,*sdf*.parse("1-1-2000"));

Book b2=**new** Book("Ancient\_India","Upinder Singh",600,***HISTORY***,30,*sdf*.parse("1-1-2009"));

Book b3=**new** Book("General\_Science","Sachine Bhaske",300,***SCIENCE***,50,*sdf*.parse("2-9-2001"));

Book b4=**new** Book("Dracula","Bram Stoker",300,***NOVELS***,15,*sdf*.parse("12-2-1999"));

Book b5=**new** Book("Origins\_of\_Marvel\_Comics","Stan Lee",350,***COMICS***,30,*sdf*.parse("21-7-2004"));

list.add(b1);

list.add(b2);

list.add(b3);

list.add(b4);

list.add(b5);

**return** list;

}

}

**5) BookManagement.java**

**package** tester;

**import** **static** utils.CollectionUtils.*populateBooksData*;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Date;

**import** java.util.Iterator;

**import** java.util.Scanner;

**import** utils.CollectionUtils;

**import** com.app.core.Book;

**import** custom\_exception.BookHandlingException;

**import** **static** com.app.core.Book.*sdf*;

**import** **static** com.app.core.Category.*valueOf*;

**public** **class** BookManagement {

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

**try** (Scanner sc = **new** Scanner(System.***in***)) {

ArrayList<Book> books = *populateBooksData*();

**boolean** exit = **false**;

**while** (!exit) {

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

System.***out***.println("Options: 1)display all book details, 2)Issue Out book, 3)Return book,"

+ "\n4)apply discount on old books, 5) sort books as per title. 100)Exit.");

System.***out***.print("Enter Your option: ");

**try** {

**switch** (sc.nextInt()) {

// display books

**case** 1:

Iterator<Book> itr = books.iterator();

**while** (itr.hasNext()) {

Book book = itr.next();

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

}

**break**;

/\*

\* 2. Issue out book i/p isbn & quantity o/p error mesg if book is not available

\* or insufficient quantity or success mesg with book quantity updated

\*/

**case** 2:

// ArrayList<Book> book1 = populateBooksData();

System.***out***.println("Enter title and quantity");

String title = sc.next();

**int** qty = sc.nextInt();

**int** index = books.indexOf(**new** Book(title));

**if** (index == -1)

**throw** **new** BookHandlingException("Invalid Book title");

Book book = books.get(index);

**if** (qty > book.getQuantity())

**throw** **new** BookHandlingException("Insufficient book quantity");

book.setQuantity(book.getQuantity() - qty);

System.***out***.println("book updated");

**break**;

// System.out.println("Enter title and quantity");

// String btitle=sc.next();

// int qty=sc.nextInt();

// for(Book b:books)

// {

// if(b.getTitle().equals(btitle))

// {

// if(qty>b.getQuantity())

// {

// System.out.println("Insufficient quantity of "+b.getTitle()+" book");

// }

//

// else if(qty<b.getQuantity())

// {

// b.setQuantity(b.getQuantity()-qty);

// System.out.println("Book issue done succefully");

// System.out.println(b);

// }

// }

// }

// break;

/\*

\* 3. Return book i/p isbn & quantity Update the quantity if book already exists

\* Otherwise ask for complete book details & then add the book.

\*/

**case** 3:

System.***out***.println("Enter title and quanitiy of book:");

String rtitle = sc.next();

**int** rqty = sc.nextInt();

**int** index1 = books.indexOf(**new** Book(rtitle));

**if** (index1 == -1) {

System.***out***.println("Enter your Book details to add book:");

System.***out***.println(

"String title,String author,double price,Category category,int quantity,Date publish\_Date");

books.add(**new** Book(sc.next(), sc.next(), sc.nextDouble(), *valueOf*(sc.next().toUpperCase()),

sc.nextInt(), *sdf*.parse(sc.next())));

**break**;

// throw new BookHandlingException( " these Book is not available");

}

System.***out***.println(index1);

Book book2 = books.get(index1);

System.***out***.println("Before return book: "+book2);

book2.setQuantity(book2.getQuantity() + rqty);

System.***out***.println("After return book: "+book2);

**break**;

/\*

\* 4. Apply discount on old books (i.e reduce price of books published before

\* specific date) i/p date & discount amount

\*/

**case** 4:

System.***out***.println("Specify date(dd-MM-yyyy)");

Date spdate = *sdf*.parse(sc.next());

**for** (Book b : books) {

**if** (b.getPublish\_date().before(spdate)) {

b.setPrice(b.getPrice() - 0.20 \* b.getPrice());

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

}

}

**break**;

// 5. Sort the books as per isbn.

**case** 5:

Collections.*sort*(books);

System.***out***.println("sorted booklist by title");

**for** (Book b : books)

System.***out***.println(b);// sorted o/p

**break**;

**case** 100:

exit = **true**;

**break**;

}

} **catch** (Exception e) {

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

}

}

} **catch** (Exception e) {

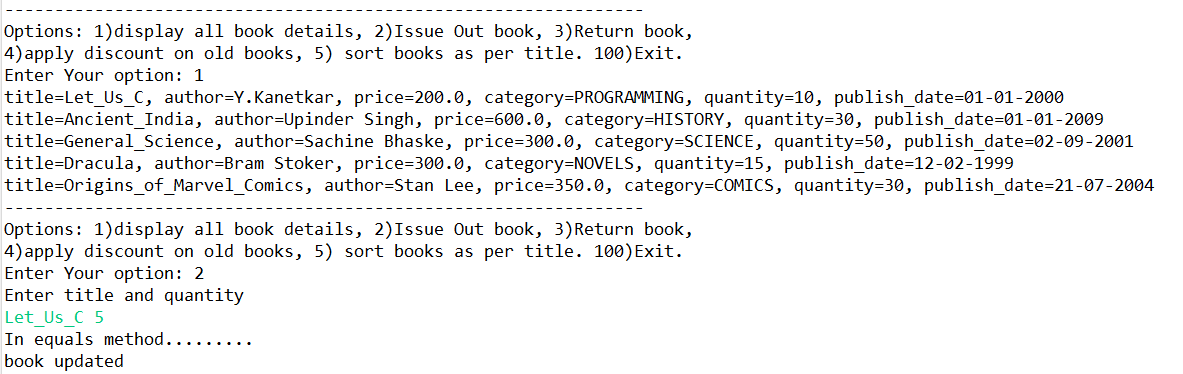
e.printStackTrace();

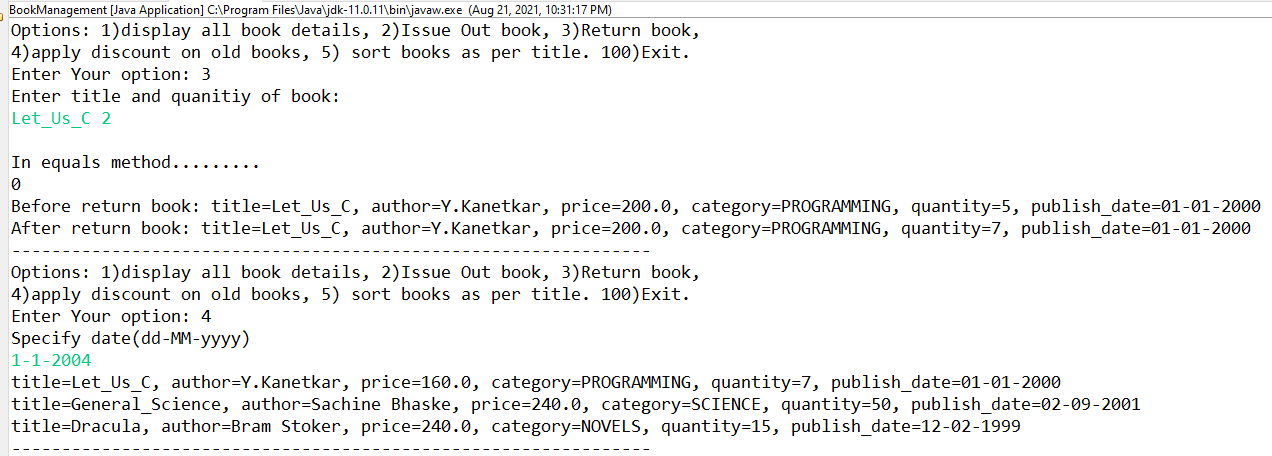
}

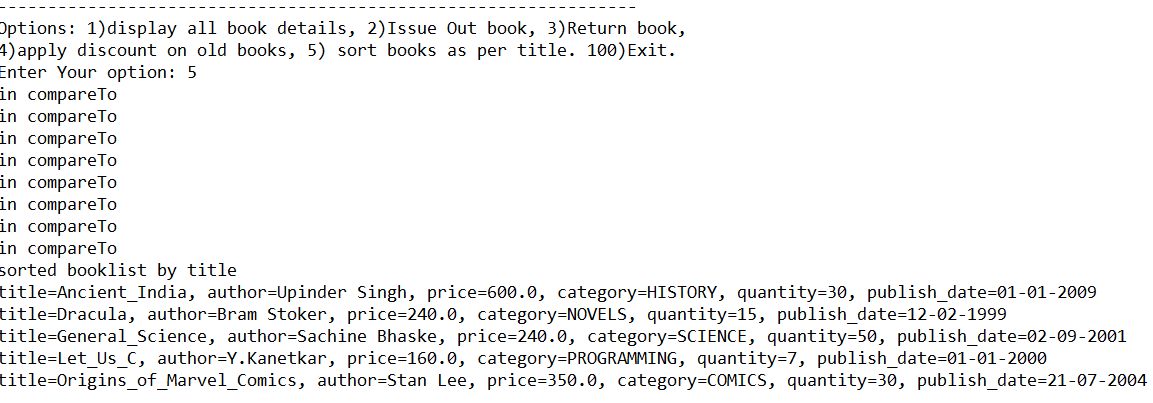
}

}

**Output:**

****

****

****

**1) Book.java**

**package** com.app.core;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**public** **class** Book **implements** Comparable<Book>{

**private** String title;

**private** String author;

**private** **double** price;

**private** Category category;

**private** **int** quantity;

**private** Date publish\_date;

**public** **static** SimpleDateFormat *sdf*;

**static** {

*sdf*=**new** SimpleDateFormat("dd-MM-yyyy");

}

//constructor

**public** Book(String title,String author,**double** price,Category category,**int** quantity,Date publish\_date)

{

**super**();

**this**.title=title;

**this**.author=author;

**this**.category=category;

**this**.price=price;

**this**.quantity=quantity;

**this**.publish\_date=publish\_date;

}

//constructor

**public** Book(String title)

{

**super**();

**this**.title=title;

}

//tostringMethod

@Override

**public** String toString() {

**return** "title=" + title + ", author=" + author + ", price=" + price + ", category=" + category

+ ", quantity=" + quantity + ", publish\_date=" + *sdf*.format(publish\_date);

}

//equals method

@Override

**public** **boolean** equals(Object o)

{

System.***out***.println("In equals method.........");

**if**(o **instanceof** Book)

**return** **this**.title.equals(((Book)o).title); //typecasting

**return** **false**;

}

@Override

**public** **int** compareTo(Book anotherBook)

{

System.***out***.println("in compareTo");

//sorting criteria : emp id

**int** s=**this**.title.compareTo(anotherBook.getTitle());

**if**(s<0)

**return** -1;

**if**(**this**.title == anotherBook.getTitle())

**return** 0;

**return** 1;

}

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

**public** String getAuthor() {

**return** author;

}

**public** **void** setAuthor(String author) {

**this**.author = author;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

**public** Category getCategory() {

**return** category;

}

**public** **void** setCategory(Category category) {

**this**.category = category;

}

**public** **int** getQuantity() {

**return** quantity;

}

**public** **void** setQuantity(**int** quantity) {

**this**.quantity = quantity;

}

**public** Date getPublish\_date() {

**return** publish\_date;

}

**public** **void** setPublish\_date(Date publish\_date) {

**this**.publish\_date = publish\_date;

}

**public** **static** SimpleDateFormat getSdf() {

**return** *sdf*;

}

**public** **static** **void** setSdf(SimpleDateFormat sdf) {

Book.*sdf* = sdf;

}

}

**2) Category.java**

**package** com.app.core;

**public** **enum** Category {

***SCIENCE***,***NOVELS***,***HISTORY***,***COMICS***,***POETRY***,***BUSINESS***,***PROGRAMMING***;

}

**3) BookHandlingException.java**

**package** custom\_exception;

**public** **class** BookHandlingException **extends** Exception{

**public** BookHandlingException(String mesg) {

**super**(mesg);

}

}

**4) CollectionUtils.java**

**package** utils;

**import** java.text.ParseException;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** **static** com.app.core.Book.*sdf*;

**import** com.app.core.Book;

**import** **static** com.app.core.Category.\*;

**public** **class** CollectionUtils {

**public** **static** ArrayList<Book> populateBooksData() **throws** ParseException

{

ArrayList<Book> list=**new** ArrayList<>();

//String title,String author,double price,Category category,int quantity,Date publish\_date

Book b1=**new** Book("Let\_Us\_C","Y.Kanetkar",200,***PROGRAMMING***,10,*sdf*.parse("1-1-2000"));

Book b2=**new** Book("Ancient\_India","Upinder Singh",600,***HISTORY***,30,*sdf*.parse("1-1-2009"));

Book b3=**new** Book("General\_Science","Sachine Bhaske",300,***SCIENCE***,50,*sdf*.parse("2-9-2001"));

Book b4=**new** Book("Dracula","Bram Stoker",300,***NOVELS***,15,*sdf*.parse("12-2-1999"));

Book b5=**new** Book("Origins\_of\_Marvel\_Comics","Stan Lee",350,***COMICS***,30,*sdf*.parse("21-7-2004"));

list.add(b1);

list.add(b2);

list.add(b3);

list.add(b4);

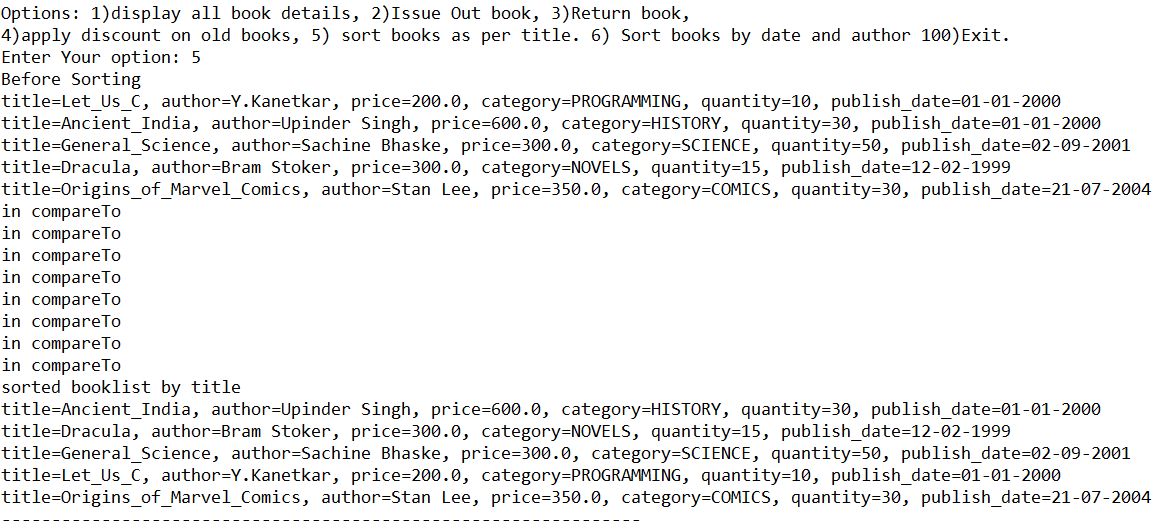
list.add(b5);

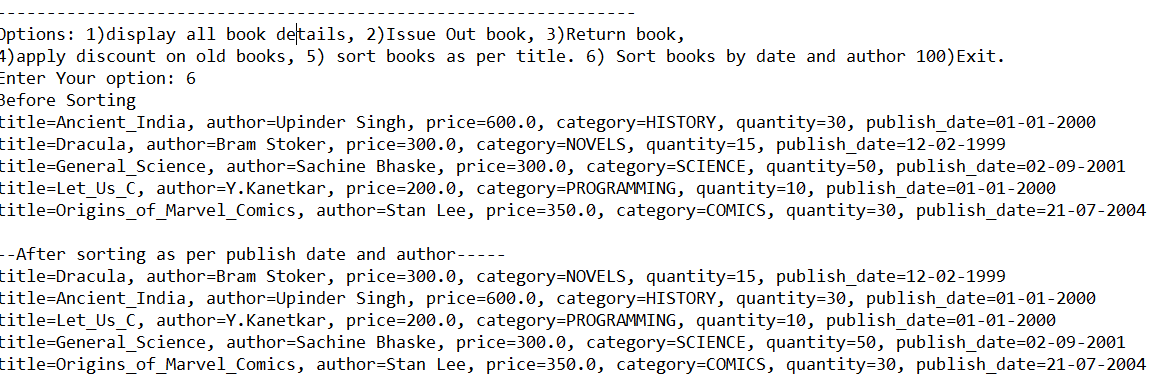
**return** list;

}

}

**Output :**

****

****

**1) BankAccount.java**

**package** com.app.core;

**import** java.time.LocalDate;

**import** custom\_exception.AccountHandlingException;

**import** **static** utils.AccountValidation.*validateBalance*;

**public** **class** BankAccount {

/\*

\* acctNo(int : PK) , customer name , type(enum) , balance , opening date (Use

\* Java 8 API : java.time.LocalDate)

\*/

**private** **int** acctNo;

**private** String customerName;

**private** AccountType type;

**private** **double** balance;

**private** LocalDate creationDate;

**public** BankAccount(**int** acctNo, String customerName, AccountType type, **double** balance, LocalDate creationDate) {

**super**();

**this**.acctNo = acctNo;

**this**.customerName = customerName;

**this**.type = type;

**this**.balance = balance;

**this**.creationDate = creationDate;

}

@Override

**public** String toString() {

**return** "BankAccount [acctNo=" + acctNo + ", customerName=" + customerName + ", type=" + type + ", balance="

+ balance + ", creationDate=" + creationDate + "]";

}

//B.L withdraw , deposit , funds transfer , apply interest (Simple int)

**public** **void** deposit(**double** amount)

{

balance += amount;

}

**public** **void** withdraw(**double** amount) **throws** AccountHandlingException

{

*validateBalance*(balance-amount);

balance -= amount;

}

**public** **void** transferFunds(BankAccount dest,**double** amount) **throws** AccountHandlingException

{

//withdraw from src a/c

**this**.withdraw(amount);

//deposit into dest a/c

dest.deposit(amount);

}

//apply interest (Simple interest)

**public** **void** applyInterest(**double** interestRate,**int** periodInYears)

{

**double** simpleInterest=(balance\*periodInYears\*interestRate)/100;

deposit(simpleInterest);

}

**public** **int** getAcctNo() {

**return** acctNo;

}

**public** String getCustomerName() {

**return** customerName;

}

**public** AccountType getType() {

**return** type;

}

**public** **double** getBalance() {

**return** balance;

}

**public** LocalDate getCreationDate() {

**return** creationDate;

}

}

**2) AccountType.java**

**package** com.app.core;

**public** **enum** AccountType {

***SAVING***, ***CURRENT***, ***FD***, ***LOAN***, ***DMAT***

}

**3) AccountHandlingException.java**

**package** custom\_exception;

@SuppressWarnings("serial")

**public** **class** AccountHandlingException **extends** Exception {

**public** AccountHandlingException(String errMesg) {

**super**(errMesg);

}

}

**4) AccountValidation.java**

**package** utils;

**import** custom\_exception.AccountHandlingException;

**public** **class** AccountValidation {

**public** **static** **final** **double** ***MIN\_BALANCE***;

**static** {

***MIN\_BALANCE*** = 1000;

}

//add static method for validating min balance condition

**public** **static** **double** validateBalance(**double** balance) **throws** AccountHandlingException

{

**if** (balance < ***MIN\_BALANCE***)

**throw** **new** AccountHandlingException("Insufficient Balance !!!!!!");

**return** balance;

}

}

**5) CollectionUtils.java**

**package** utils;

**import** java.time.LocalDate;

**import** java.util.HashMap;

**import** com.app.core.AccountType;

**import** com.app.core.BankAccount;

**import** **static** com.app.core.AccountType.\*;

**import** **static** java.time.LocalDate.*parse*;

**public** **class** CollectionUtils {

//add a static method to populate bank accts in a map

**public** **static** HashMap<Integer, BankAccount> populateMap() {

// create empty HashMap

HashMap<Integer, BankAccount> accts = **new** HashMap<>();

// public BankAccount(int acctNo, String customerName, AccountType type, double

// balance, LocalDate creationDate) {

System.***out***.println(accts.put(101, **new** BankAccount(101, "Riya", ***SAVING***, 10000, *parse*("2010-01-01"))));// null

System.***out***.println(accts.put(10, **new** BankAccount(10, "Mihir", ***SAVING***, 18000, *parse*("2010-11-01"))));// null

System.***out***.println(accts.put(55, **new** BankAccount(55, "Shubham", ***FD***, 90000, *parse*("2009-06-21"))));// null

System.***out***.println(accts.put(15, **new** BankAccount(15, "Rama", ***LOAN***, 15000, *parse*("2013-09-15"))));// null

System.***out***.println(accts.put(101, **new** BankAccount(101, "Kirti", ***SAVING***, 11000, *parse*("2012-12-01"))));// not

// null

// =>

// Riya's

// details

System.***out***.println(accts.putIfAbsent(101, **new** BankAccount(101, "Rucha", ***SAVING***, 20000, *parse*("2011-01-01"))));// not

// null

// :

// rets

// kirti's

// dtls

System.***out***.println(accts);//map will contain these details : {Mihir,Shubham,Rama,Kirti}

System.***out***.println(accts.size());//4

**return** accts;

}

}

**6) BankingApp.java**

**package** tester;

**import** **static** com.app.core.AccountType.*valueOf*;

**import** **static** java.time.LocalDate.*parse*;

**import** **static** utils.AccountValidation.*validateBalance*;

**import** **static** utils.CollectionUtils.*populateMap*;

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.HashMap;

**import** java.util.Scanner;

**import** java.util.TreeMap;

**import** com.app.core.AccountType;

**import** com.app.core.BankAccount;

**import** custom\_exception.AccountHandlingException;

**public** **class** BankingApp {

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

**try** (Scanner sc = **new** Scanner(System.***in***)) {

// get populated map of sample data

HashMap<Integer, BankAccount> acctMap = *populateMap*();

**boolean** exit = **false**;

**while** (!exit) {

System.***out***.println("1. Create A/C 2. Display All Accounts 3.Get A/C summary"

+ "4. Funds Transfer 5. Close A/C 6.Search by A/C Type 7. Sort a/cs by acct no100.Exit");

**try** {

**switch** (sc.nextInt()) {

**case** 1: // create a/c : validation rule : no dup a/cs

System.***out***.println(

"Enter a/c details acctNo, customerName, type, balance, creationDate(yyyy-MM-dd)");

**int** acctNo = sc.nextInt();

**if** (acctMap.containsKey(acctNo))

**throw** **new** AccountHandlingException("Dup a/c no!!!!!");

// new a/c no

acctMap.put(acctNo, **new** BankAccount(acctNo, sc.next(), *valueOf*(sc.next().toUpperCase()),

*validateBalance*(sc.nextDouble()), *parse*(sc.next())));

System.***out***.println("A/c created...");

**break**;

**case** 2:

System.***out***.println("All A/c Details");

// can u attach an iterator / for-each / for to the map ? NO

// convert Map ---> Collection (keySet / values /entrySet)

**for** (BankAccount a : acctMap.values())

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

**break**;

**case** 3: // get a/c summary : i/p acct no

System.***out***.println("Enter acct no to get a/c details");

BankAccount a = acctMap.get(sc.nextInt());// auto boxing (int ---> Integer)

**if** (a == **null**)

**throw** **new** AccountHandlingException("Invalid acct no!!!!!");

System.***out***.println("A/C Summary " + a);

**break**;

**case** 4: // transfer funds i/p : src acct no , dest acct no amount

System.***out***.println("Enter src a/c no , dest a/c no amount");

**int** srcNo = sc.nextInt();

**int** destNo = sc.nextInt();

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

// get src acct details from src a/c no (validate if src a/c exists)

BankAccount src = acctMap.get(srcNo);

**if** (src == **null**)

**throw** **new** AccountHandlingException("Invalid src acct no!!!!!");

BankAccount dest = acctMap.get(destNo);

**if** (dest == **null**)

**throw** **new** AccountHandlingException("Invalid dest acct no!!!!!");

// a/cs exists

src.transferFunds(dest, amount);

System.***out***.println("funds transferred...");

**break**;

**case** 5:

System.***out***.println("Enter acct no");

a = acctMap.remove(sc.nextInt());

**if** (a == **null**)

**throw** **new** AccountHandlingException("Can't close A/C : invalid acct no!!!!!!");

System.***out***.println("Closed A/C " + a);

**break**;

**case** 6:

System.***out***.println("Enter a/c type to fetch a/c balance");

AccountType type = *valueOf*(sc.next().toUpperCase());

System.***out***.println("Printing balance of all " + type + " type of a/cs");

**for** (BankAccount a1 : acctMap.values())

**if** (a1.getType().equals(type))

System.***out***.println(a1.getBalance());

**break**;

**case** 7:

// sorted map : TreeMap(Map<? extends K,? extends V> map)

TreeMap<Integer, BankAccount> sortedMap = **new** TreeMap<>(acctMap);

// JVM invokes Integer's compareTo

System.***out***.println("Accts sorted as per a/c no ");

**for** (BankAccount a1 : sortedMap.values())

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

**break**;

**case** 8: // sort accts as per descending order of acct nos n display them

// Since sorting criteria is Key based : can be done using TreeMap

// desc : custom ordering : Comparator

// TreeMap(Comparator<? super K> comp)

TreeMap<Integer, BankAccount> acctsByDescAcctNo = **new** TreeMap<>(**new** Comparator<Integer>() {

@Override

**public** **int** compare(Integer acctNo1, Integer acctNo2) {

System.***out***.println("in compare : desc acct no");

**return** acctNo2.compareTo(acctNo1);

}

});

System.***out***.println(acctsByDescAcctNo);//{} : empty treemap

acctsByDescAcctNo.putAll(acctMap);//invokes interally : ano inner cls's compare method

System.***out***.println("A/cs sorted as per desc acct nos");

**for**(BankAccount a1 : acctsByDescAcctNo.values())

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

**break**;

**case** 9 : //sort accts as per acct creation date

//Can you solve this using a TreeMap : NO (since it's creationDate : value based criteria)

//Map ---> Collection<V>

Collection<BankAccount> acctCollection=acctMap.values();

//Collections.sort(List<T> list,Comparator<? super T> comp)

//Collection<V> ---> List(AL/LL/Vector)

ArrayList<BankAccount> list=**new** ArrayList<>(acctCollection);

Collections.*sort*(list, **new** Comparator<BankAccount>() {

@Override

**public** **int** compare(BankAccount o1, BankAccount o2) {

// **TODO** Auto-generated method stub

**return** o1.getCreationDate().compareTo(o2.getCreationDate());

}

});

**for**(BankAccount a1 : list)

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

**break**;

**case** 100:

exit = **true**;

**break**;

}

} **catch** (Exception e) {

e.printStackTrace();

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

}

// clear the buffere of a scanner till new line

sc.nextLine();

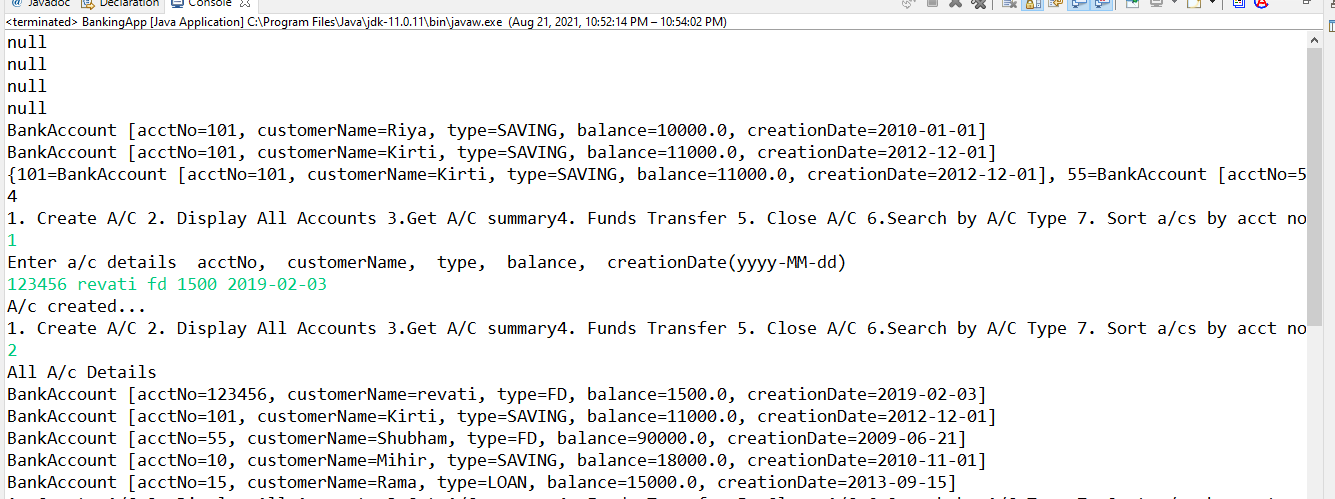
}

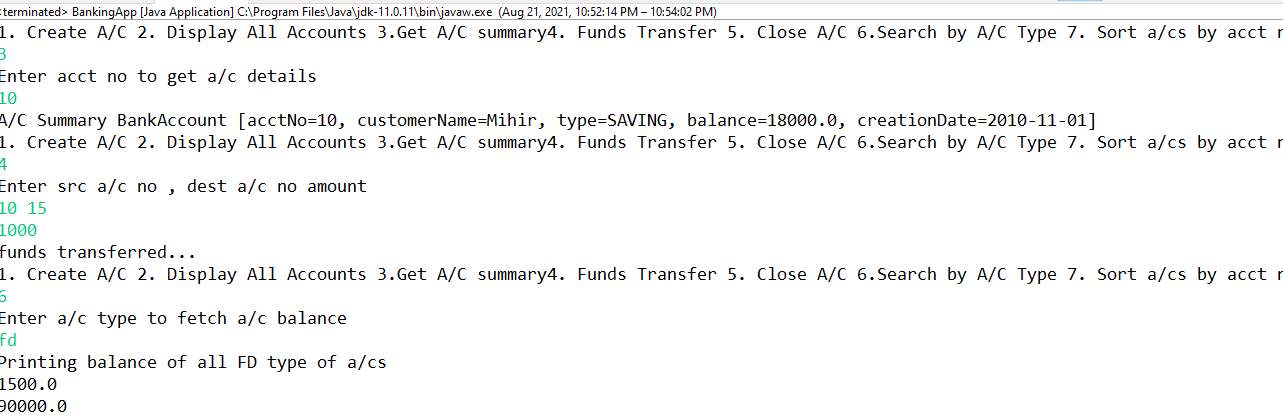
}

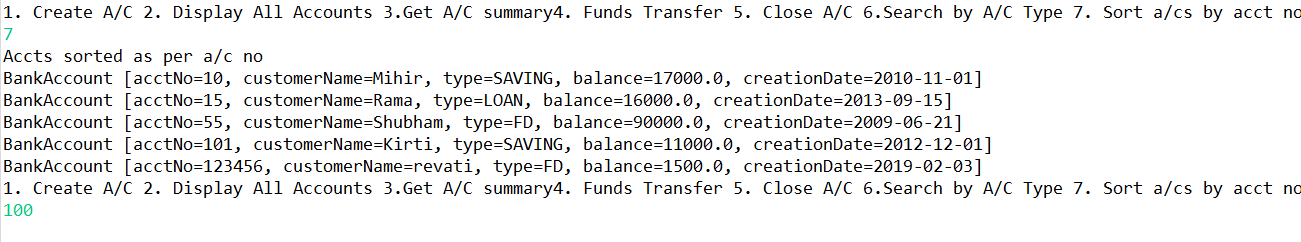
}

}

**Output:**

****

****

****

**-----------------------------------DAY 15---------------------------------------**

**No Assignments were given**

**-----------------------------------DAY 16---------------------------------------**

**1) Student.java**

**package** com.app.core;

**import** java.time.LocalDate;

/\*

\* Create Student class -- rollNo(string),name,dob(LocalDate),subject(enum),gpa(double)

Create Subject enum --JAVA,DBT,ANGULAR,REACT,SE

\*/

**public** **class** Student {

**private** String rollNo;

**private** String name;

**private** LocalDate dob;

**private** Subject subject;

**private** **double** gpa;

**public** Student(String rollNo, String name, LocalDate dob, Subject subject, **double** gpa) {

**super**();

**this**.rollNo = rollNo;

**this**.name = name;

**this**.dob = dob;

**this**.subject = subject;

**this**.gpa = gpa;

}

@Override

**public** String toString() {

**return** "Student [rollNo=" + rollNo + ", name=" + name + ", dob=" + dob + ", subject=" + subject + ", gpa=" + gpa

+ "]";

}

**public** String getRollNo() {

**return** rollNo;

}

**public** String getName() {

**return** name;

}

**public** LocalDate getDob() {

**return** dob;

}

**public** Subject getSubject() {

**return** subject;

}

**public** **double** getGpa() {

**return** gpa;

}

}

**2) Subject.java**

**package** com.app.core;

**public** **enum** Subject {

***JAVA***, ***DBT***, ***ANGULAR***, ***REACT***, ***SE***, ***DEVOPS***

}

**3) StudentCollectionUtils.java**

**package** utils;

**import** java.util.ArrayList;

**import** java.util.List;

**import** com.app.core.Student;

**import** **static** java.time.LocalDate.*parse*;

**import** **static** com.app.core.Subject.\*;

**public** **interface** StudentCollectionUtils {

//add a static method to populate student list

**static** List<Student> populateList() {

ArrayList<Student> students = **new** ArrayList<>();

students.add(**new** Student("dac-001", "Rama", *parse*("1999-01-01"), ***ANGULAR***, 8.1));

students.add(**new** Student("dac-009", "Tara", *parse*("1999-11-01"), ***JAVA***, 7.1));

students.add(**new** Student("dac-002", "Mihir", *parse*("1998-01-11"), ***REACT***, 5.8));

students.add(**new** Student("dac-005", "Rekha", *parse*("1997-11-06"), ***JAVA***, 8.8));

students.add(**new** Student("dac-003", "Anish", *parse*("1999-01-21"), ***SE***, 4.1));

students.add(**new** Student("dac-004", "Kirti", *parse*("2000-01-01"), ***DEVOPS***, 9.1));

students.add(**new** Student("dac-008", "Anshuman", *parse*("1998-06-07"), ***JAVA***, 5.2));

students.add(**new** Student("dac-006", "Timir", *parse*("1999-12-21"), ***DBT***, 6.9));

students.add(**new** Student("dac-007", "Tarun", *parse*("1997-03-21"), ***ANGULAR***, 6.6));

students.add(**new** Student("dac-010", "Priti", *parse*("1999-01-08"), ***JAVA***, 3.6));

**return** students;

}

}

**4) Test1.java**

**package** tester;

**import** com.app.core.Student;

**import** java.util.List;

**import** **static** utils.StudentCollectionUtils.*populateList*;

**public** **class** Test1 {

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

//1. Display all student details

List<Student> students=*populateList*();

// for(Student s:students)

// System.out.println(s);

// //OR

// students.forEach(s->System.out.println(s));

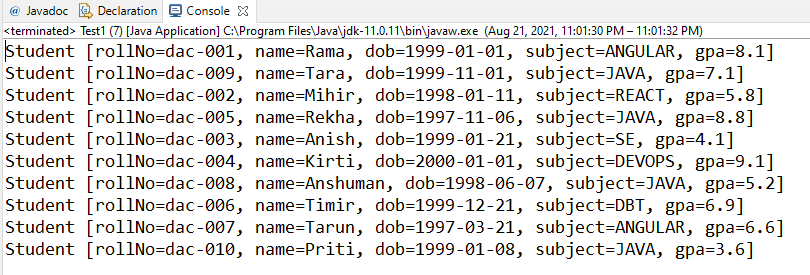
//OR

students.forEach(System.***out***::println);

}

}

**Output:**

****

**5) Test2.java**

**package** tester;

**import** **static** utils.StudentCollectionUtils.*populateList*;

**import** java.util.Comparator;

**import** java.util.Collections;

**import** java.util.List;

**import** com.app.core.Student;

**public** **class** Test2 {

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

//1.5 Display all student details sorted as per GPA

List<Student> students=*populateList*();

System.***out***.println("--------Student List before Sorting------------");

students.forEach(System.***out***::println);

/\* 1)

Collections.sort(students,new Comparator<Student>(){

@Override

public int compare(Student s1, Student s2) {

return ((Double)s1.getGpa()).compareTo(s2.getGpa());

}

});

\*/

/\*2)

Comparator<Student> comp=(s1,s2)->((Double)s1.getGpa()).compareTo(s2.getGpa());

students.sort(comp);

\*/

System.***out***.println("\n--------Student Sorted List as per GPA------------");

students.stream().

//sorted(Comparator.comparing(s->s.getGpa()));

sorted(Comparator.*comparing*(Student::getGpa)).

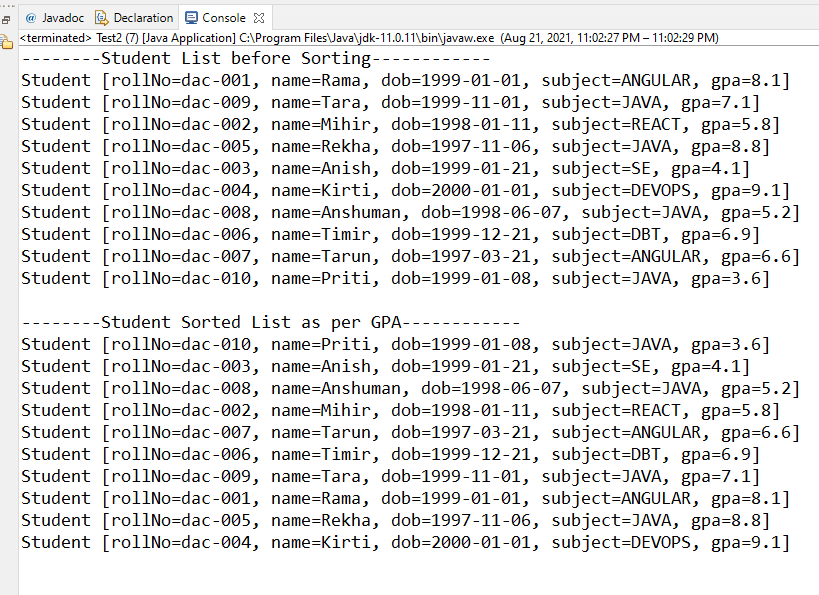
forEach(System.***out***::println);

//forEach(s->System.out.println(s));

}

}

**Output:**

****

**6) Test3.java**

**package** tester;

**import** **static** utils.StudentCollectionUtils.*populateList*;

**import** java.util.Comparator;

**import** java.util.List;

**import** com.app.core.Student;

**public** **class** Test3 {

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

// 1.5 Display all student details sorted as per descending order GPA

List<Student> students = *populateList*();

// Comparator<Student> comp = (s1, s2) -> ((Double) s2.getGpa()).compareTo(s1.getGpa());

// Collections.sort(students,comp);

// students.forEach(System.out::println);

// OR

// students.stream(). // Stream<Student> : un sorted

// sorted(comp). // Stream<Student> : sorted as per desc gpa

// forEach(System.out::println);

students.stream(). //Stream<Student> : un sorted

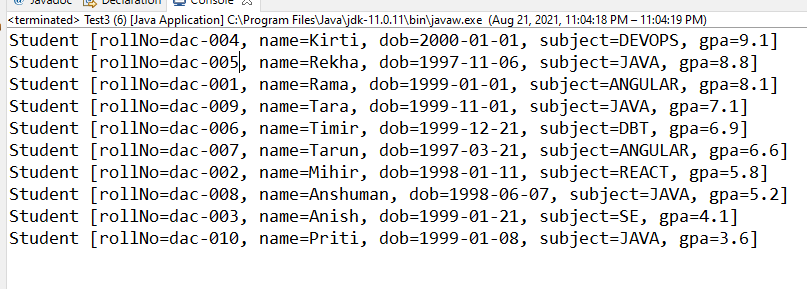
sorted(Comparator.*comparing*(Student::getGpa).reversed()). // s -> s.getGpa() Stream<Student> : sorted as per gpa

forEach(System.***out***::println);

}

}

**Output:**

****

**7) Test4.java**

**package** tester;

**import** java.util.List;

**import** java.util.Scanner;

**import** **static** utils.StudentCollectionUtils.*populateList*;

**import** com.app.core.Student;

**import** com.app.core.Subject;

**public** **class** Test4 {

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

//2. Write a tester to print sum of gpa of students for the specified subject

//(Hint : filter,map,reduce)

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

List<Student> students=*populateList*();

System.***out***.println("--------Student List ---------");

students.forEach(System.***out***::println);

System.***out***.println("\nEnter Subject: ");

Subject subject=Subject.*valueOf*(sc.next().toUpperCase());

**double** sumOfGPA=students.stream().

filter(s->s.getSubject().equals(subject)).

mapToDouble(Student::getGpa). //mapToDouble(s->s.getGpa()).

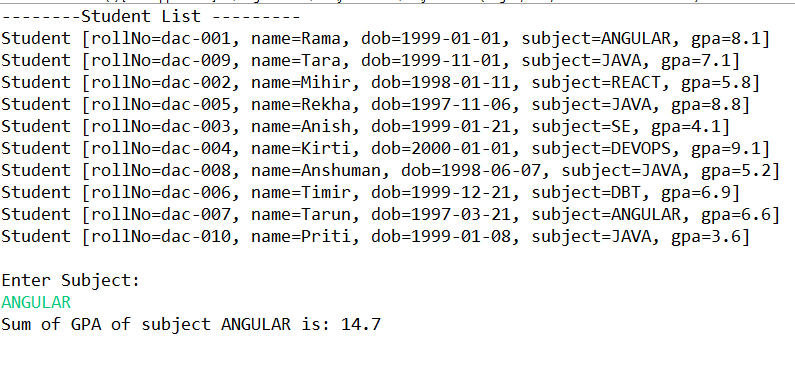
sum();

System.***out***.println("Sum of GPA of subject "+subject+" is: "+sumOfGPA);

}

}

**Output:**

****

**8) Test5.java**

**package** tester;

**import** **static** utils.StudentCollectionUtils.*populateList*;

**import** java.util.List;

**import** java.util.Scanner;

**import** com.app.core.Student;

**import** com.app.core.Subject;

**public** **class** Test5 {

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

//3. Write a tester to print average of gpa of students for the specified subject

//(Hint : filter,map,reduce)

**try**(Scanner sc=**new** Scanner(System.***in***)){

// Display all student details

List<Student> students = *populateList*();

System.***out***.println("Students : ");

students.forEach(System.***out***::println);

System.***out***.println("Enter subject: ");

Subject subject=Subject.*valueOf*(sc.next().toUpperCase());

**double** avgOfGPA=students.stream().

filter(s -> s.getSubject()==subject).

mapToDouble(Student :: getGpa). //(s-> s.getGpa())

average().orElse(-100); //return optional double

System.***out***.println("Sum of GPA of subject "+subject+" is: "+avgOfGPA);

} **catch** (Exception e) {

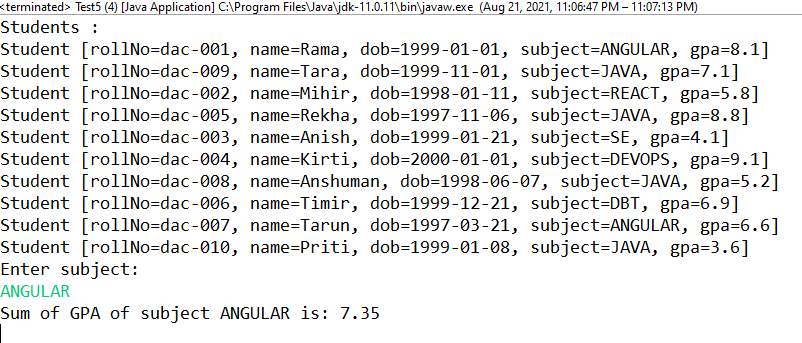
e.printStackTrace();

}

}

}

**Output:**

****

**9) Test7.java**

**package** tester;

**import** **static** utils.StudentCollectionUtils.*populateList*;

**import** java.util.Comparator;

**import** java.util.List;

**import** java.util.Optional;

**import** java.util.Scanner;

**import** com.app.core.Student;

**import** com.app.core.Subject;

**public** **class** Test7 {

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

//4 Print name of specified subject topper

//(filter reduce -- max )

**try**(Scanner sc=**new** Scanner(System.***in***))

{

// Display all student details

List<Student> students = *populateList*();

System.***out***.println("Students : ");

students.forEach(System.***out***::println);

System.***out***.println("Enter subject: ");

Subject subject=Subject.*valueOf*(sc.next().toUpperCase());

Optional<Student> opStudent=students.stream().

filter(s->s.getSubject().equals(subject)).

max(Comparator.*comparing*(Student::getGpa));

**if**(opStudent.isPresent())

System.***out***.println("Topper in "+subject+" subject is: "+opStudent.get().getName());

}**catch**(Exception e) {

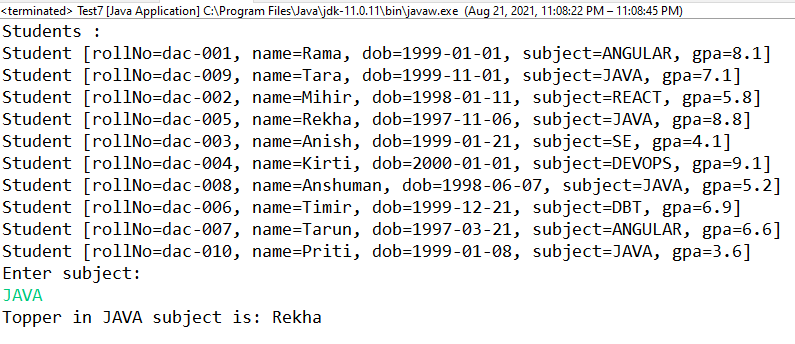
e.printStackTrace();

}

}

}

**Output:**

****

**1) Book.java**

**package** com.app.core;

**import** java.io.Serializable;

**import** java.time.LocalDate;

//Book : isbn or title(string) , author(string) , price(double), category(enum) , quantity(int) , publish date(LocalDate)

**public** **class** Book **implements** Serializable {

**private** String isbn;

**private** String author;

**private** **double** price;

**private** Category category;

**private** **int** quantity;

**private** LocalDate publishDate;

**public** Book(String isbn, String author, **double** price, Category category, **int** quantity, LocalDate publishDate) {

**super**();

**this**.isbn = isbn;

**this**.author = author;

**this**.price = price;

**this**.category = category;

**this**.quantity = quantity;

**this**.publishDate = publishDate;

}

@Override

**public** String toString() {

**return** "Book [isbn=" + isbn + ", author=" + author + ", price=" + price + ", category=" + category

+ ", quantity=" + quantity + ", publishDate=" + publishDate + "]";

}

**public** **int** getQuantity() {

**return** quantity;

}

**public** **void** setQuantity(**int** quantity) {

**this**.quantity = quantity;

}

**public** String getIsbn() {

**return** isbn;

}

**public** String getAuthor() {

**return** author;

}

**public** **double** getPrice() {

**return** price;

}

**public** Category getCategory() {

**return** category;

}

**public** LocalDate getPublishDate() {

**return** publishDate;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

}

**2) Category.java**

**package** com.app.core;

**public** **enum** Category {

***TECHNOLOGY***, ***SCIENCE***, ***SELF\_HELP***, ***YOGA***, ***MUSIC***,***FICTION***

}

**3) BookHandlingException.java**

**package** custom\_exception;

@SuppressWarnings("serial")

**public** **class** BookHandlingException **extends** Exception {

**public** BookHandlingException(String mesg) {

**super**(mesg);

}

}

**4) CollectionUtils.java**

**package** utils;

**import** **static** com.app.core.Category.***FICTION***;

**import** **static** com.app.core.Category.***SCIENCE***;

**import** **static** com.app.core.Category.***YOGA***;

**import** **static** java.time.LocalDate.*parse*;

**import** java.util.HashMap;

**import** java.util.Map;

**import** com.app.core.Book;

**public** **interface** CollectionUtils {

//add static method to populate map of books

**static** Map<String,Book> populateBooks()

{

Map<String,Book> map=**new** HashMap<>();

//String isbn, String author, double price, Category category, int quantity, LocalDate publishDate)

map.put("book1", **new** Book("book1", "Ramesh",500 ,***SCIENCE***, 50,*parse*("1999-01-01")));

map.put("book6", **new** Book("book6", "Gauri",400 ,***FICTION***, 20,*parse*("2001-01-01")));

map.put("book2", **new** Book("book2", "Ramesh",700 ,***SCIENCE***, 55,*parse*("1999-11-21")));

map.put("book3", **new** Book("book3", "Kiran",450 ,***YOGA***, 40,*parse*("1998-05-21")));

map.put("book7", **new** Book("book7", "Mihir",550 ,***SCIENCE***, 50,*parse*("1999-01-01")));

map.put("book8", **new** Book("book8", "Ramesh",510 ,***SCIENCE***, 30,*parse*("2020-01-01")));

**return** map;

}

}

**5) SerIOUtils.java**

**package** utils;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.util.Map;

**import** com.app.core.Book;

**import** **static** utils.CollectionUtils.*populateBooks*;

**public** **interface** SerIOUtils {

//store--write

//java app--->ObjectOutputStream--->FileOutputStream

**static** **void** storeBooks(Map<String,Book> books,String fileName) **throws** IOException{

**try**(ObjectOutputStream out=**new** ObjectOutputStream(**new** FileOutputStream(fileName))){

out.writeObject(books);

}

}

//restore--read

//java app<---ObjectInputStream<---FileInputStream

**static** Map<String,Book> restoreBook(String fileName){

**try**(ObjectInputStream in=**new** ObjectInputStream(**new** FileInputStream(fileName))){

**return** (Map<String,Book>)in.readObject();

}**catch**(Exception e){

**return** *populateBooks*();

}

}

}

**6) BookLibrary.java**

**package** tester;

**import** **static** utils.CollectionUtils.*populateBooks*;

**import** java.time.LocalDate;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.Map;

**import** java.util.Scanner;

**import** java.util.TreeMap;

**import** com.app.core.Book;

**import** custom\_exception.BookHandlingException;

**import** **static** java.time.LocalDate.*parse*;

**import** **static** utils.SerIOUtils.\*;

**public** **class** BookLibrary {

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

**try** (Scanner sc = **new** Scanner(System.***in***)) {

// get lib books

System.***out***.println("Enter file name to store book details");

String fileName=sc.next();

Map<String, Book> books = *restoreBook*(fileName);

**boolean** exit = **false**;

**while** (!exit) {

System.***out***.println("1. Display all books 2. Issue Out a Book 3. Return book "

+ "4. Apply discount on old books 5. Sort as per ISBN 6. Sort as per Date n Author 100.Exit");

**try** {

**switch** (sc.nextInt()) {

**case** 1: // display all books : since Map is not Iterable , Map ---> Collection view

// keySet / values/entrySet

System.***out***.println("Library books");

// for (Book b : books.values())

// System.out.println(b);

books.forEach((isbn, book) -> System.***out***.println(book));// forEach : higher order function

**break**;

**case** 2:

System.***out***.println("Enter ISBN n quantity to issue out book/s");

String isbn = sc.next();

**int** qty = sc.nextInt();

// check if book by specific isbn is available

Book book = books.get(isbn);

**if** (book == **null**)

**throw** **new** BookHandlingException("Book with ISBN " + isbn + " not found !!!!!");

// book is available , now check qty

**if** (book.getQuantity() > qty) {

book.setQuantity(book.getQuantity() - qty);

System.***out***.println("Book with ISBN " + isbn + " issued....");

} **else**

**throw** **new** BookHandlingException("Insufficient Qty for Book with ISBN " + isbn);

**break**;

**case** 3:

System.***out***.println("Enter ISBN n quantity to return a book/s");

isbn = sc.next();

qty = sc.nextInt();

book = books.get(isbn);

**if** (book == **null**)

**throw** **new** BookHandlingException("Book with ISBN " + isbn + " not found !!!!!");

book.setQuantity(book.getQuantity() + qty);

System.***out***.println("Book/s returned successfully!");

**break**;

**case** 4:

System.***out***.println("Enter date n discount amount");

LocalDate date = *parse*(sc.next());

**double** discount = sc.nextDouble();

// since searching by value based criteria : convert it to collection view n

// then filter n update

// for(Book b : books.values())

// if(b.getPublishDate().isBefore(date))

// b.setPrice(b.getPrice()-discount);

books.forEach((isbn1, bk) -> {

**if** (bk.getPublishDate().isBefore(date))

bk.setPrice(bk.getPrice() - discount);

});

System.***out***.println("Applied discount....");

**break**;

**case** 5:

// TreeMap can mamange the srting since it's based upon key based criteria

// (ISBN)

TreeMap<String, Book> sortedBooks = **new** TreeMap<>(books);

// JVM invokes String's compareTo

// display sorted books : forEach (lambda expression)

sortedBooks.forEach((isbn1, book1) -> System.***out***.println(book1));

**break**;

**case** 6:// sort as per publish date n author

// since sorting criteria is value based : can't be done with TreeMap

// convert Map ---> Collection ---> ArrayList

ArrayList<Book> list = **new** ArrayList<>(books.values());

// Collections.sort(list, new Comparator<Book>() {

//

// @Override

// public int compare(Book o1, Book o2) {

// int ret=o1.getPublishDate().compareTo(o2.getPublishDate());

// if(ret == 0) //publish date same

// return o1.getAuthor().compareTo(o2.getAuthor());

// return ret;

// }

//

// });

// function literal

Comparator<Book> bookComparator = (o1, o2) -> {

**int** ret = o1.getPublishDate().compareTo(o2.getPublishDate());

**if** (ret == 0) // publish date same

**return** o1.getAuthor().compareTo(o2.getAuthor());

**return** ret;

};

Collections.*sort*(list, bookComparator);

// sorted list

System.***out***.println("Sorted books as per date n author");

list.forEach(b -> System.***out***.println(b));

**break**;

**case** 100:

exit = **true**;

*storeBooks*(books,fileName);

System.***out***.println("Data stored successfully");

**break**;

}

} **catch** (Exception e) {

e.printStackTrace();

}

sc.nextLine();

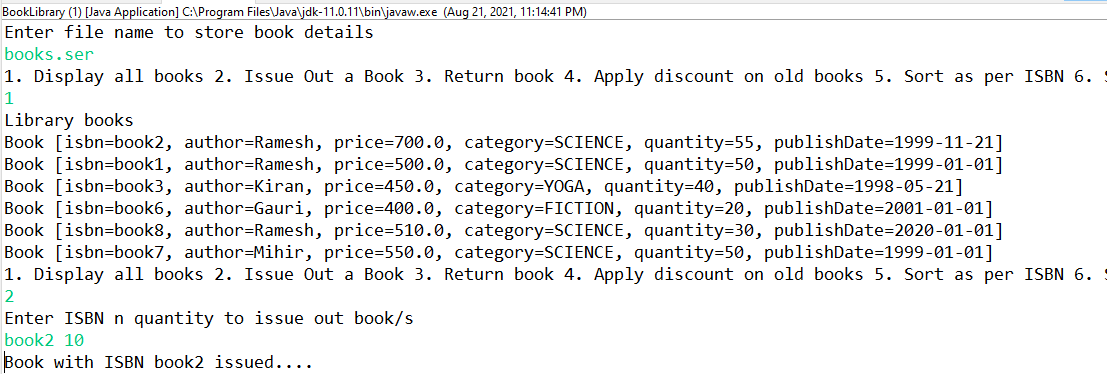
}

}

}

}

**Output:**

****

**1) EvenPrinterTask.java**

**package** runnable\_tasks;

**import** **static** java.lang.Thread.*currentThread*;

**public** **class** EvenPrinterTask **implements** Runnable {

**private** **int** start , end;

**public** EvenPrinterTask(**int** start, **int** end) {

**super**();

**this**.start = start;

**this**.end = end;

System.***out***.println("constructor of "+getClass().getName()+" invoked by "+*currentThread*().getName());

}

@Override

**public** **void** run() {

System.***out***.println(*currentThread*().getName()+" strted ");

**try** {

**for**(**int** i=start;i<=end;i++)

{

**if**(i % 2 ==0)

System.***out***.println("Even No "+i+" invoked by "+*currentThread*().getName());

// Thread.sleep(35);

}

} **catch** (Exception e) {

System.***out***.println("err in "+*currentThread*().getName()+" exc "+e);

}

System.***out***.println(*currentThread*().getName()+" over ");

}

}

**2) OddPrinterTask.java**

**package** runnable\_tasks;

**import** **static** java.lang.Thread.*currentThread*;

**public** **class** OddPrinterTask **implements** Runnable {

**private** **int** start, end;

**public** OddPrinterTask(**int** start, **int** end) {

**super**();

**this**.start = start;

**this**.end = end;

System.***out***.println("constructor of " + getClass().getName() + " invoked by " + *currentThread*().getName());

}

@Override

**public** **void** run() {

System.***out***.println(*currentThread*().getName() + " strted ");

**try** {

**for** (**int** i = start; i <= end; i++) {

**if** (i % 2 != 0)

System.***out***.println("Odd No " + i + " invoked by " + *currentThread*().getName());

// Thread.sleep(47);

}

} **catch** (Exception e) {

System.***out***.println("err in " + *currentThread*().getName() + " exc " + e);

}

System.***out***.println(*currentThread*().getName() + " over ");

}

}

**3) Tester.java**

**package** runnable\_tasks;

**import** java.util.Scanner;

**public** **class** Tester {

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

**try**(Scanner sc=**new** Scanner(System.***in***))

{

System.***out***.println("Enter begin n end values of the range");

**int** strt=sc.nextInt();

**int** end=sc.nextInt();

Thread t1=**new** Thread(**new** EvenPrinterTask(strt, end), "even");

Thread t2=**new** Thread(**new** OddPrinterTask(strt, end),"odd");

//t1 , t2 : NEW

t1.start();

t2.start(); //runnable

System.***out***.println("main waiting for child thrds to complete exec");

t1.join();

t2.join();

System.***out***.println("child thrds over... main thrd getting over!");

}**catch** (Exception e) {

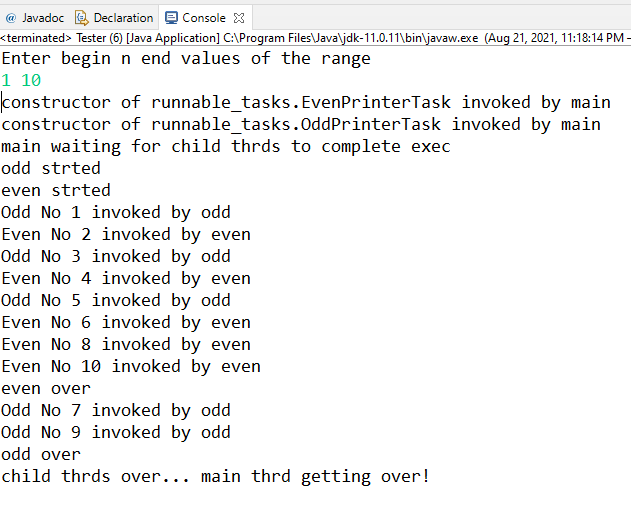
e.printStackTrace();

}

}

}

**Output:**

****