**Assignment 1**

**Q1 – Create a class Point2D , under package "com.cdac.geometry" for representing a point in x-y co-ordinate system.**

**Code -**

**package** code;

**public** **class** Point2D {

**private** **int** x;

**private** **int** y;

**public** Point2D(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

}

@Override

**public** String toString() {

**return** **this**.x + " " + **this**.y;

}

**public** **boolean** isEqual(Point2D anathorPoint) {

// boolean ans = false;

// if (this.x == AnathorPoint.x && this.y == AnathorPoint.y) {

// ans = true;

// } else

// ans = false;

// return ans;

**return** **this**.x == anathorPoint.x && **this**.y == anathorPoint.y;

}

**public** **double** calculateDistance(Point2D secondPoint) {

**return** Math.*sqrt*(Math.*pow*((secondPoint.x - **this**.x), 2) + Math.*pow*((secondPoint.y - **this**.y), 2));

}

}

Tester - **package** tester;

**import** java.util.Scanner;

**import** code.Point2D;

**public** **class** Test1 {

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

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("How Many Point to Plot");

Point2D[] arr = **new** Point2D[in.nextInt()];

**for** (**int** i = 0; i < arr.length; i++) {

System.***out***.println("Enter Co-ordinates point");

arr[i] = **new** Point2D(in.nextInt(), in.nextInt());

}

System.***out***.println("All Points : ");

**for** (Point2D i : arr) {

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

}

// for (int i = 0; i < arr.length; i++) {

// System.out.println(arr[i]);

// }

System.***out***.println("Enter Two Point to check Equality");

**int** startIndex = in.nextInt();

**int** endIndex = in.nextInt();

// validate

**if** (startIndex >= 0 && endIndex <= arr.length) {

Point2D startPoint = arr[startIndex];

Point2D endPoint = arr[endIndex];

**if** (startPoint.isEqual(endPoint)) {

System.***out***.println("Point is Equal");

} **else** {

System.***out***.println("Point is not Equal and Distance between Two Point is = ");

System.***out***.println(startPoint.calculateDistance(endPoint));

}

} **else**

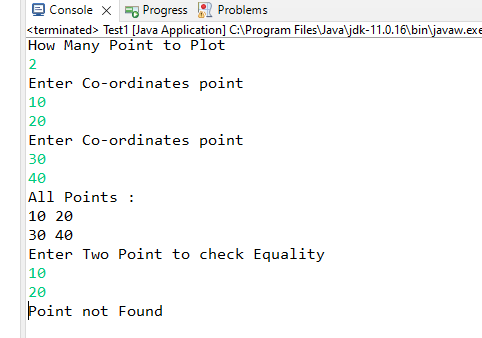
System.***out***.println("Point not Found");

in.close();

}

}

Output –



**Assignment No 2**

**Q1 - Can you arrange Fruit,Apple,Orange,Mango in inheritance hierarchy ?**

**Code**

**package** code;

**public** **class** Fruit {

**private** String name;

**private** **boolean** fresh;

**public** Fruit(String name) {

**this**.name = name;

**this**.fresh = **true**;

}

@Override

**public** String toString() {

**return** "Name=" + **this**.name;

}

**public** String taste() {

**return** "No Specific Taste";

}

**public** String getName() {

**return** "Name=" + **this**.name;

}

**public** **void** setFresh(**boolean** fresh) {

**this**.fresh = fresh;

}

**public** **void** pulp() {

};

**public** **void** jam() {

};

**public** **void** juice() {

};

}

**package** code;

**public** **class** Apple **extends** Fruit {

**private** String color;

**private** **double** weight;

**public** Apple(String color, **double** weight, String name) {

**super**(name);

**this**.color = color;

**this**.weight = weight;

}

@Override

**public** String toString() {

**return** **super**.toString() + " " + "Weigth=" + **this**.weight + " " + "color=" + **this**.color + " " + "Taste="

+ **this**.taste();

}

@Override

**public** String taste() {

**return** "Sweet ans Sour";

}

**public** **void** jam() {

System.***out***.println(**super**.toString() + " " + "Speciality : " + "Making Jam");

}

}

**package** code;

**public** **class** Mango **extends** Fruit {

**private** String color;

**private** **double** weight;

**public** Mango(String color, **double** weight, String name) {

**super**(name);

**this**.color = color;

**this**.weight = weight;

}

@Override

**public** String toString() {

**return** **super**.toString() + " " + "Weigth=" + **this**.weight + " " + "color=" + **this**.color + " " + "Taste="

+ **this**.taste();

}

@Override

**public** String taste() {

**return** "sweet";

}

**public** **void** pulp() {

System.***out***.println(**super**.toString() + " " + "Speciality : " + "Mango");

}

}

**package** code;

**public** **class** Orange **extends** Fruit {

**private** String color;

**private** **double** weight;

**public** Orange(String color, **double** weight, String name) {

**super**(name);

**this**.color = color;

**this**.weight = weight;

}

@Override

**public** String toString() {

**return** **super**.toString() + " " + "Weigth=" + **this**.weight + " " + "color=" + **this**.color + " " + "Taste="

+ **this**.taste();

}

@Override

**public** String taste() {

**return** "sour";

}

**public** **void** juice() {

System.***out***.println(**super**.toString() + " " + "Speciality : " + "Extracting juice");

}

}

Tester –

package tester;

import java.util.Scanner;

import code.Apple;

import code.Fruit;

import code.Mango;

import code.Orange;

public class FruitBasket {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

System.out.println("Enter basket size");

int size = in.nextInt();

Fruit[] basket = new Fruit[size];

int choice;

int count = 0;

if (count < basket.length) {

do {

System.out.println("Enter choice 1.Mango 2.Apple 3.Orange 4.Name of All Fruit 5.Display All ");

System.out.println("6.Update not Fresh 7.sour not Fresh 8.Fruit Speciality 9.Exit");

choice = in.nextInt();

switch (choice) {

case 1:

System.out.println("Enter details as : color , weight , name ");

basket[count] = new Mango(in.next(), in.nextDouble(), in.next());

count++;

break;

case 2:

System.out.println("Enter details as : color , weight , name ");

basket[count] = new Apple(in.next(), in.nextDouble(), in.next());

count++;

break;

case 3:

System.out.println("Enter details as : color , weight , name ");

basket[count] = new Orange(in.next(), in.nextDouble(), in.next());

count++;

break;

case 4:

for (Fruit f : basket) {

if (f != null) {

if (f instanceof Mango) {

System.out.println(((Mango) f).getName());

} else if (f instanceof Apple) {

System.out.println(((Apple) f).getName());

} else

System.out.println(((Orange) f).getName());

}

}

break;

case 5:

for (Fruit f : basket) {

if (f != null)

System.out.println(f);

}

break;

case 6:

System.out.println("Enter Index of Fruit which turn Stale ");

int index = in.nextInt();

if (index >= 0 && index < count) {

basket[index].setFresh(false);

} else

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

break;

case 7:

for (Fruit f : basket) {

if (f != null && f.taste().equals("sour")) {

f.setFresh(false);

} else

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

}

break;

case 8:

System.out.println("Enter Index to Get Speciality of Fruit : ");

int ind = in.nextInt();

for (Fruit f : basket) {

if (f != null) {

if (f instanceof Mango) {

basket[ind].pulp();

} else if (f instanceof Apple) {

basket[ind].jam();

} else

basket[ind].juice();

}

}

break;

}

} while (choice != 9);

} else

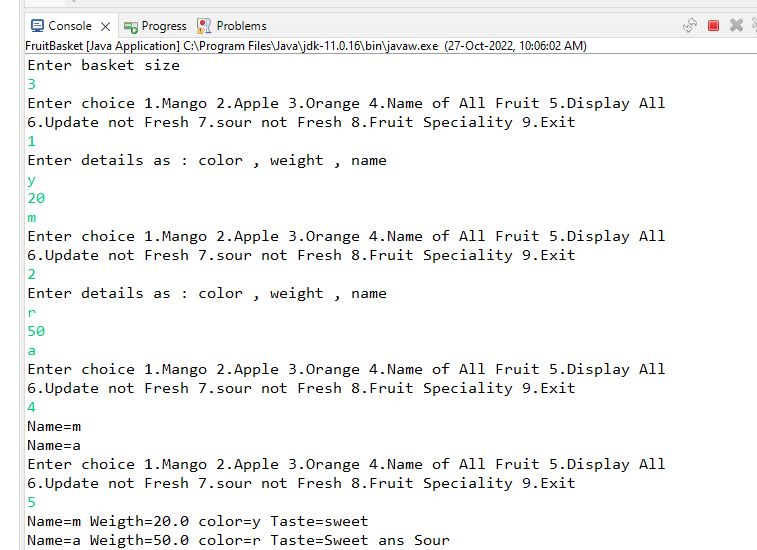
System.out.println("basket is full");

in.close();

}

}

Ans –



**Assignment No3 –**

**Q1 - Create Emp based organization structure --- Emp , Mgr , Worker All of above classes must be in --com.app.org**

**Code**

**package** code;

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

**private** **int** id;

**private** String firstName;

**private** String lastName;

**private** **int** deptID;

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

**private** **static** **int** *counter*;

**public** Employee(String fn, String ln, **int** deptid, **double** basicsal) {

**this**.firstName = fn;

**this**.lastName = ln;

**this**.deptID = deptid;

**this**.basicSalary = basicsal;

**this**.id = *counter*;

*counter*++;

}

**public** Employee(String fname, String lName) {

**this**.firstName = fname;

**this**.lastName = lName;

}

**static** {

*counter* = 1;

}

@Override

**public** String toString() {

**return** "Employee No :" + " " + **this**.id + " " + "Full Name :" + **this**.firstName + **this**.lastName + " " + "DeptID :"

+ **this**.deptID;

}

**public** **double** computNetSalary() {

**return** **this**.basicSalary;

}

**public** **void** setSal(**double** sal) {

**this**.basicSalary = basicSalary + sal;

}

**public** **int** getId() {

**return** id;

}

**public** **static** **int** getCounter() {

**return** *counter*;

}

@Override

**public** **boolean** equals(Object anathorEmp) {

**if** (anathorEmp **instanceof** Employee) {

**return** **this**.firstName.equals(((Employee) anathorEmp).firstName)

&& **this**.lastName.equals(((Employee) anathorEmp).lastName);

}

**return** **false**;

}

}

**package** code;

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

**private** **double** perfBonus;

**public** Manager(String fn, String ln, **int** deptid, **double** basicsal, **double** perfBonus) {

**super**(fn, ln, deptid, basicsal);

**this**.perfBonus = perfBonus;

}

@Override

**public** String toString() {

**return** **super**.toString();

}

@Override

**public** **double** computNetSalary() {

**return** **super**.computNetSalary() + perfBonus;

}

**public** **double** getPerBonus() {

**return** perfBonus;

}

}

**package** code;

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

**private** **int** hrWorked;

**private** **int** hrRate;

**public** Worker(String fn, String ln, **int** deptid, **double** basicsal, **int** hrwork, **int** hrrate) {

**super**(fn, ln, deptid, basicsal);

**this**.hrRate = hrrate;

**this**.hrWorked = hrwork;

}

@Override

**public** String toString() {

**return** **super**.toString();

}

@Override

**public** **double** computNetSalary() {

**return** **super**.computNetSalary() + (**this**.hrRate \* **this**.hrWorked);

}

**public** **int** getHrRate() {

**return** **this**.hrRate;

}

}

**package** code.custome\_exception;

**public** **class** IdNotFoundException **extends** Exception {

**public** IdNotFoundException(String mesg) {

**super**(mesg);// to invoke immediate super cls(Exception)

}

}

**package** code.custome\_exception;

**import** code.Employee;

**public** **class** IdValidation {

**public** **void** IdValid(**int** id) **throws** IdNotFoundException {

}

**public** **static** **void** validateName(String fname, String lname, Employee[] arr) **throws** IdNotFoundException {

Employee emp = **new** Employee(fname, lname);

**for** (Employee E : arr) {

**if** (E != **null** && E.equals(emp)) {

**throw** **new** IdNotFoundException("Emp Already Exist");

} **else**

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

}

}

}

**Tester -**

package tester;

import java.util.Scanner;

import code.Employee;

import code.Manager;

import code.Worker;

import code.custome\_exception.IdNotFoundException;

public class TestOrganization {

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

Scanner in = new Scanner(System.in);

System.out.println("Enter the Number of Employee");

Employee[] emp = new Employee[in.nextInt()];

int count = 0;

int choice;

do {

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

System.out.println("1.Hire Manager 2.Hire Worker 3.All Emp Info 4.Update base Sal 5.Exit");

choice = in.nextInt();

switch (choice) {

case 1:

System.out.println("Enter Details as First Name , Last Name , DeptId, Base sal , Perf Bonus");

emp[count] = new Manager(in.next(), in.next(), in.nextInt(), in.nextDouble(), in.nextDouble());

count++;

break;

case 2:

System.out.println("Enter Details as First Name , Last Name , DeptId, Base sal , Hr Rate , Hr Work");

emp[count] = new Worker(in.next(), in.next(), in.nextInt(), in.nextDouble(), in.nextInt(),

in.nextInt());

count++;

break;

case 3:

for (Employee e : emp) {

if (e != null)

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

}

break;

case 4:

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

int id = in.nextInt();

if (id >= 1 && id <= Employee.getCounter()) {

for (int i = 0; i < count; i++) {

if (emp[i].getId() == id) {

System.out.println("Enter Increase Salary");

emp[i].setSal(in.nextDouble());

}

}

} else

throw new IdNotFoundException(" ID Not Found");

break;

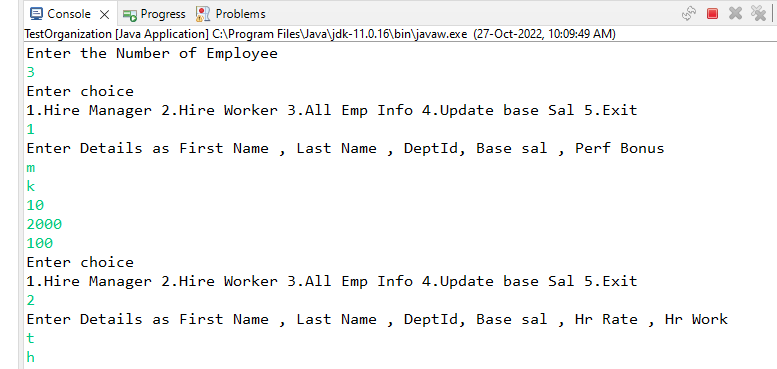
}

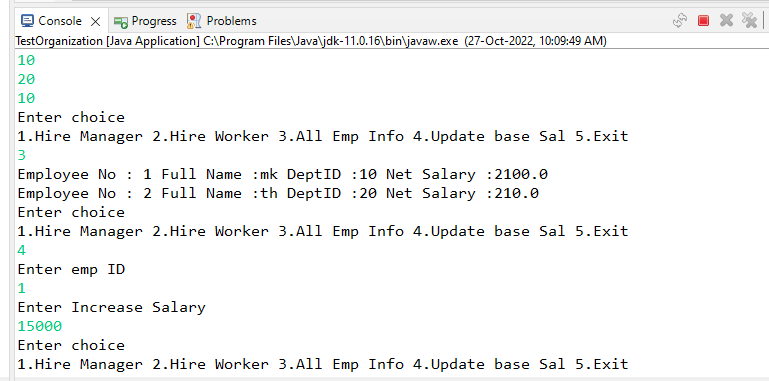
} while (choice != 5);

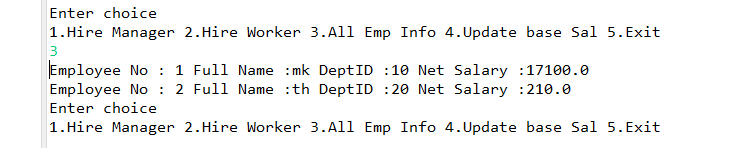
in.close();

}

}

Ans - 





**ASSIGNMENT 4**

package Showroom;

import java.util.Date;

import java.text.SimpleDateFormat;

public class Vehicle {

private String chasino;

private String name;

private Color color;

private double price;

private Date mfctdate;

private String company;

private static int chascnt;

private static String chasint;

public static SimpleDateFormat sdf;

static

{

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

chascnt=153;

chasint="CPD-";

}

public Vehicle(String name,Color color,double price,Date mfctdate,String company)

{

super();

this.chasino=chasint+chascnt;

this.name=name;

this.color=color;

this.price=price;

this.mfctdate=mfctdate;

this.company=company;

chascnt++;

}

public Vehicle(String chasino)

{

this.chasino=chasino;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Color getColor() {

return color;

}

public void setColor(Color color) {

this.color = color;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public Date getMfctdate() {

return mfctdate;

}

public void setMfctdate(Date mfctdate) {

this.mfctdate = mfctdate;

}

public String getCompany() {

return company;

}

public void setCompany(String company) {

this.company = company;

}

@Override

public String toString()

{

return "Vehicle:- [(\nName:- "+name+"\nChassis No.:- "+chasino+"\nColor:- "+color+"\nPrice:- "+price+"\nManufacturing Date:- "+sdf.format(mfctdate)+"\nCompany:- "+company+" ]";

}

@Override

public boolean equals(Object anvehicle)

{

if(anvehicle instanceof Vehicle)

{

return this.chasino.equals(((Vehicle)anvehicle).chasino);

}

return false;

}

}

**package** Showroom;

**public** **enum** Color {

***WHITE***,***BLACK***,***SILVER***,***RED***,***BROWN***,***BLUE***

}

**package** Showroom;

@SuppressWarnings("serial")

**public** **class** ShowroomException **extends** Exception{

**public** ShowroomException(String mesg)

{

**super**(mesg);

}

}

package Showroom;

import java.text.ParseException;

import java.util.Arrays;

import java.util.Date;

import Showroom.Color;

import Showroom.Vehicle;

import Showroom.ShowroomException;

import static Showroom.Vehicle.sdf;

public class ValidationSr {

public static final double Min\_price;

public static final double Max\_price;

public static Date begindate;

public static Date enddate;

static

{

Min\_price=20000;

Max\_price=200000;

try

{

begindate = sdf.parse("01-08-2022");

enddate = sdf.parse("09-02-2025");

} catch (ParseException e)

{

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

}

}

public static double validateprice(double price) throws ShowroomException

{

if(price < Min\_price || price > Max\_price)

throw new ShowroomException("Price is out of the range....!!!!!!!");

return price;

}

public static Color validatecolor(String clr) throws ShowroomException

{

try

{

return Color.valueOf(clr.toUpperCase());

} catch (IllegalArgumentException e)

{

StringBuilder sb = new StringBuilder("Available Colors \n");

sb.append(Arrays.toString(Color.values())+"\n");

sb.append("Choose correct colour.....");

throw new ShowroomException(sb.toString());

}

}

public static Date parseAndvalidatedate(String mdate) throws ParseException,ShowroomException

{

Date mfcdate=sdf.parse(mdate);

if(mfcdate.before(begindate) || mfcdate.after(enddate))

throw new ShowroomException("Invalid Date!!!!!!!!");

return mfcdate;

}

public static Vehicle validatechasis(String chasino,Vehicle[] varr) throws ShowroomException

{

Vehicle newvehicle = new Vehicle(chasino);

for(Vehicle v : varr)

if(v != null)

if(v.equals(newvehicle))

return newvehicle;

throw new ShowroomException("Duplicate ChasisNo. is Detected!!!!");

}

}

package tester;

import java.util.Date;

import java.util.Scanner;

import Showroom.Color;

import Showroom.Vehicle;

import Showroom.ShowroomException;

import static Showroom.ValidationSr.\*;

import static Showroom.Vehicle.sdf;

public class TestShowroom {

public static void main(String[] args) {

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

{

System.out.println("Enter Capacity os Showroom");

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

int count=0;

boolean flag=false;

while(!flag)

{

System.out.println("\n1.Add a vehicle \n2.Display All Vehicles \n3.DisplayAll Vehicles by Color \n4.Apply Discount to a Vehicle \n6.Exit");

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

try {

switch(sc.nextInt())

{

case 1:

if(count<vehicles.length)

{

System.out.println("Enter 1.Name 2.Color 3.Price 4.Manufacturing Date 5.Company");

Vehicle v=new Vehicle(sc.next(),validatecolor(sc.next()),validateprice(sc.nextDouble()),parseAndvalidatedate(sc.next()),sc.next());

vehicles[count++]=v;

System.out.println("Vehicle is added to the Showroom");

}

else

throw new ShowroomException("Showroom capacity is full");

break;

case 2:

System.out.println("Show Details");

for(Vehicle v1: vehicles)

if(v1 != null)

System.out.println(v1);

break;

case 3:

for(Color c:Color.values())

System.out.println(c);

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

Color cr=validatecolor(sc.next().toUpperCase());

for(Vehicle v:vehicles) {

if(v!=null && v.getColor().equals(cr))

System.out.println(v);

}

break;

case 4:

System.out.println("Enter Vehicle Chasis No.");

Vehicle vr=validatechasis(sc.next(), vehicles);

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

vr.setPrice(sc.nextDouble());

break;

case 6:

flag=true;

break;

}

}catch(Exception e)

{

//e.printStackTrace();

sc.nextLine();

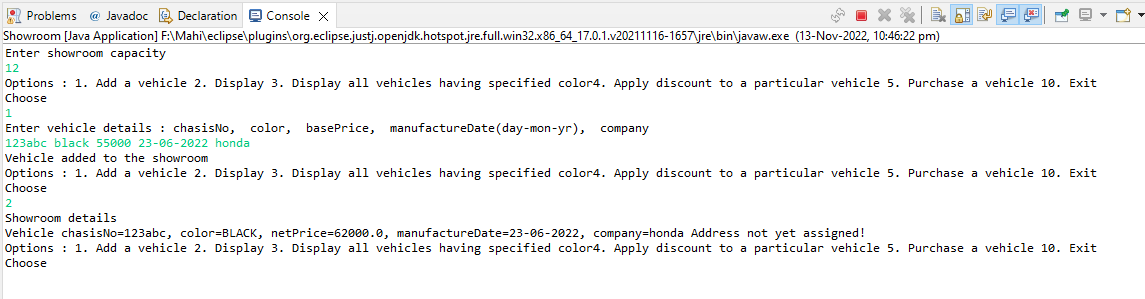
}

}

}

}

}

Ans

**ASSIGNMENT 5 - CMS**

Customer Class : -

**package** code;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

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

**private** **int** id;

**private** String name;

**private** String email;

**private** String password;

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

**private** Date dob;

**private** Type type;

**private** **static** **int** *counter*;

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

**public** Customer(String name, String email, String password, **double** registrationAmount, Date dob, Type type) {

**this**.id = *counter*;

**this**.name = name;

**this**.email = email;

**this**.password = password;

**this**.registrationAmount = registrationAmount;

**this**.dob = dob;

**this**.type = type;

*counter*++;

}

**public** Customer(String email) {

**this**.email = email;

}

**static** {

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

*counter* = 1;

}

@Override

**public** **boolean** equals(Object anathorCust) {

**if** (anathorCust **instanceof** Customer) {

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

}

**return** **false**;

}

@Override

**public** String toString() {

**return** "Customer [id=" + id + ", name=" + name + ", email=" + email + ", password=" + password

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

}

// Natural Ordering

@Override

**public** **int** compareTo(Customer o) {

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

**return** **this**.email.compareTo(o.email);

}

**public** String getEmail() {

**return** email;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getPassword() {

**return** password;

}

**public** Type getType() {

**return** type;

}

**public** String getName() {

**return** name;

}

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

**return** *sdf*;

}

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

Customer.*sdf* = sdf;

}

**public** Date getDob() {

**return** dob;

}

}

Demo Class :-

**package** code;

**import** **static** code.ValidationRules.*parsAndDatevalidation*;

**import** java.text.ParseException;

**import** java.util.ArrayList;

**public** **class** CustomerDemoDetails {

**public** **static** ArrayList<Customer> customerlist() **throws** CustomerHandlingException, ParseException {

ArrayList<Customer> customers = **new** ArrayList<Customer>(100);

customers.add(**new** Customer("Mahendra", "mkolhe23@gmail.com", "Mahi@23", 2500,

*parsAndDatevalidation*("01-05-1994"), Type.***GOLD***));

customers.add(**new** Customer("Mahi", "mahikolhe23@gmail.com", "Mahi@123", 2700,

*parsAndDatevalidation*("23-07-1992"), Type.***DIAMOND***));

customers.add(**new** Customer("MK", "mahi23@gmail.com", "Mahi@1234", 3000, *parsAndDatevalidation*("11-09-1990"),

Type.***PLATINUM***));

**return** customers;

}

}

Custom Exception : -

**package** code;

@SuppressWarnings("serial")

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

**public** CustomerHandlingException(String msg) {

**super**(msg);

}

}

Enum : -

**package** code;

**public** **enum** Type {

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

}

Validation Rules :-

**package** code;

**import** **static** code.Customer.*sdf*;

**import** java.text.ParseException;

**import** java.util.Arrays;

**import** java.util.Date;

**import** java.util.List;

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

// Email Validation Method

**public** **static** String emailValidation(String email) **throws** CustomerHandlingException {

String pattern = "^[\\w-\\.]+@([\\w-]+\\.)+(com|org)$";

// Customer newM = new Customer(email);

**if** (email.matches(pattern)) {

**return** email;

} **else** {

**throw** **new** CustomerHandlingException("Enter valid Email");

}

}

**public** **static** String emailValidate(String email, List<Customer> listN) **throws** CustomerHandlingException {

Customer newM = **new** Customer(email);

*emailValidation*(email);

**if** (listN.contains(newM)) {

**throw** **new** CustomerHandlingException("Customer Already Exist!!!!!!!");

}

**return** email;

}

// Password Validation method

**public** **static** String passwordValidation(String password) **throws** CustomerHandlingException {

String regex = "(?=.\*\\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[#@$\*%]).{5,20}";

**if** (!password.matches(regex)) {

**throw** **new** CustomerHandlingException("Enter valid Password");

}

**return** password;

}

// Type validation method

**public** **static** Type typeValidation(String type) **throws** CustomerHandlingException {

**try** {

**return** Type.*valueOf*(type.toUpperCase());

} **catch** (IllegalArgumentException e) {

StringBuilder sb = **new** StringBuilder("Select Valid Types");

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

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

}

}

// Dob validation method

**public** **static** Date parsAndDatevalidation(String date) **throws** ParseException, CustomerHandlingException {

Date dob = *sdf*.parse(date);

**if** (dob.after(*sdf*.parse("01-01-1995"))) {

**throw** **new** CustomerHandlingException("Dob Must before 01-01-1995");

}

**return** dob;

}

}

Tester –

**package** tester;

**import** **static** code.Customer.*sdf*;

**import** **static** code.CustomerDemoDetails.*customerlist*;

**import** **static** code.ValidationRules.\*;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.util.Iterator;

**import** java.util.Scanner;

**import** code.Customer;

**import** code.CustomerHandlingException;

**import** code.Type;

**public** **class** Registration {

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

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

ArrayList<Customer> customerList = *customerlist*();

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

**while** (!exit) {

System.***out***.println("1.Customer Registration 2.Display All customer Details 3.Customer Login "

+ "4.Change Passworrd 5.Un-Subscribe Customer 6.Name of customer borned between specified dates "

+ "7.Specific plan customer details 8.Exit");

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

**try** {

**switch** (in.nextInt()) {

**case** 1:

System.***out***.println("Enter Customer Details: Name,Email,Password,Registraion Amount,"

+ "Date of Birth,Choose Service Plan From 1.Silver,2.Gold 3.Platinum 4.Diamond ");

// Checking all customer details and register if valid

Customer c1 = **new** Customer(in.next(), *emailValidate*(in.next(), customerList),

*passwordValidation*(in.next()), in.nextDouble(), *parsAndDatevalidation*(in.next()),

*typeValidation*(in.next()));

customerList.add(c1);

System.***out***.println("Customer Registration successfully");

**break**;

**case** 2:

System.***out***.println("All Customer Details");

// Using Iterator Display customer details

Iterator<Customer> itr = customerList.iterator();

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

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

}

**break**;

**case** 3:

System.***out***.println("Enter Email");

String find = in.next();

**for** (Customer i : customerList) {

**if** (i.getEmail().equals(find)) {

System.***out***.println("Enter Password");

**if** (i.getPassword().equals(in.next())) {

System.***out***.println("Login Successfully");

**break**;

} **else**

**throw** **new** CustomerHandlingException("Invalid Password");

} **else**

**throw** **new** CustomerHandlingException("Invalid Email");

}

**break**;

**case** 4:

System.***out***.println("Enter Email to change Password");

find = in.next();

**for** (Customer i : customerList) {

**if** (i.getEmail().equals(find)) {

System.***out***.println("Enter old Password");

**if** (i.getPassword().equals(in.next())) {

System.***out***.println("Enter new Password");

i.setPassword(in.next());

System.***out***.println("Password Change Successfully");

**break**;

} **else**

**throw** **new** CustomerHandlingException("Invalid Old Password");

} **else**

**throw** **new** CustomerHandlingException("Invalid Email");

}

**break**;

**case** 5:

System.***out***.println("Enter User Id ");

**int** id = in.nextInt();

Iterator<Customer> itrr = customerList.iterator();

**for** (Customer i : customerList) {

**if** (i.getId() == id) {

itrr.next();

itrr.remove();

} **else**

**throw** **new** CustomerHandlingException("User Not Found");

}

**break**;

**case** 6:

System.***out***.println("Enter Dob of customer");

Date beginDate = *sdf*.parse(in.next());

Date endDate = *sdf*.parse(in.next());

**for** (Customer i : customerList) {

**if** (i.getDob().after(beginDate) && i.getDob().before(endDate)) {

i.getName();

}

}

**break**;

**case** 7:

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

Type t = Type.*valueOf*(in.next().toUpperCase());

**for** (Customer i : customerList) {

**if** (i.getType() == t) {

i.getName();

}

}

**break**;

}

} **catch** (Exception e) {

e.printStackTrace();

in.nextLine();

}

}

} **catch** (Exception e) {

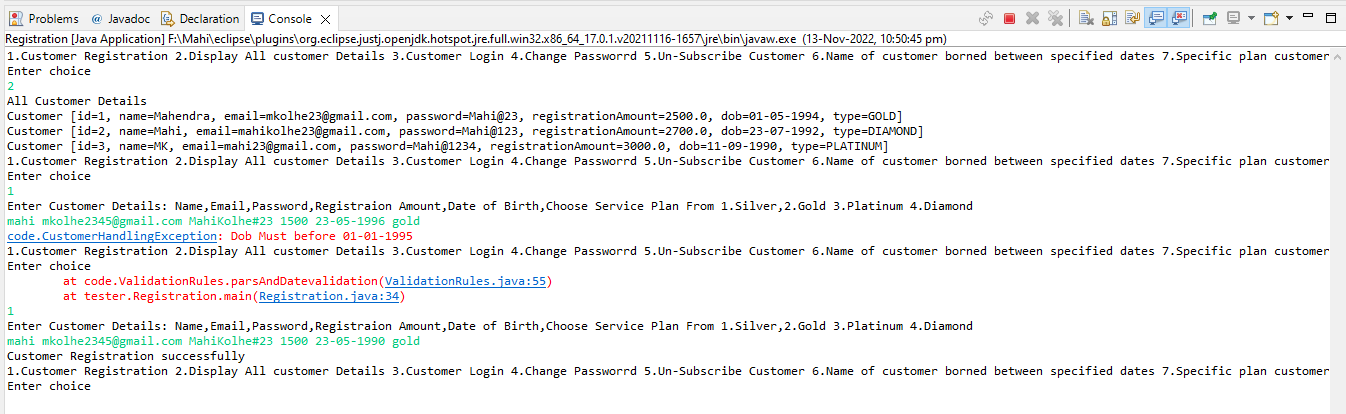
e.printStackTrace();

}

}

}

Ans –



**ASSIGNMENT 6 - ArrayList**

4.1 Create empty arrylist to store strings

4.2. Add some strings : "one","two","three","four" ....."ten","three","five"

Confirm if dups are allowed or not

Is it a ordered collection ?

Is it a sorted collection ?

Display list contents , using for-each

4.3 Accept a string from user(using scanner)

If it exists in the list , display the index of it's 1st occurrence.

Otherwise print a mesg : doesn't exist

4.4 Upper case all the strings from the array list , which starts with char 't'

Display the list contents again

4.5 Accept index from user

Remove the string at a specified index.

4.6 Create another arraylist , having the same contents of the earlier list

Display it's contents

4.7 Sort 1st arraylist.

Display contents of both of the arraylist

4.8 Remove all the strings from the 2nd list , containing a char 'e'

Display it !

**package** code;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Collections;

**import** java.util.Iterator;

**import** java.util.Scanner;

**public** **class** StringArrayList {

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

ArrayList<String> stringList = **new** ArrayList<>();

stringList.addAll(Arrays.*asList*("one", "two", "three", "four", "five", "six", "one"));

// 4.2 Display list

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

// 4.3 Exist or not

System.***out***.println("Enter string to checking is exist or not ");

Scanner in = **new** Scanner(System.***in***);

String checker = in.next();

System.***out***

.println(stringList.contains(checker) ? "Exist at index " + stringList.indexOf(checker) : "Not Exist");

// 4.4 Upper case all

System.***out***.println("To Print string in uppercase start with t");

**for** (**int** i = 0; i < stringList.size(); i++) {

**if** (stringList.get(i).startsWith("t")) {

stringList.set(i, stringList.get(i).toUpperCase());

}

}

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

// 4.5 remove string

System.***out***.println("Enter the index to remove element ");

**int** index = in.nextInt();

stringList.remove(index);

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

// 4.6 new arraylist

System.***out***.println("Adding element in new arralist");

ArrayList<String> st2 = **new** ArrayList<>();

**for** (String i : stringList) {

st2.add(i);

}

System.***out***.println(st2);

// 4.7 sort first AL

System.***out***.println("Sorting AL");

Collections.*sort*(stringList);

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

// 4.8 remove string contain e

System.***out***.println("Remove String that contains e");

Iterator<String> itr = st2.iterator();

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

**if** (itr.next().contains("e")) {

itr.remove();

}

}

System.***out***.println("Print using iterator");

System.***out***.println(st2);

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

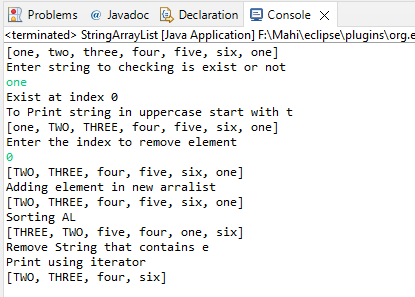
System.***out***.print(itr.next() + " ");

in.close();

}

}

}

Ans –

**ASSIGNMENT 7 - LMS**

**Books Class -**

**package** code;

**import** java.io.Serializable;

**import** java.time.LocalDate;

**public** **class** Books **implements** Serializable {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String bookTitle;

**private** Category category;

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

**private** LocalDate publishDate;

**private** String authorName;

**private** **int** qty;

**public** Books(String bookTitle, Category category, **double** price, LocalDate publishDate, String authorName, **int** qty) {

**this**.bookTitle = bookTitle;

**this**.category = category;

**this**.price = price;

**this**.publishDate = publishDate;

**this**.authorName = authorName;

**this**.qty = qty;

}

@Override

**public** String toString() {

**return** "Book Details [bookTitle=" + bookTitle + ", category=" + category + ", price=" + price + ", publishDate="

+ publishDate + ", authorName=" + authorName + ", qty=" + qty + "]";

}

}

Enum Category –

**package** code;

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

***NOVEL***, ***BIOGRAPHY***, ***TECHNICAL***, ***BUSINESS***, ***FINANCE***

}

Custom Exception

**package** exception;

//@SuppressWarnings("serial")

**public** **class** CustomHandlingException **extends** Exception {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

// Exceptin Ctor taking string msg and pass to super class

**public** CustomHandlingException(String msg) {

**super**(msg);

}

}

IO Utils –

**package** utils;

**import** **static** utils.SampleBooksDetails.*populateBooks*;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.util.Map;

**import** code.Books;

**import** exception.CustomHandlingException;

**public** **class** IOUtils {

// add static method to store books details

**public** **static** **void** storeBookDetails(String fileName, Map<String, Books> books)

**throws** FileNotFoundException, IOException {

// create ser chain: java app---->OOS--->FOS(bin file name)

**try** (ObjectOutputStream out = **new** ObjectOutputStream(**new** FileOutputStream(fileName))) {

out.writeObject(books);

System.***out***.println("Stored book details successfully");

}

}

// add static method to return populated map having sample data n in case file

// dosent exist otherwise in case

// return the map restored from bin file

@SuppressWarnings("unchecked")

**public** **static** Map<String, Books> RestoreBooks(String fileName)

**throws** ClassNotFoundException, IOException, CustomHandlingException {

File f = **new** File(fileName);

// Validate file for read

**if** (f.isFile() && f.canRead()) {

// attached data strms: java app<----OIS<----FIS(bin file Name)

**try** (ObjectInputStream in = **new** ObjectInputStream(**new** FileInputStream(fileName))) {

**return** (Map<String, Books>) in.readObject();

}

}

// no valid file

System.***out***.println("Invalid File");

**return** *populateBooks*();// rets map with sample data

}

}

Demo Books –

**package** utils;

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

**import** java.util.HashMap;

**import** java.util.Map;

**import** code.Books;

**import** **static** code.Category.\*;

**import** exception.CustomHandlingException;

**public** **class** SampleBooksDetails {

// Adding static method of Books Sample For testing

**public** **static** Map<String, Books> populateBooks() **throws** CustomHandlingException {

Map<String, Books> books = **new** HashMap<>();

books.put("abc1", **new** Books("abc1", ***BIOGRAPHY***, 1000, *parse*("2022-05-06"), "MKG1", 10));

books.put("abc2", **new** Books("abc2", ***TECHNICAL***, 1100, *parse*("2021-01-09"), "MKG2", 20));

books.put("abc3", **new** Books("abc3", ***BUSINESS***, 1200, *parse*("2020-12-06"), "MKG3", 35));

books.put("abc4", **new** Books("abc4", ***FINANCE***, 1300, *parse*("2019-07-06"), "MKG4", 16));

books.put("abcd5", **new** Books("abcd5", ***NOVEL***, 1400, *parse*("2018-04-22"), "MKG5", 30));

**return** books;

}

}

Tester –

**package** tester;

**import** **static** utils.IOUtils.*RestoreBooks*;

**import** java.time.LocalDate;

**import** java.util.Map;

**import** java.util.Scanner;

**import** code.Books;

**import** code.Category;

**import** exception.CustomHandlingException;

**public** **class** LibraryManagement {

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

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

// DS for Storing Books with key is String and value is books details

System.***out***.println("Enter file Name to read data");

String fileName = in.nextLine();

// restore data

Map<String, Books> books = *RestoreBooks*(fileName);

// display map

// for (Books b : books.values()) {

// System.out.println(b);

// }

// // store books

// storeBookDetails(fileName, books);

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

// while loop for menu

**while** (!exit) {

System.***out***.println("Enter choice 1.Adding Book 2.Display Books 3.Remove Books");

**try** {

**switch** (in.nextInt()) {

**case** 1:

System.***out***.println("Enter Book Title");

String BT1 = in.next();

**if** (!books.containsKey(BT1)) {

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

"Enter Book Details :Category(NOVEL, BIOGRAPHY, TECHNICAL, BUSINESS, FINANCE),"

+ "Price,PublishDate(yyyy-mm-dd),AuthorName,Qty");

books.put(BT1, **new** Books(BT1, Category.*valueOf*(in.next().toUpperCase()), in.nextDouble(),

LocalDate.*parse*(in.next()), in.next(), in.nextInt()));

System.***out***.println("Books Added Succesfully");

} **else**

**throw** **new** CustomHandlingException("Invalid Book details");

**break**;

**case** 2:

System.***out***.println("All Books Details");

**for** (Books b : books.values()) {

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

}

**break**;

**case** 3:

// Remove Book

System.***out***.println("Enter Book Title");

Books RemoveBook = books.remove(in.nextLine());

**if** (RemoveBook == **null**)

**throw** **new** CustomHandlingException("Invalid book title,Cant fid Book");

System.***out***.println("Book Removed" + RemoveBook);

**break**;

**case** 100:

// storeBookDetails(fileName, books);

**break**;

}

} **catch** (Exception e) {

e.printStackTrace();

in.nextLine();

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

Ans –



**ASSIGNMENT 8 - Threads**

**Bank Class -**

**package** com.thread;

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

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

**public** BankAccount(**double** balance) {

**this**.balance = balance;

}

@Override

**public** String toString() {

**return** "BankAccount [balance=" + balance + "]";

}

**public** **synchronized** **void** withdraw(**double** amount) **throws** InterruptedException {

System.***out***.println("started by : " + Thread.*currentThread*().getName());

**if** (balance < amount) {

Thread.*sleep*(1000);

System.***out***.println("Waiting for deposite");

**super**.wait();

}

**this**.balance = **this**.balance - amount;

System.***out***.println("after withdraw " + balance);

System.***out***.println("Ended by : " + Thread.*currentThread*().getName());

}

**public** **synchronized** **void** deposite(**double** amount) {

System.***out***.println("Started by :" + Thread.*currentThread*().getName());

**this**.balance = **this**.balance + amount;

System.***out***.println("after deposite : " + balance);

System.***out***.println("Ended by : " + Thread.*currentThread*().getName());

**super**.notify();

}

}

Deposite Thread –

**package** com.thread;

**public** **class** DepositeThread **extends** Thread {

**private** BankAccount b1;

**public** DepositeThread(BankAccount b1) {

**this**.b1 = b1;

}

@Override

**public** **void** run() {

System.***out***.println("---------started--------" + Thread.*currentThread*().getName());

System.***out***.println("Enter Deposite Amount");

b1.deposite(6000);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("---------ended----------" + Thread.*currentThread*().getName());

}

}

WithDraw Thread -

**package** com.thread;

**public** **class** WithdrawThread **extends** Thread {

**private** BankAccount b1;

**public** WithdrawThread(BankAccount b1) {

**this**.b1 = b1;

}

@Override

**public** **void** run() {

System.***out***.println("--------------started ---------" + Thread.*currentThread*().getName());

**try** {

System.***out***.println("Enter withdraw Amount");

b1.withdraw(5000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("---------------ended-----------" + Thread.*currentThread*().getName());

}

}

Tester –

**package** com.thread;

**import** java.util.Scanner;

**public** **class** TestThread {

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

Scanner in = **new** Scanner(System.***in***);

System.***out***.println("Enter initial Balance");

BankAccount b1 = **new** BankAccount(in.nextDouble());

WithdrawThread w1 = **new** WithdrawThread(b1);

w1.setName("WithDraw Thread");

DepositeThread d1 = **new** DepositeThread(b1);

d1.setName("deposite Thread");

w1.start();

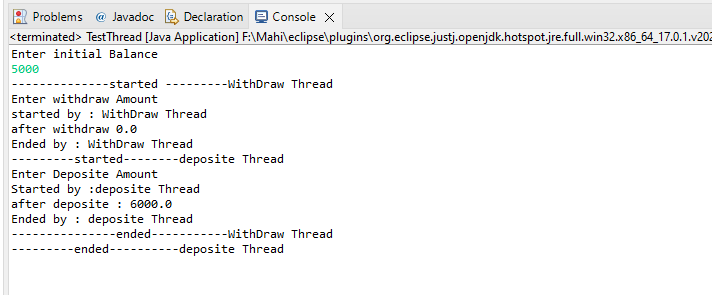
d1.start();

in.close();

}

}

Ans -



**ASSIGNMENT 9 – Even Odd Threads**

**Even Class -**

**package** runnable\_tasks;

**public** **class** EvenPrinterTask **implements** Runnable {

// state :

**private** **int** begin;

**private** **int** end;

**public** EvenPrinterTask(**int** begin, **int** end) {

**super**();

**this**.begin = begin;

**this**.end = end;

System.***out***.println("ctor invoked by " + Thread.*currentThread*().getName());

}

@Override

**public** **void** run() {

System.***out***.println(Thread.*currentThread*().getName() + " strted exec");

**try** {

**for** (**int** i = begin; i <= end; i++) {

**if** (i % 2 == 0)

System.***out***.println("Even No " + i + " printed by " + Thread.*currentThread*().getName());

Thread.*sleep*(100);

}

} **catch** (Exception e) {

System.***out***.println(Thread.*currentThread*().getName() + " got exc " + e);

}

System.***out***.println(Thread.*currentThread*().getName() + " exec over");

}

}

Odd Class –

**package** runnable\_tasks;

**public** **class** OddPrinterTask **implements** Runnable {

// state :

**private** **int** begin;

**private** **int** end;

**public** OddPrinterTask(**int** begin, **int** end) {

**super**();

**this**.begin = begin;

**this**.end = end;

System.***out***.println("ctor invoked by " + Thread.*currentThread*().getName());

}

@Override

**public** **void** run() {

System.***out***.println(Thread.*currentThread*().getName() + " strted exec");

**try** {

**for** (**int** i = begin; i <= end; i++) {

**if** (i % 2 != 0)

System.***out***.println("Odd No " + i + " printed by " + Thread.*currentThread*().getName());

Thread.*sleep*(130);

}

} **catch** (Exception e) {

System.***out***.println(Thread.*currentThread*().getName() + " got exc " + e);

}

System.***out***.println(Thread.*currentThread*().getName() + " exec over");

}

}

Tester –

**package** tester;

**import** java.util.Scanner;

**import** runnable\_tasks.EvenPrinterTask;

**import** runnable\_tasks.OddPrinterTask;

**public** **class** TestConcurrency {

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

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

System.***out***.println("Enter range");

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

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

// Create task instance n attach it to thrd

// THread(Runnable target,String name)

Thread t1 = **new** Thread(**new** EvenPrinterTask(start, end), "even");// NEW

Thread t2 = **new** Thread(**new** OddPrinterTask(start, end), "odd");// NEW

t1.start();

t2.start();// main , even, odd

System.***out***.println("main waiting for child thrds to finish exec");

t1.join();

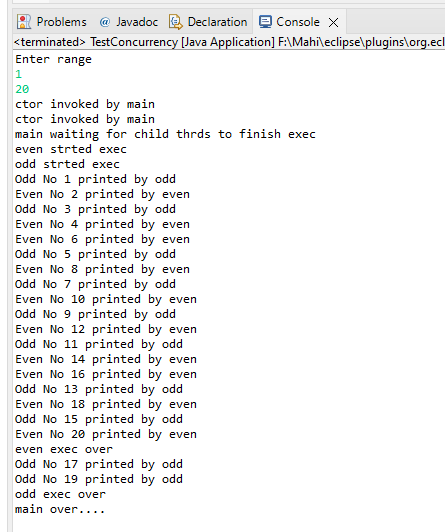
t2.join();

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

} **catch** (Exception e) {

e.printStackTrace();

}



}

}

**ASSIGNMENT 10**

**Employee Class -**

**package** com.code;

**public** **class** Employee **implements** Cloneable{

//data members:fields

**private** **int** empid;

**private** String name,address;

**private** **double** salary;

**public** Employee(**int** empid, String name, String address, **double** salary) {

**super**();

**this**.empid = empid;

**this**.name = name;

**this**.address = address;

**this**.salary = salary;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

@Override

**public** String toString() {

**return** "Employee [empid=" + empid + ", name=" + name + ", address=" + address + ", salary=" + salary + "]";

}

@Override

**protected** Object clone() **throws** CloneNotSupportedException {

System.***out***.println("Calling Object clone method");

//return super.clone();//Deep cloning done by Object

Employee emp=**new** Employee(**this**.empid, **this**.name, **this**.address, **this**.salary);

**return** emp;//deep cloning

//return this;//shallow cloning

}

}

Tester Class –

**package** com.code;

**import** java.util.Date;

**public** **class** TestCloning {

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

Employee emp1 = **new** Employee(101, "Snehal", "Pune", 80000);

//emp1.

// emp.

System.***out***.println("Emp1:" + emp1);

Employee emp2 = (Employee) emp1.clone();

System.***out***.println("Emp2:" + emp2);

System.***out***.println("Emp1:" + emp1.hashCode());

System.***out***.println("Emp2:" + emp2.hashCode());

emp1.setName("Sonali");// change name in orignal object

System.***out***.println("Emp1:" + emp1);

System.***out***.println("Emp2:" + emp2);

Date dt = **new** Date();

Employee emp3 = emp1;

}

}

ANS –

