**ASSIGNMENT OOP WITH JAVA**

DAY5

**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;})**

*Solution*

**package** com.org.app;

**public** **class** Emp {

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

**private** String Name;

**private** String deptId;

**private** **double** basic;

**public** **double** getBasic() {

**return** basic;

}

**public** Emp(**int** id, String name, String deptId, **double** basic) {

**super**();

**this**.id = id;

Name = name;

**this**.deptId = deptId;

**this**.basic = basic;

}

**public** **double** netSalary()

{

**return** 0;

}

@Override

**public** String toString() {

**return** " [id=" + id + ", Name=" + Name + ", deptId=" + deptId + ", basic\_sal=" + basic + "]";

}

}

**package** com.org.app;

**import** inheritance.Person;

**public** **class** Manager **extends** Emp {

**private** **double** permomanceBonus;

**public** Manager(**int** id, String name, String deptId, **double** basic, **double** permomanceBonus) {

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

**this**.permomanceBonus = permomanceBonus;

}

@Override

**public** String toString() {

**return** "manager details="+ **super**.toString()+"[permomanceBonus=" + permomanceBonus + "]";

}

@Override

**public** **double** netSalary()

{

**return** getBasic()+permomanceBonus;

}

**public** **double** getPermounceBonus()

{

**return** permomanceBonus;

}

}

**package** com.org.app;

**public** **class** Worker **extends** Emp {

**private** **int** hrsworked;

**private** **int** hrsrate;

**public** Worker(**int** id, String name, String deptId, **double** basic\_sal, **int** hrsworked, **int** hrsrate) {

**super**(id, name, deptId, basic\_sal);

**this**.hrsworked = hrsworked;

**this**.hrsrate = hrsrate;

}

@Override

**public** String toString() {

**return** "worker details="+**super**.toString()+"[hrsworked=" + hrsworked + ", hrsrate=" + hrsrate + "]";

}

@Override

**public** **double** netSalary()

{

**return** getBasic()+(hrsrate\*hrsworked);

}

**public** **int** gethrsrate()

{

**return** hrsrate;

}

}

package tester;

import java.util.Scanner;

import com.org.app.Emp;

import com.org.app.Manager;

import com.org.app.Worker;

public class TestEmp {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("how much reqiurement?");

Emp[] e = new Emp[sc.nextInt()];//it is array ref refering to array object.which can hold emp type ref

int counter = 0, choice;

boolean flag = true;

while (flag != false) {

System.out.println("===========WELCOME============ ");

System.out.println("1-Hire Manager + 2-Hire Worker + 3-Display Information of Employee + 10-exits");

choice = sc.nextInt();

switch (choice) {

case 1:

// hire manager

if (counter < e.length) {

System.out.println("enetr manager details: id name deptid basic performanceBonus");

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

} else {

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

}

break;

case 2:

if (counter < e.length) {

System.out.println("enetr worker details: id name deptid basic hrsworks,hrsrate");

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

sc.nextInt());

} else {

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

}

break;

case 3:

for (Emp emp : e) {

if (emp != null) {

if (emp instanceof Manager) {

System.out.println(emp);

System.out.println("Total net salary of manager =" + emp.netSalary());

System.out.println(" Performanus bonus=" + ((Manager) emp).getPermounceBonus());

} else if (emp instanceof Worker) {

System.out.println(emp);

System.out.println("Total net salary of Worker is=" + emp.netSalary());

System.out.println("Houraly rate=" + ((Worker) emp).gethrsrate());

}

}

}

break;

case 10:

flag = false;

break;

default:

System.out.println("Invalid choice..... Please select correct choice!!!!!!!!!");

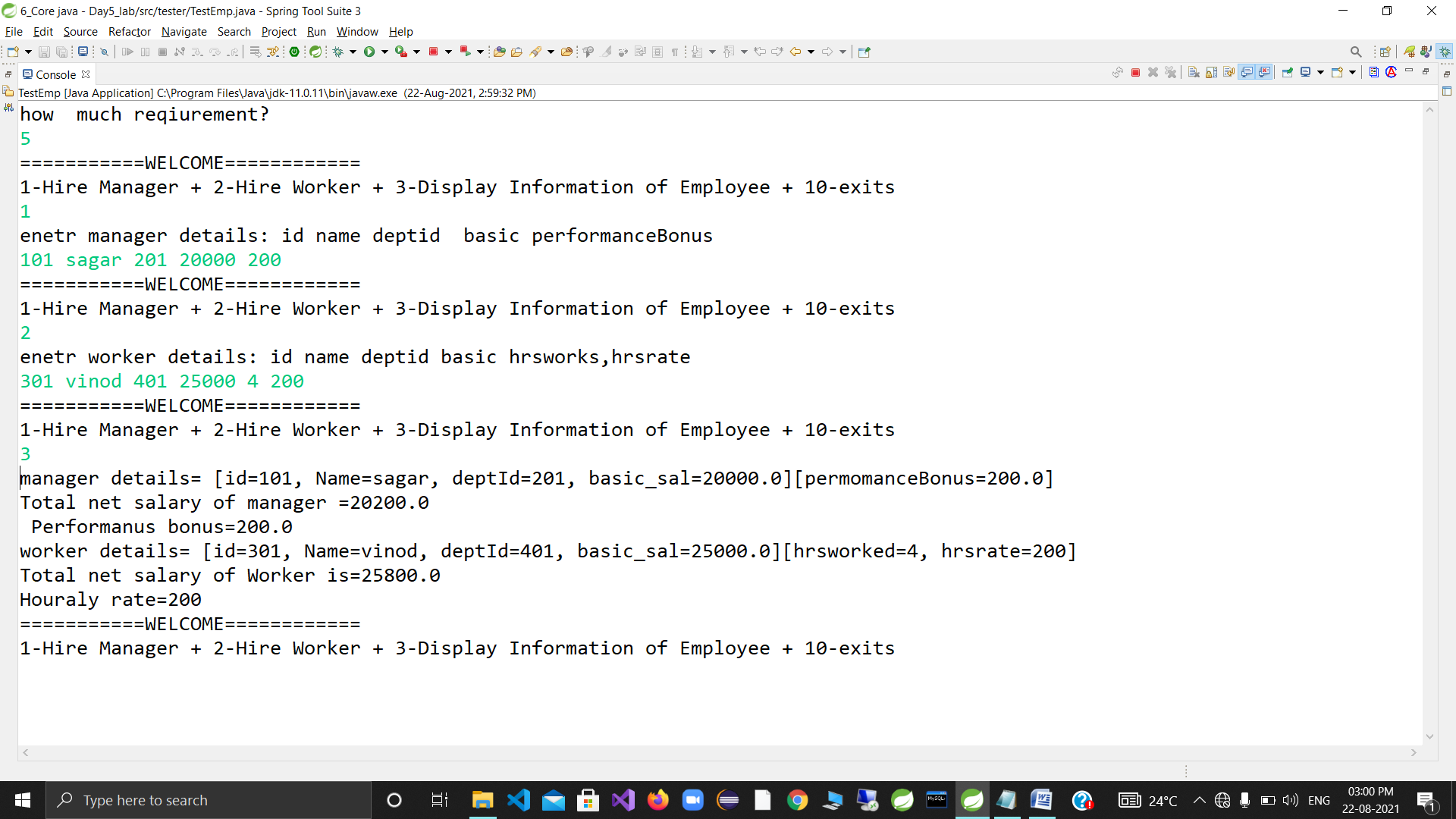
break;

}

}

}

}



**package** inheritance;

**public** **class** Person {

**private** String fname;

**private** String lname;

**public** Person(String fname, String lname)

{

**super**();

**this**.fname = fname;

**this**.lname = lname;

}

@Override

**public** String toString() {

**return** " [fname=" + fname + ", lname=" + lname + "]";

}

**public** String getFname() {

**return** fname;

}

**public** String getLname() {

**return** lname;

}

}

**package** inheritance;

**public** **class** Faculty **extends** Person {

**private** **int** expriyears;

**private** String expinsubject;

**public** Faculty( String fname,String lname, **int** expriyears,String expinsubject)

{

**super**(fname,lname);

**this**.expriyears=expriyears;

**this**.expinsubject=expinsubject;

}

**public** String toString()

{

**return** "faculty details="+**super**.toString()+" "+"exprince year="+expriyears+" "+"expertin subject="+expinsubject;

}

**public** **void** teach()

{

System.***out***.println(getLname()+"=are teaching");

}

}

**package** inheritance;

**public** **class** Student **extends** Person {

**private** **int** gry;

**private** String cname;

**private** **double** fees;

**private** **int** marks;

**public** Student(String fname, String lname, **int** gry, String cname, **double** fees, **int** marks) {

**super**(fname, lname);

**this**.gry = gry;

**this**.cname = cname;

**this**.fees = fees;

**this**.marks = marks;

}

@Override

**public** String toString()

{

**return** "student details="+**super**.toString()+"gradution year="+gry+"cname="+"fees="+"marks="+marks;

}

**public** **void** study()

{

System.***out***.println(getFname()+"=are studing "+cname);

}

}

package tester;

import inheritance.Faculty;

import inheritance.Person;

import inheritance.Student;

public class TestPerson {

public static void main(String[] args) {

//Person per=new Student("vikas", "yedave", 2019, "edac", 600000, 200);

//System.out.println(per);

Person per=new Faculty("yogesh", "karande", 2, "math");

System.out.println(per);

//((Faculty )per).teach();

//per=new Student("vikas", "yedave", 2019, "edac", 600000, 200);

//((Student) per).study();

//((Faculty)per).teach();

if(per instanceof Faculty)

{

((Faculty)per).teach();

}

else if(per instanceof Student)

{

((Student) per).study();

}

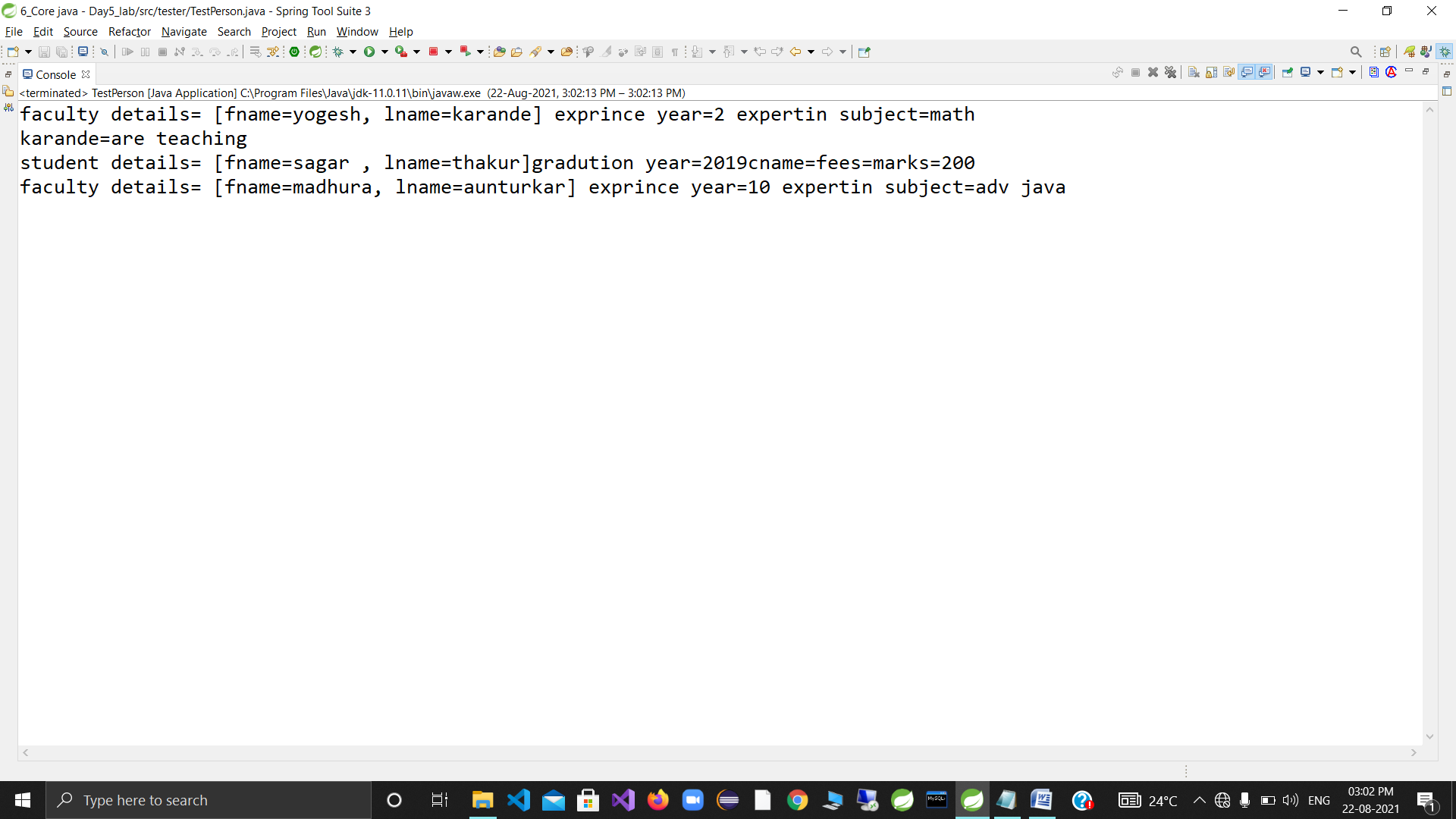
Person []people= {new Student("sagar ", "thakur", 2019, "edac", 200000, 200),new Faculty("madhura", "aunturkar", 10, "adv java")};

for(Person p:people)

System.out.println(p);

}

}



Day 6-

4. 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

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))

*SOLUTION*

**package** com.app.vehicle;

**public** **class** Vehicle {

**private** **int** regNo;

**private** String color;

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

@Override

**public** String toString() {

**return** "Vehicle [regNo=" + regNo + ", color=" + color + ", price=" + price + "]";

}

**public** Vehicle(**int** regNo, String color, **double** price)

{

**super**();

**this**.regNo = regNo;

**this**.color = color;

**this**.price = price;

}

@Override

**public** **boolean** equals(Object anotherObject)

{

System.***out***.println("in vehical equal");

**if**(anotherObject **instanceof** Vehicle )

{

//unique id=regNo=> primary key

**return** **this**.regNo==((Vehicle) anotherObject).regNo;//downcasting

}

**return** **false**;

}

// public boolean isEqual(Vehicle ano)

// {

//

// return this.regNo==ano.regNo;

// }

}

package driver;

import java.util.Scanner;

import com.app.vehicle.Vehicle;

public class TestVehicle {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

// System.out.println("enetr 1st vehicale detils: regNo color price");

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

// System.out.println("enetr 2st vehicale detils: regNo color price");

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

//

// System.out.println(v1.equals(v2)?"Same":"Diff");

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

System.out.println("enetr 1st vehicale detils: regNo color price");

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

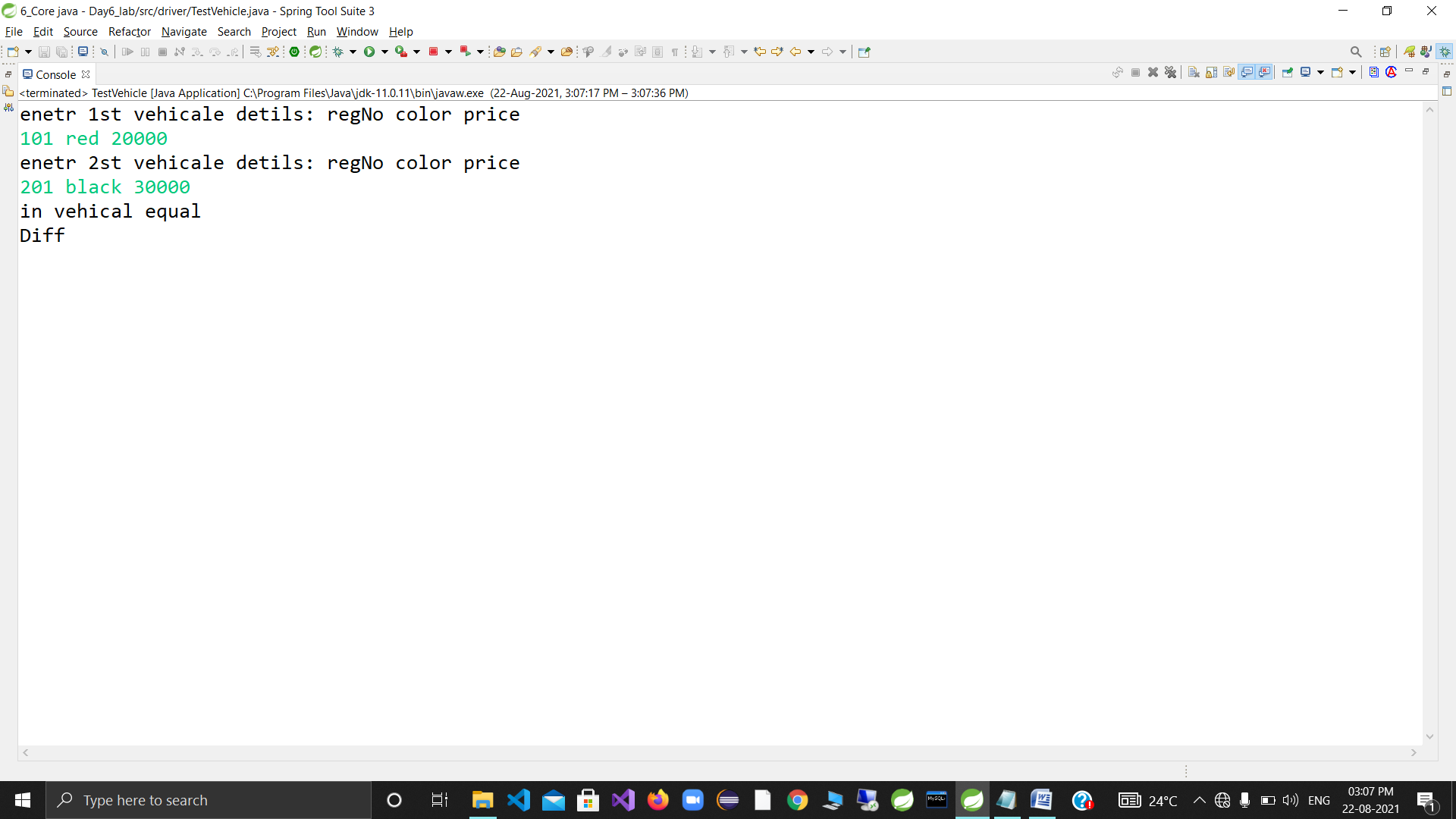
System.out.println("enetr 2st vehicale detils: regNo color price");

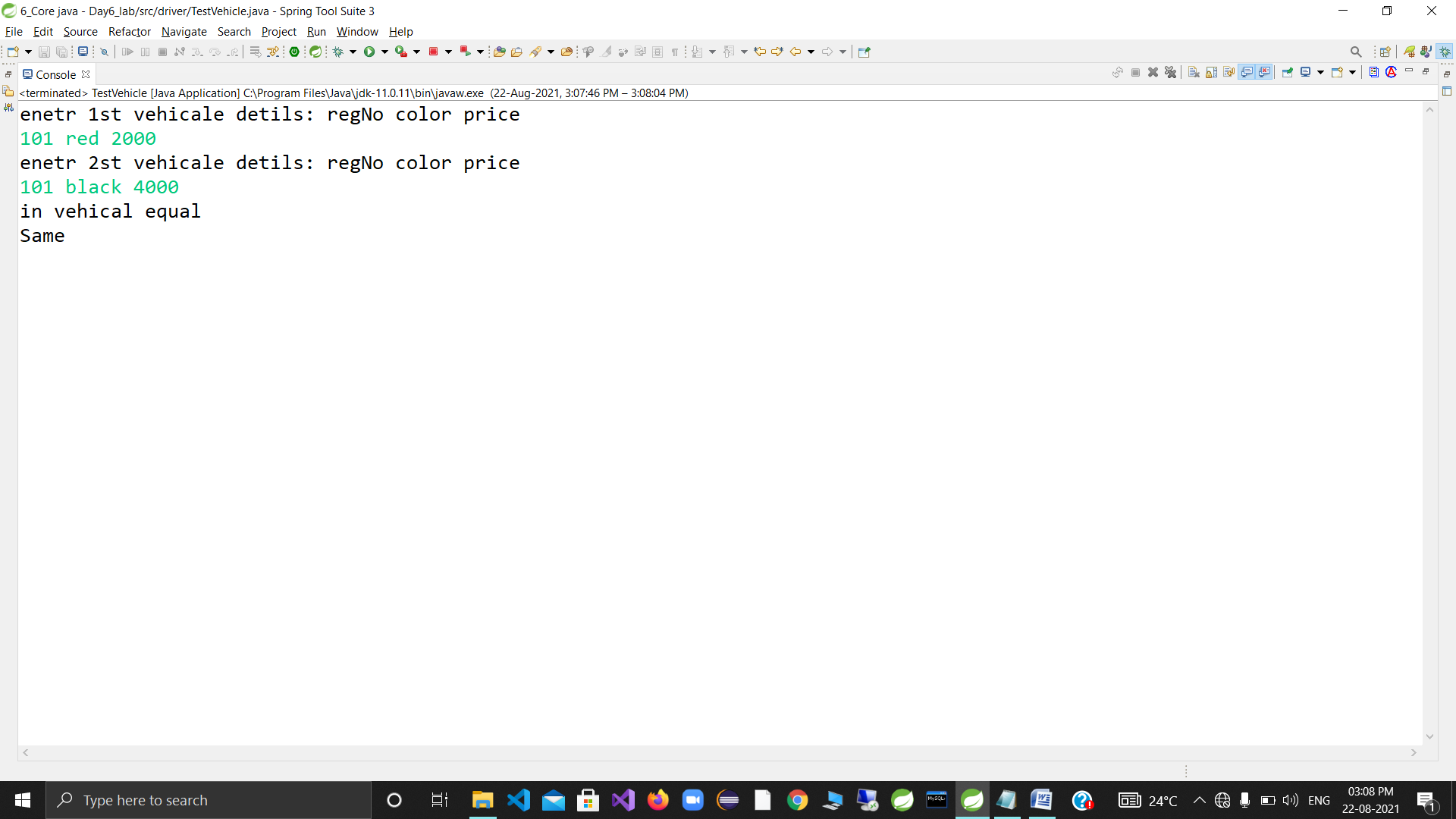
Object v2=new Vehicle(sc.nextInt(),sc.next(),sc.nextDouble());

System.out.println(v1.equals(v2)?"Same":"Diff");//jvm take type of object vehical

}

}





Day 7-

**package** com.app.vehicle;

**public** **class** Vehicle {

**private** **int** regNo;

**private** String color;

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

@Override

**public** String toString() {

**return** "Vehicle [regNo=" + regNo + ", color=" + color + ", price=" + price + "]";

}

**public** Vehicle(**int** regNo, String color, **double** price)

{

**super**();

**this**.regNo = regNo;

**this**.color = color;

**this**.price = price;

}

@Override

**public** **boolean** equals(Object anotherObject)

{

System.***out***.println("in vehical equal");

**if**(anotherObject **instanceof** Vehicle )

{

//unique id=regNo=> primary key

Vehicle v=(Vehicle) anotherObject;//downcasting

//return this.regNo== v.regNo && this.color==v.color ;//anotther location

**return** **this**.regNo== v.regNo && **this**.color.equals(v.color) ;

}

**return** **false**;

}

**public** Vehicle(**int** regNo, String color) {

**super**();

**this**.regNo = regNo;

**this**.color = color;

}

// public boolean isEqual(Vehicle ano)

// {

//

// return this.regNo==ano.regNo;

// }

}

**package** custom.exception;

@SuppressWarnings("serial")

**public** **class** VehicalHandlingException **extends** Exception {

**public** VehicalHandlingException( String errMassgee)

{

**super**(errMassgee);

}

}

package tester;

import java.util.Scanner;

import com.app.vehicle.Vehicle;

import utils.ValidationRules;

import utils.VehicalValidationRule;

public class TestVehicalCustomException {

public static void main(String[] args) {

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

int choice, registrationNumber, counter = 0;

double price;

String color;

System.out.println("How many vehicles you want to enter");

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

do {

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

System.out.println("1-Add a Vhehicle");

System.out.println("2-Display All Vehicles");

System.out.println("3-Exit");

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

choice = sc.nextInt();

switch (choice) {

case 1:

boolean value = true;

System.out.println("Enter Vehicle Registration Number");

registrationNumber = sc.nextInt();

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

color = sc.next();

System.out.println("Enter Vehicle Price");

price = sc.nextDouble();

try {

value = VehicalValidationRule.checkDuplicateVehicle(vh, registrationNumber, color);

}

catch (Exception e) {

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

}

if (value==false) {

vh[counter++] = new Vehicle(registrationNumber, color, price);

}

break;

case 2:

for (Vehicle v : vh)

if(v!=null)

System.out.println(v);

break;

case 3:

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

break;

default:

System.out.println("Invalid choice...Please select choice...");

break;

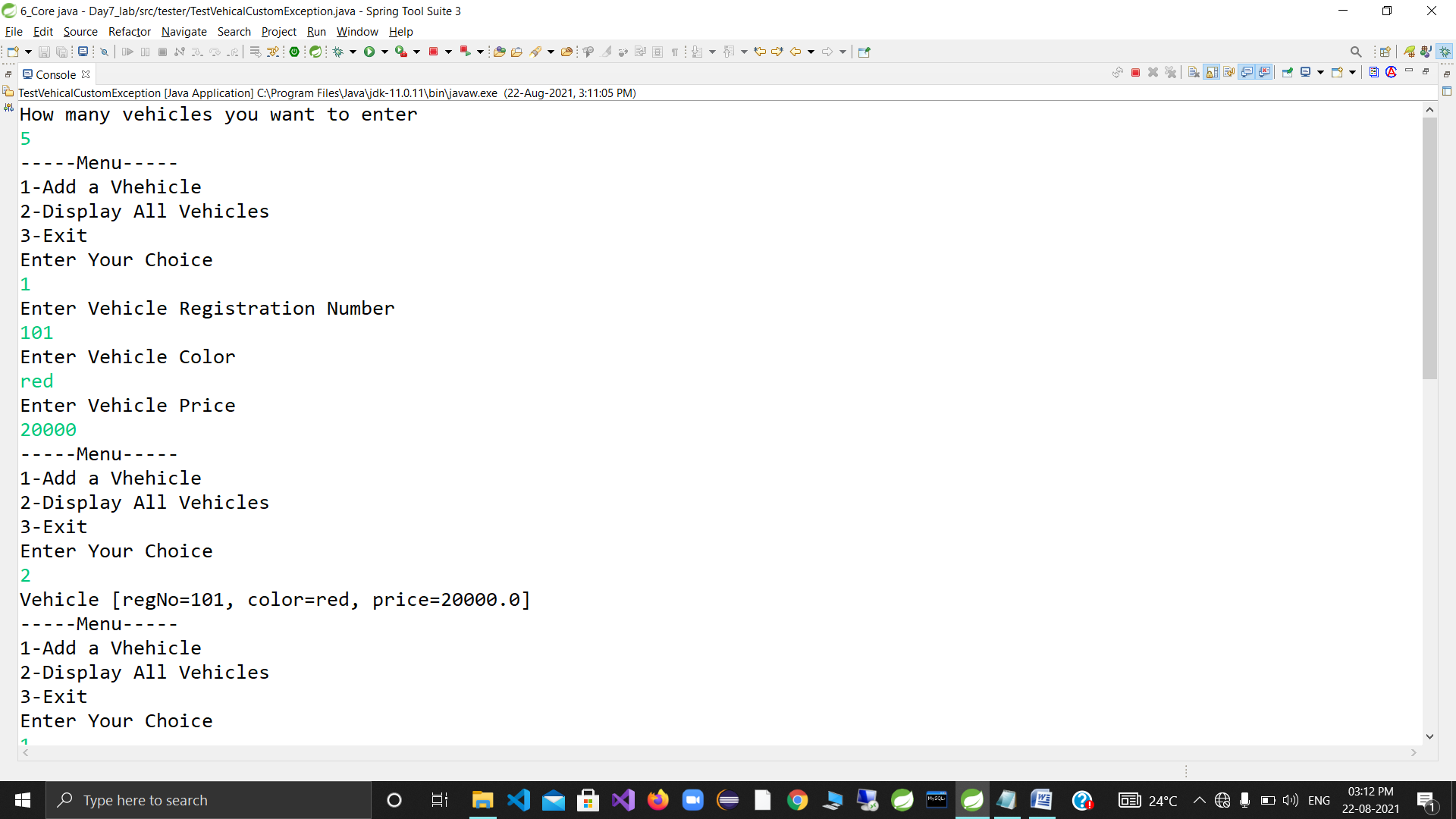
}

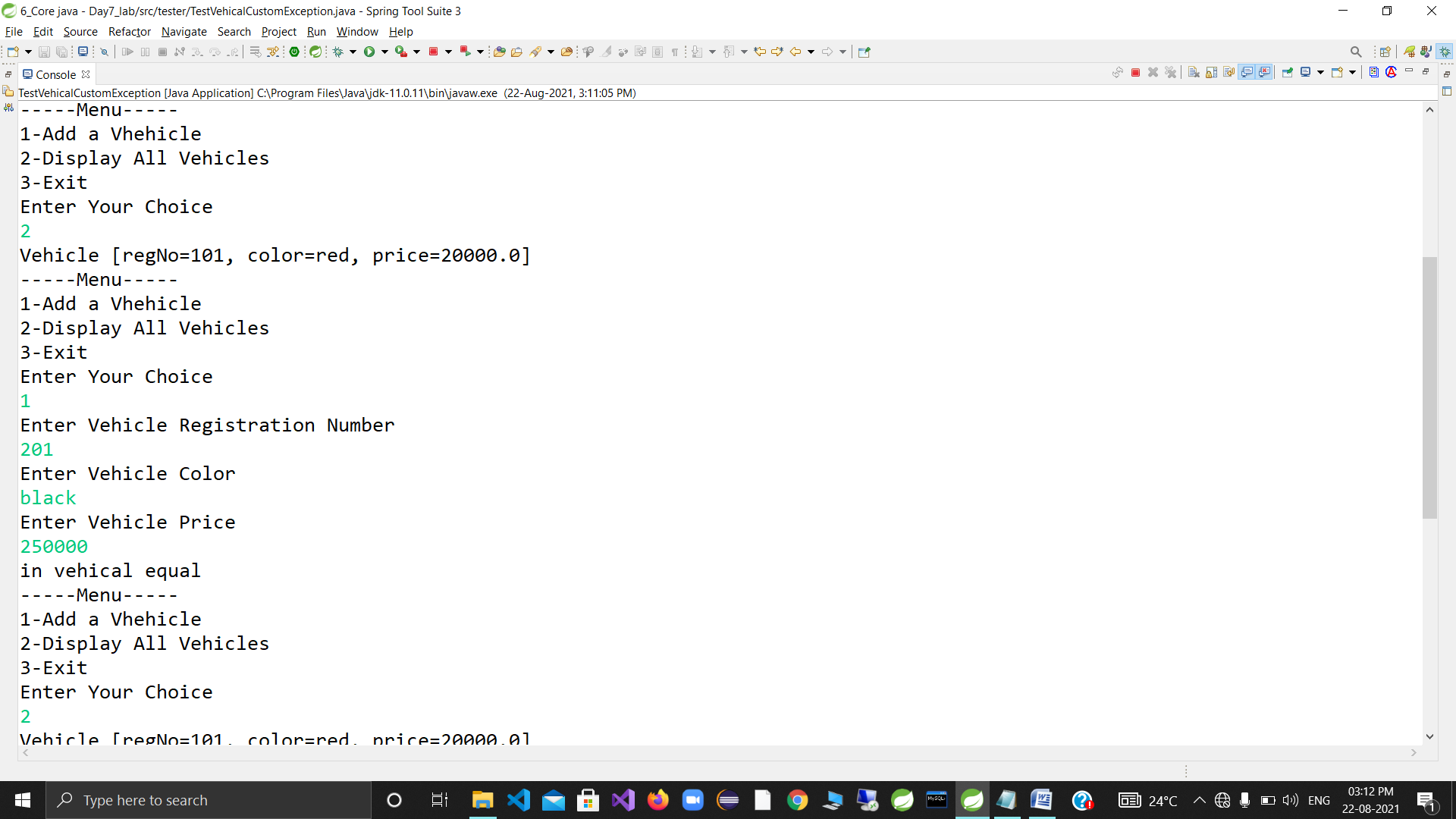
} while (choice != 3);

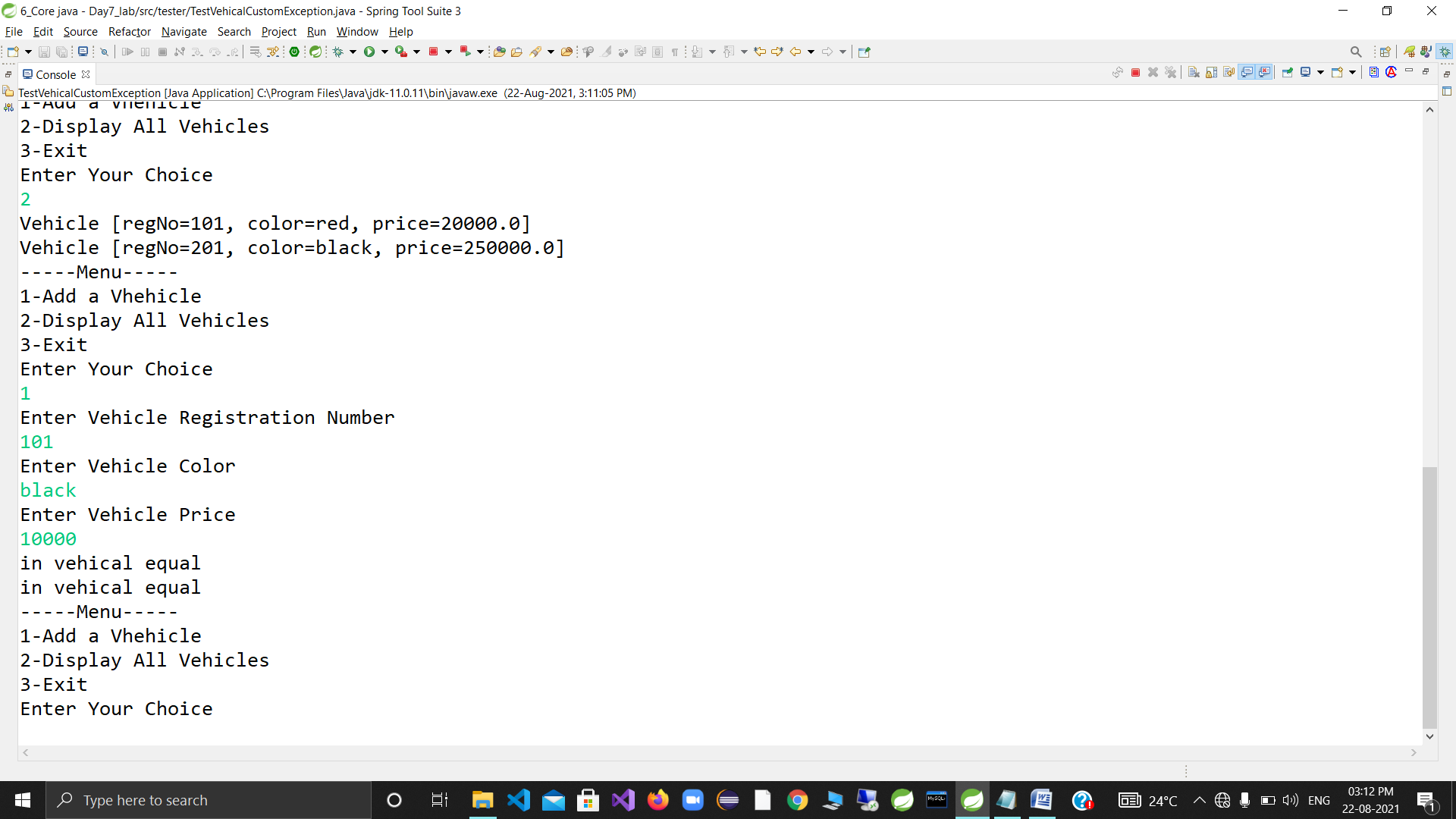
}

}

}







DAY8

Day 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

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

4.2 Create custom exception class in a separate package

CustomerHandlingException

4.3 Create in "utils" package : ValidationRules

Rules

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

password must be min 4 max 10 chars long

reg amount should be multiples of 500

dob should be before 1st Jan 1995

Add 2 different static methods : validateEmail n validatePassword

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

eg : convertDate

4.4 Create a simple tester : Tester1 (no while loop , no switch/case)

MUST use try-with-resources to create scanner

Prompt for customer details

Validate customer details .

Iff it's valid , create customer instance , display customer details(via toString) , o.w display err mesg via catch block 8

*SOLUTION*

package com.app.customer;

import java.util.Date;

import java.text.SimpleDateFormat;

public class Customer {

private String name;

private String email;

private String pass;

private double regAmount;

private Date dob;

public static SimpleDateFormat sdf;

static

{

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

}

public Customer(String name, String email, String pass, double regAmount, Date dob) {

super();

this.name = name;

this.email = email;

this.pass = pass;

this.regAmount = regAmount;

this.dob=dob;

}

@Override

public String toString() {

return "Customer [name=" + name + ", email=" + email + ", pass=" + pass + ", regAmount=" + regAmount + ", dob="

+ sdf.format(dob) + "]";

}

@Override

public boolean equals(Object anotherObject)

{

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

if(anotherObject instanceof Customer)

{

Customer c=(Customer)anotherObject;

return this.email==c.email;

}

return false;

}

}

**package** custom\_exception;

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

**public** CustomerHandlingException(String errMassage)

{

**super**(errMassage);

}

}

package utils;

import com.app.customer.Customer;

import java.util.Date;

import java.text.ParseException;

import custom\_exception.CustomerHandlingException;

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

public class ValidationRules {

public static final int MIN\_LENGTH;

public static final int MAX\_LENGTH;

public static Date thresholdDate;

static

{

MIN\_LENGTH=4;

MAX\_LENGTH=10;

try {

thresholdDate=sdf.parse("1-1-1995");

} catch (ParseException p) {

System.out.println(" error in static intialisation block....");

}

}

public static String validateEmail(String email) throws CustomerHandlingException

{

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

{

return email;

}

throw new CustomerHandlingException("Invalid email");

}

public static String validatepass(String pass) throws CustomerHandlingException

{

if(pass.length()<MIN\_LENGTH || pass.length()>10)

{

throw new CustomerHandlingException("Password length not matched as per req....");

}

return pass;

}

public static double validateRegAmount(double regAmt) throws CustomerHandlingException

{

if(regAmt % 500!=0)

{

throw new CustomerHandlingException("Invalid amount reg....");

}

return regAmt;

}

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

{

Date d1=sdf.parse(dob);

if(d1.after(thresholdDate))

{

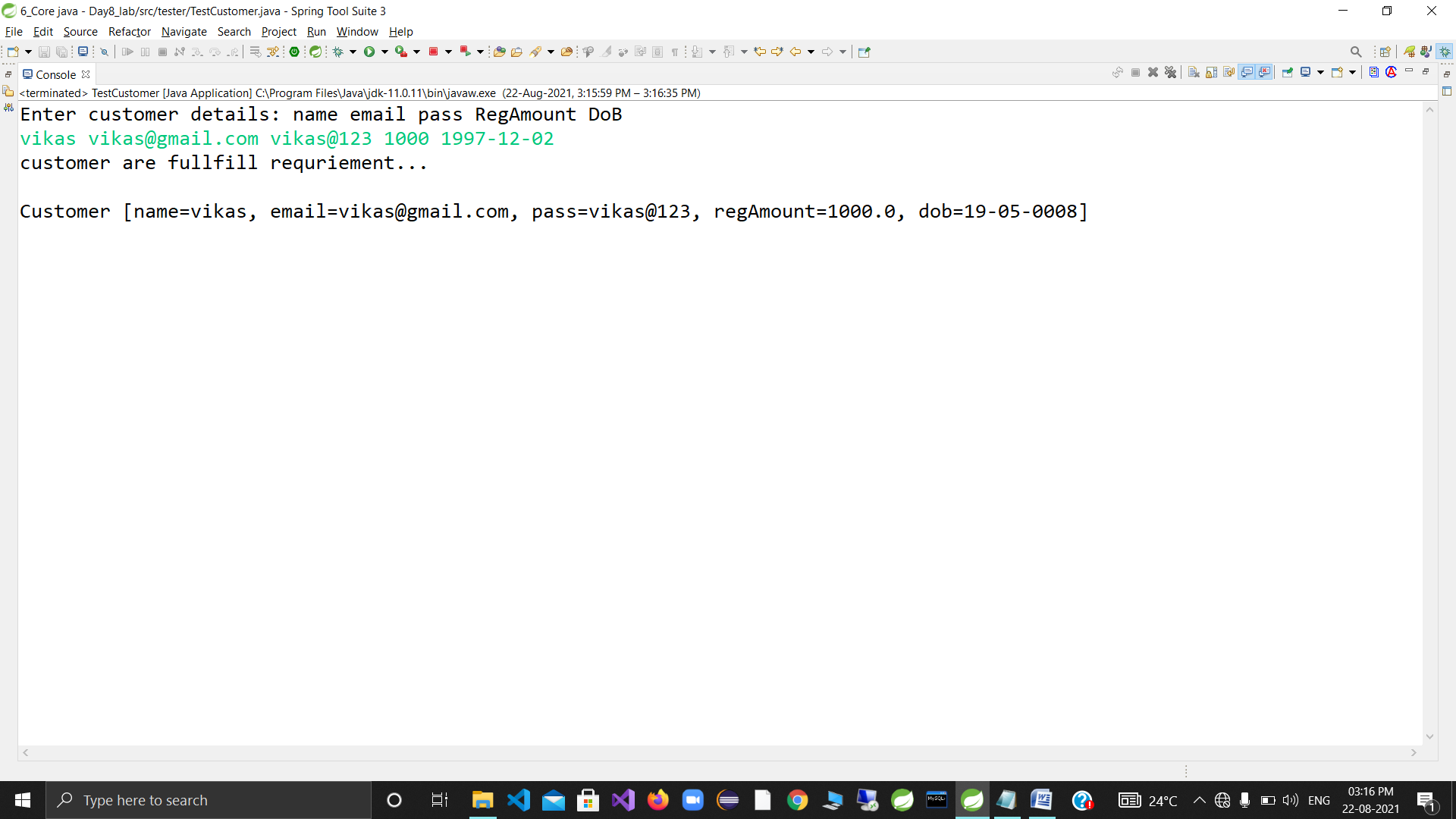
throw new CustomerHandlingException("Invalid date...");

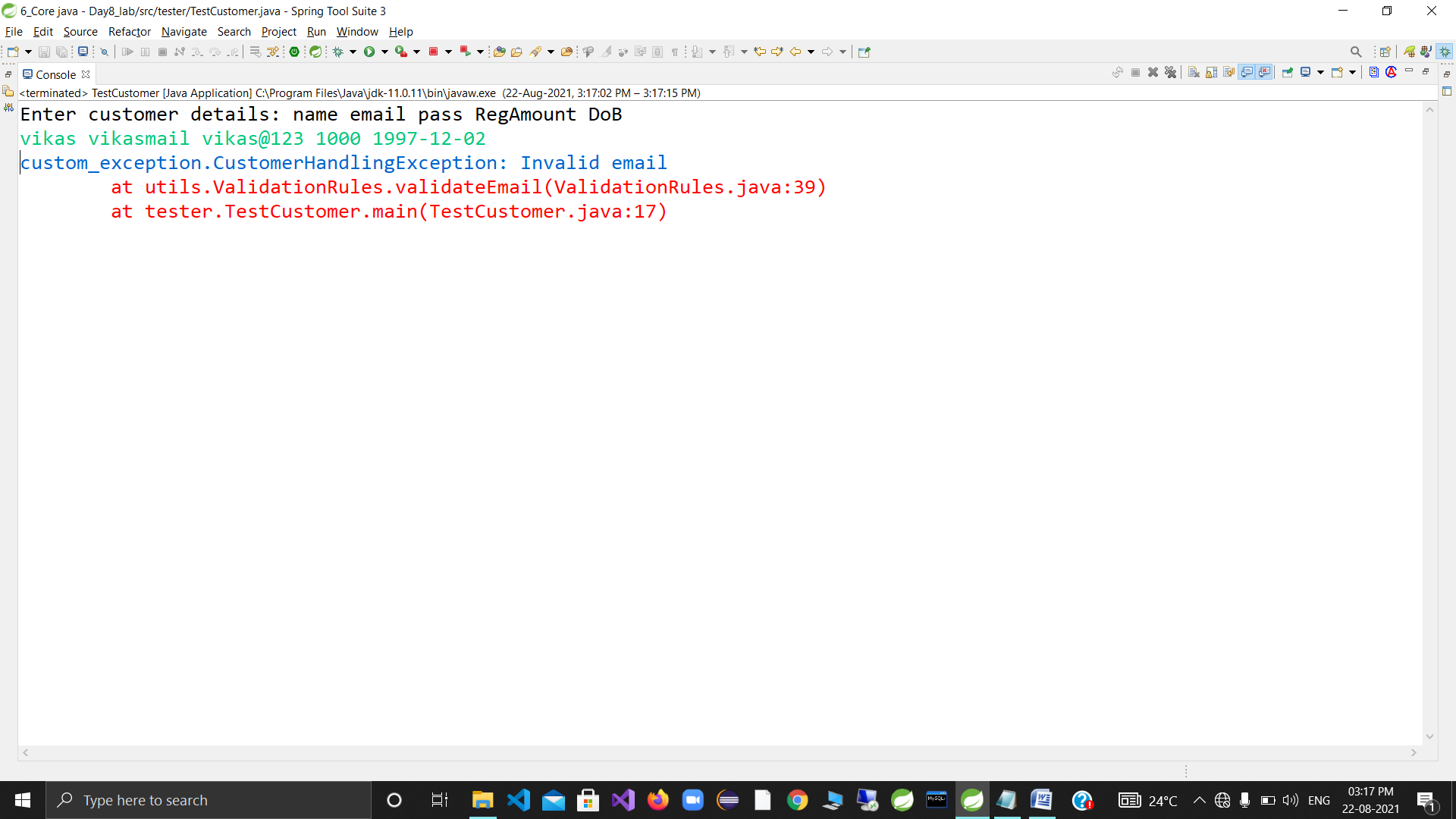
}

return d1;

}

}





Day 9

3.2 Customers may belong to different categories , depending upon their reg amount(they can avail different type of services)

Create CustomerCategory as an enum.

Add these constants : SILVER,GOLD,DIAMOND,PLATINUM

Add HAS-A relationship between Customer & CustomerCategory

(Customer HAS-A CustomerCategory)

package com.app.customer;

import java.util.Date;

import enums.CustomerCatagory;

import java.text.SimpleDateFormat;

public class Customer {

private String name;

private String email;

private String pass;

private double regAmount;

private Date dob;

public static SimpleDateFormat sdf;

private CustomerCatagory type;

static

{

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

}

public Customer(String name, String email, String pass, double regAmount, Date dob ,CustomerCatagory type) {

super();

this.name = name;

this.email = email;

this.pass = pass;

this.regAmount = regAmount;

this.dob=dob;

this.type=type;

}

@Override

public String toString() {

return "Customer [name=" + name + ", email=" + email + ", pass=" + pass + ", regAmount=" + regAmount + ", dob="

+ sdf.format(dob) +"customer type="+type+"]";

}

@Override

public boolean equals(Object anotherObject)

{

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

if(anotherObject instanceof Customer)

{

Customer c=(Customer)anotherObject;

return this.email==c.email;

}

return false;

}

}

**package** custom\_exception;

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

**public** CustomerHandlingException(String errMassage)

{

**super**(errMassage);

}

}

**package** enums;

**public** **enum** CustomerCatagory {

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

@Override

**public** String toString()

{

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

}

}

package tester;

import java.util.Date;

import java.util.Scanner;

import static utils.ValidationRules.\*;

import com.app.customer.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 pass RegAmount DoB customer catagory");

Customer c1= new Customer(sc.next(), validateEmail(sc.next()), validatepass(sc.next()),

validateRegAmount(sc.nextDouble()), validateDate(sc.next()),validatecustomerCatagory(sc.next()));

System.out.println("customer are fullfill requriement...");

System.out.println();

System.out.println(c1);

} catch (Exception e) {

e.printStackTrace();

}

}

}

//String name,email,pass;

//double regAmt;

//Date Dob;

//System.out.println("enetr name,email,pass, regAmt,dob");

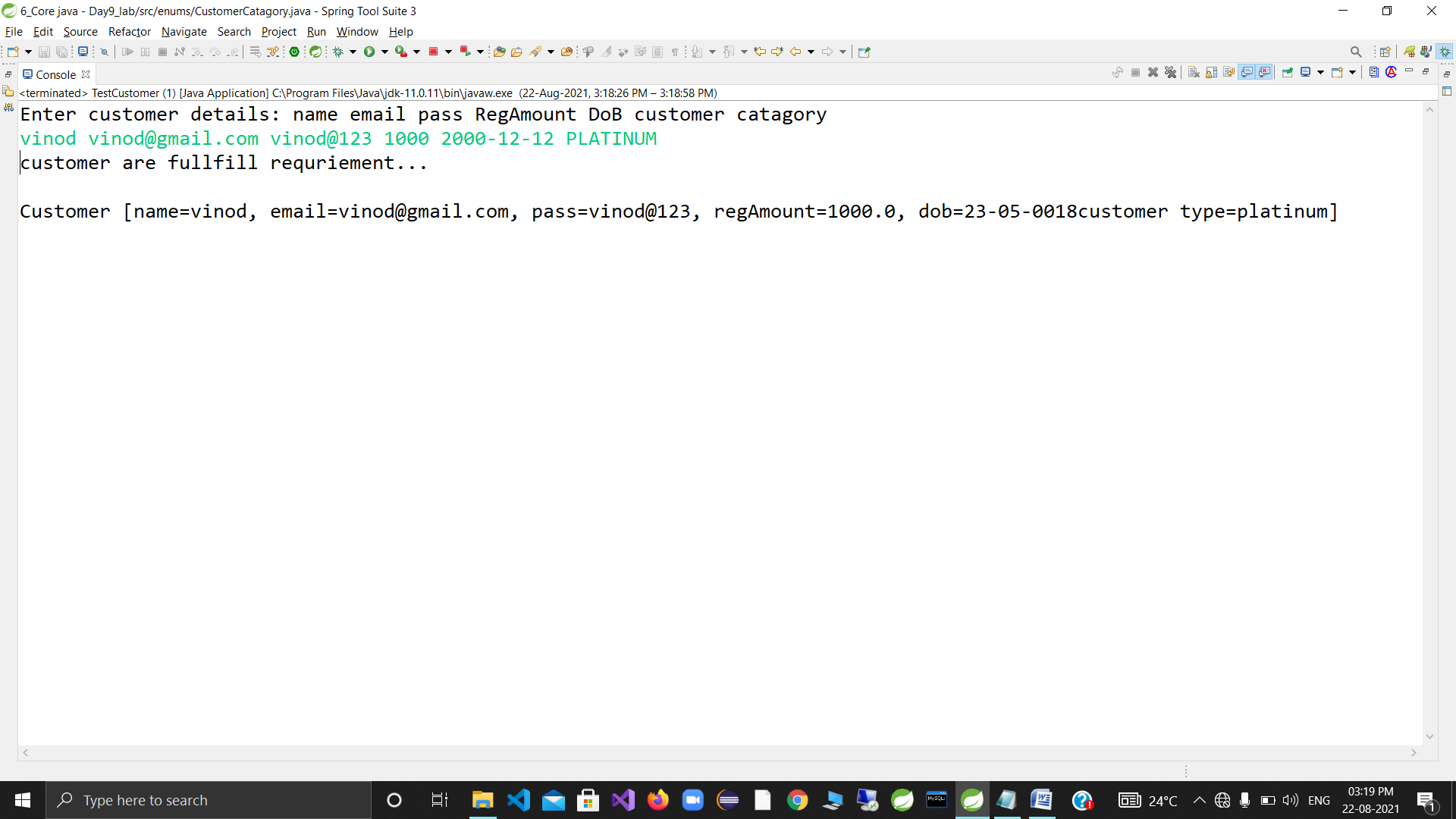
//

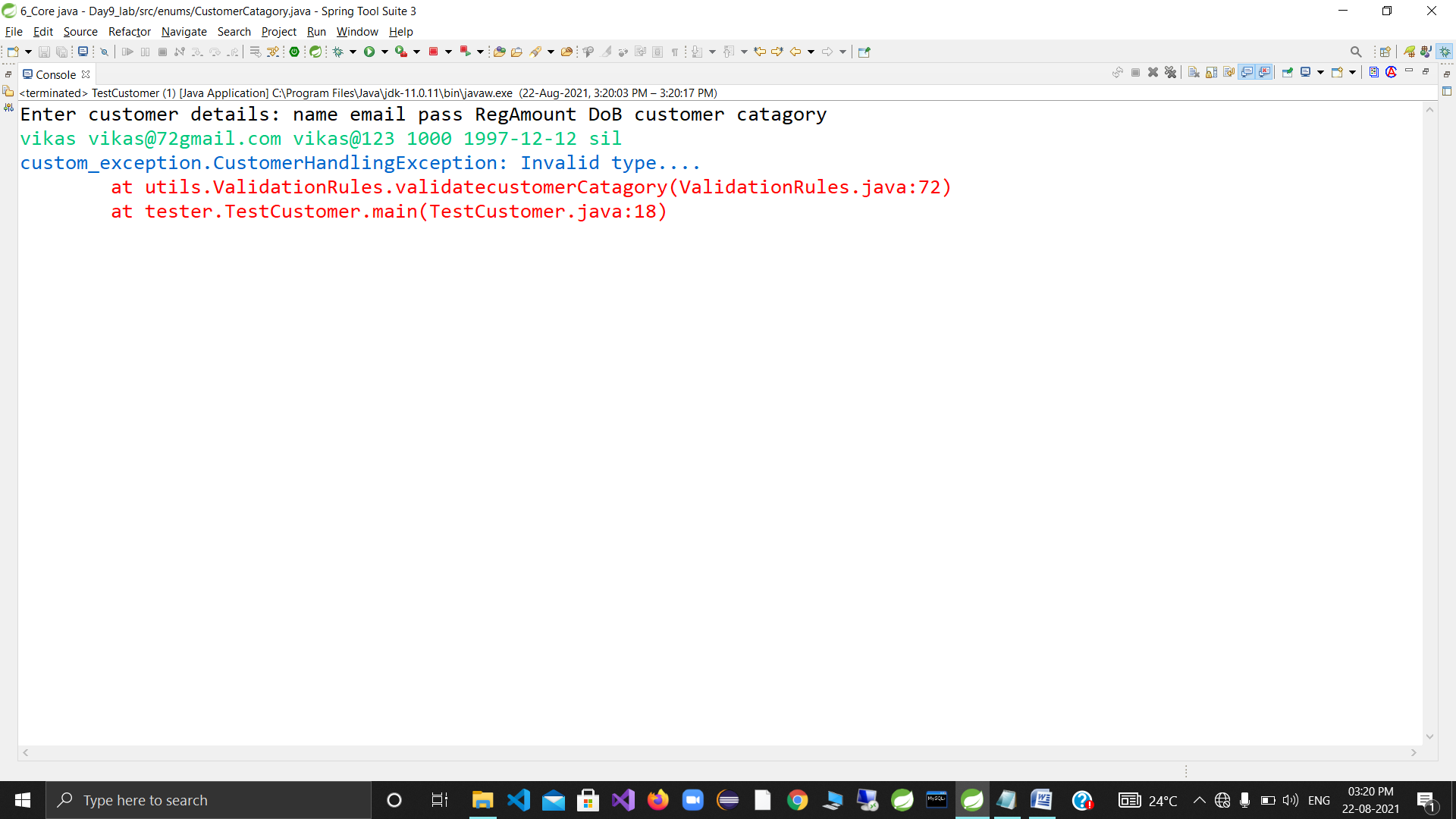
//

//Customer c=checkForEmail(sc.next(),sc.next(),sc.next(),sc.nextDouble(),sc.next());

//System.out.println("customer are fullfill profile data requariment");

//System.out.println(c)





DAY 10

5.1 Add HAS-A relationship in form of composition , between Customer & AdharCard

(Customer HAS-A AdharCard)

AdharCard details : uid (adhar card no : String) , createdOn : Date

User SHOULD NOT be able to create AdharCard instance w/o enclosing instance of Customer.

HOW ????????????????

5.2 Make another copy in Tester2

Accept basic customer details as earlier.

Iff registration succeeds : Prompt for Adhar card details .

Link it to the customer details

5.3 Create a new Tester : Tester3

Create suitable array of customers

Options :

1. Register Customer

Accept n validate all details . Iff validations succed , store the details

(w/o adhar card)

2. Link Adhar Card

I/P : email

If customer is found , accept adhar card details & link adhar card details to the customer

3. Activate Service (eg AWS or any cloud based computing offers services)

I/P : email

If customer is found , fetch it's details.

If customer has linked adhar card , display services available to the customer, otherwise throw exception.

*SOULUTION*

package com.app.core;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Customer

{

private String name;

private String email;

private String password;

private double regAmount;

private Date dob;

//has-a reln

private CustomerCategory category;

//customer has adharcard

private AdharCard acard;

public static SimpleDateFormat sdf;

static

{

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

}

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

super();

this.name = name;

this.email = email;

this.password = password;

this.regAmount = regAmount;

this.dob = dob;

this.category=category;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder("");

if(acard==null)

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

else

sb.append(acard);

return "Customer [name=" + name + ", email=" + email + ", regAmount=" + regAmount + ", dob=" + sdf.format(dob) + "category=" + category+" "+ sb ;

}

public Customer(String email) {

super();

this.email = email;

}

@Override

public boolean equals(Object o) {

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

if(o instanceof Customer)

{

Customer c =(Customer)o;

return this.email.equals(c.email);

}

return super.equals(o);

}

//add nonstatic method to link adhar details to "this" customer

public void linkAdharCard(String uid , String CreationDate) throws ParseException

{

this.acard = new AdharCard(uid,sdf.parse(CreationDate));

}

//nonstatic nested class inner class : to represent adhar card details

class AdharCard

{

private String uid;

private Date createdOn;

public AdharCard(String uid, Date createdOn) {

super();

this.uid = uid;

this.createdOn = createdOn;

}

@Override

public String toString()

{

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

}

}

}**package** com.app.core;

**import** java.util.Arrays;

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

***SILVER***(500,**new** String[] {"platform"}),

***GOLD***(1000,**new** String[] {"platform,web server"}),

***DIAMOND***(2000,**new** String[] {"platform,web server,appdeployment"}),

***PLATINUM***(3000,**new** String[] {"platform,web server,appdeployment,docker"});

//additional data member

**private** **double** charges;

**private** String[]services;

//add a parameterized constr

**private** CustomerCategory(**double** charges, String[]services)

{

//super(name,ordinal)

**this**.charges=charges;

**this**.services=services;

}

//for later validations,supply getters

**public** **double** getCharges() {

**return** charges;

}

**public** String[] getServices() {

**return** services;

}

@Override

**public** String toString()

{

**return** name()+"charges:" +charges+"services"+ Arrays.*toString*(services);

}

}

package utils;

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

import java.text.ParseException;

import java.util.Arrays;

import java.util.Date;

import com.app.core.Customer;

import com.app.core.CustomerCategory;

import custom\_exception.CustomerHandlingException;

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

public class ValidationRules {

public static final int MIN\_LENGTH;

public static final int MAX\_LENGTH;

public static Date thresholdDate;

static {

MIN\_LENGTH = 4;

MAX\_LENGTH = 10;

try {

thresholdDate = sdf.parse("1-1-1995");

} catch (ParseException e) {

System.out.println("err in sib" + e);

// e1.printStackTrace();

}

}

// validate email

public static String validateEmail(String email) throws CustomerHandlingException {

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

return email;// return validated email-id

// throw throwable instance

throw new CustomerHandlingException("Invalid email format");

}

// validate Password

public static String validatePassword(String password) throws CustomerHandlingException {

if (password.length() < MIN\_LENGTH || password.length() < MAX\_LENGTH)

return password;// return validated email-id

throw new CustomerHandlingException("Invalid Password format");

}

// static method for validate reg amt

// public static double validateRegAmount(double amount)throws CustomerHandlingException

// {

// if(amount % 500 !=0)

// throw new CustomerHandlingException("Invalid amount");

// return amount;

// }

// add static method to convert string to date

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

Date d1 = sdf.parse(dob); // conversion

if (d1.after(thresholdDate))

throw new CustomerHandlingException("Invalid date!!!!!!!!");

return d1;

}

// static method to validate catgry along with amount

public static CustomerCategory validateCategory(double amount, String category) throws CustomerHandlingException {

try {

// convert category(str - to enum)

CustomerCategory chosenCategory = valueOf(category.toUpperCase());

// CustomerCategory valueof(string name)throws illegal exception

if (amount != chosenCategory.getCharges())

throw new CustomerHandlingException(

"Invalid amount chosen for plan, pay " + chosenCategory.getCharges());

return chosenCategory;

} catch (IllegalArgumentException e) {

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

sb.append("valid category:\n ");

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

// System.out.println(sb);

throw new CustomerHandlingException(sb.toString());

}

}

// add static method to find cust details by email

// add a static method to find customer details by email

public static Customer findCustomerByEmail(String email, Customer[] custs) throws CustomerHandlingException {

// done by Customer's equals

// create customer instance by wrapping email (unique id)

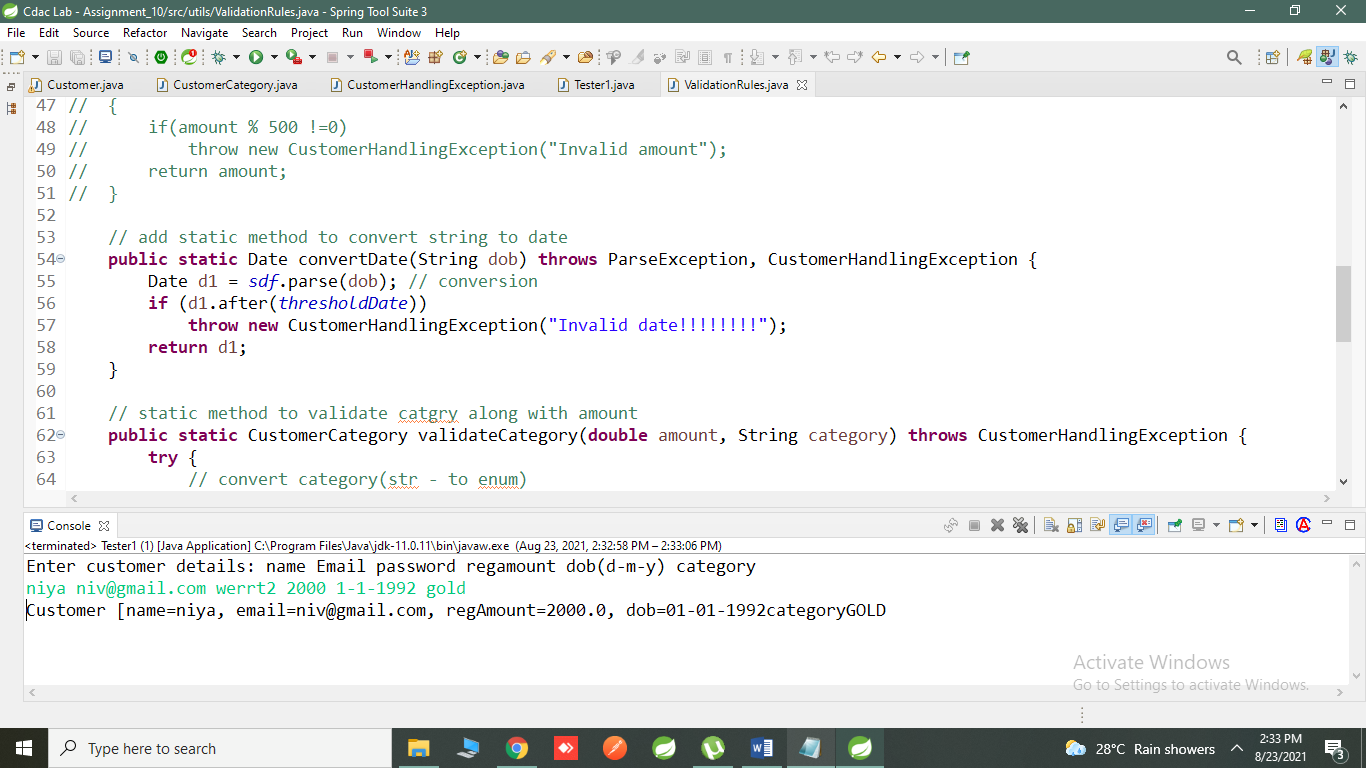
Customer c1 = new Customer(email);

for (Customer c : custs)

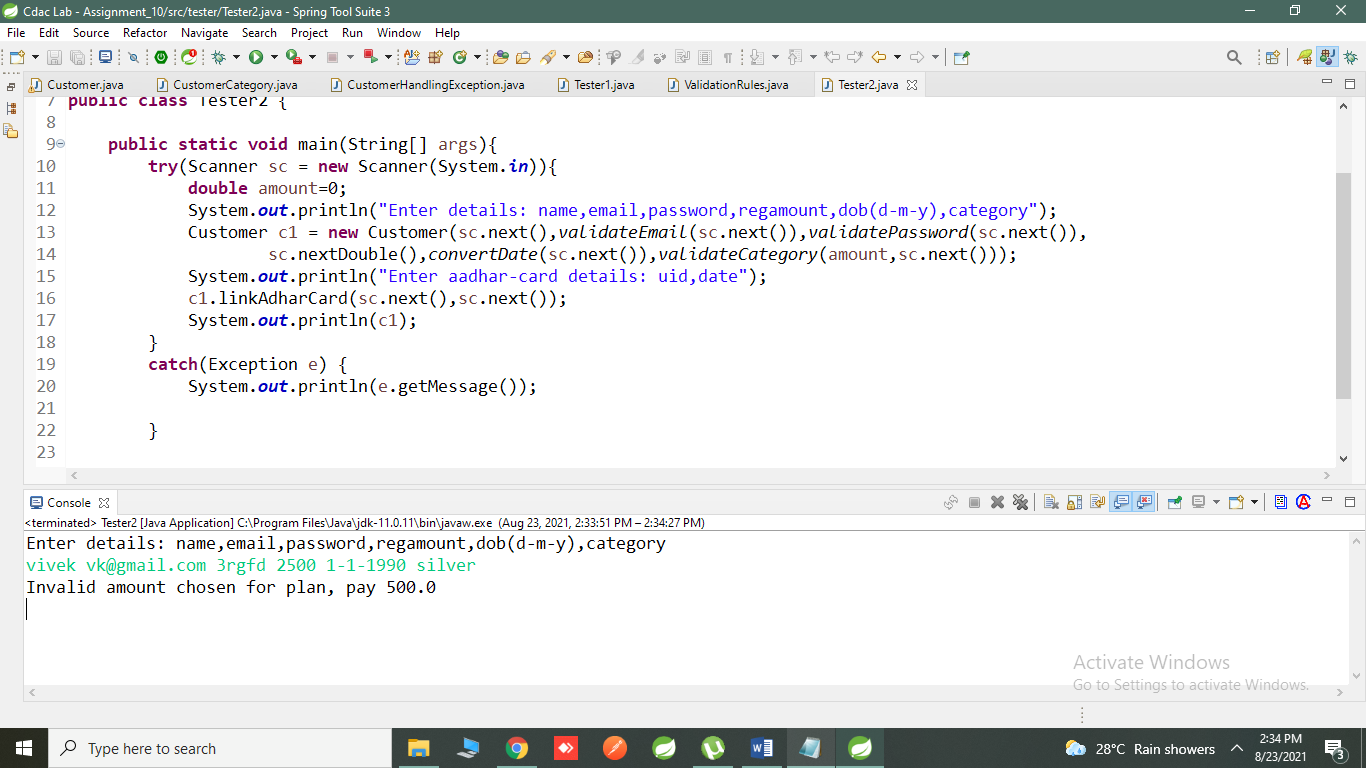
if (c != null && c.equals(c1))

return c;

throw new CustomerHandlingException("Customer not found : invalid email id!!!!!!");



}



package tester;

import static utils.ValidationRules.\*;

import java.util.Scanner;

import com.app.core.Customer;

public class Tester2 {

public static void main(String[] args){

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

double amount=0;

System.out.println("Enter details: name,email,password,regamount,dob(d-m-y),category");

Customer c1 = new Customer(sc.next(),validateEmail(sc.next()),validatePassword(sc.next()),

sc.nextDouble(),convertDate(sc.next()),validateCategory(amount,sc.next()));

System.out.println("Enter aadhar-card details: uid,date");

c1.linkAdharCard(sc.next(),sc.next());

System.out.println(c1);

}

catch(Exception e) {

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

}

}

}

package tester;

import static utils.ValidationRules.convertDate;

import static utils.ValidationRules.validateCategory;

import static utils.ValidationRules.validateEmail;

import static utils.ValidationRules.validatePassword;

import static utils.ValidationRules.findCustomerByEmail;

import java.util.Scanner;

import com.app.core.Customer;

public class Tester3 {

public static void main(String[] args)

{

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

{

Customer[] customers = new Customer[100];

boolean exit= false;

int counter =0;

while(!exit)

{

System.out.println("1.regiter customer 2.link adhar card 3.Display services 4.change plan 5.display all 10.exit");

try {

switch (sc.nextInt()) {

case 1:

double amount =0;

System.out.println("Enter details: name,email,password,regamount,dob(d-m-y),category");

Customer c1 = new Customer(sc.next(),validateEmail(sc.next()),validatePassword(sc.next()),

sc.nextDouble(),convertDate(sc.next()),validateCategory(amount,sc.next()));

customers[counter++]=c1;

break;

case 2:

System.out.println("Enter email to link aadhar card:");

//invoke method to check if customer with given email exist?

c1= findCustomerByEmail(sc.next(),customers);

System.out.println("Enter adhar no and date:");

c1.linkAdharCard(sc.next(), sc.next());

break;

case 5:

for(Customer c: customers)

if(c!=null)

System.out.println(c);

case 10:

exit=true;

break;

default:

break;

}

}catch (Exception e) {

e.printStackTrace();

}

//rid off data from scanner till endofline

sc.nextLine();

}

}

}

}

DAY 11

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

*SOLUTION*

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);

}

}

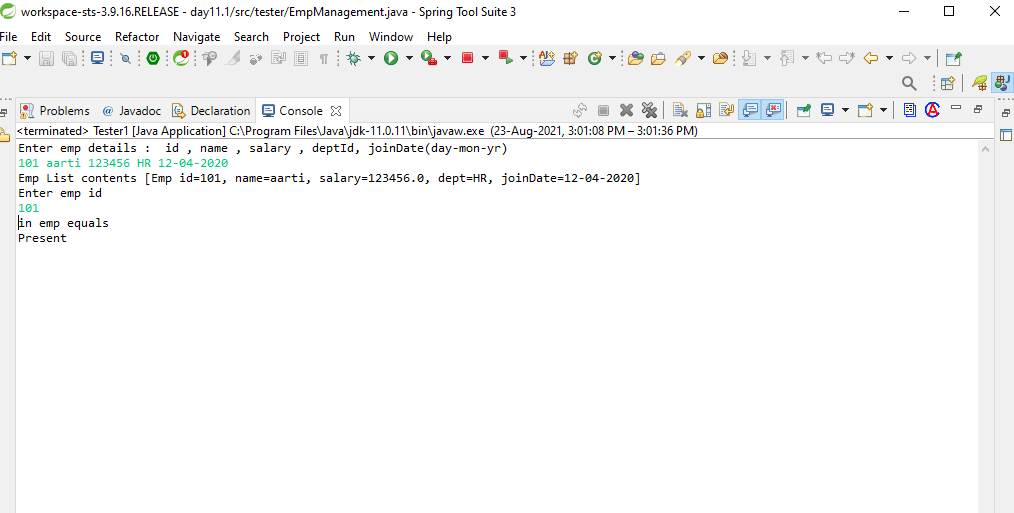
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EmpManagement.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**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) {

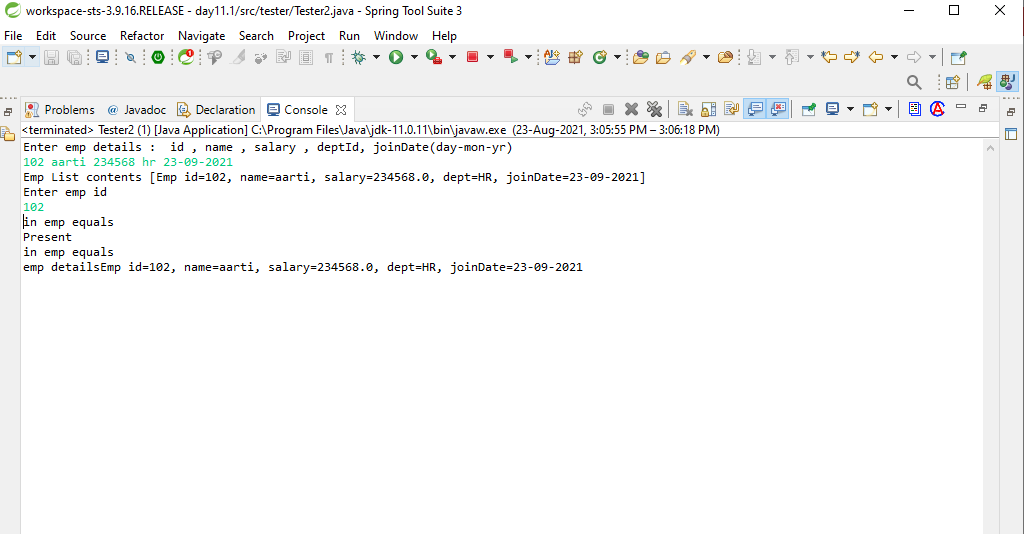
e.printStackTrace();

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*output\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



Day 12:

Create java application for library of books.

Book : isbn or title(string) , author(string) , price(double), category(enum) , quantity(int) , publish date(Date)

No validation rules for today.

Custom exception class : BookHandling exception.

Create CollectionUtils to add at least 5 book details.(sample data)

Create a Tester for the following

Get populated book list from the CollectionUtils

Options

0. Display all available books using Iterator

1. 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

2. Return book

i/p isbn & quantity

Update the quantity if book already exists

otherwise throw exception

3. Apply discount on old books (i.e reduce price of books published before specific date)

i/p date & discount amount

4. Sort the books as per isbn.

Day 13:

6. Complete Day 12 assignment & add this option

Option 5 : Sort the books as per publish date & author , using custom ordering + ano inner class

*DAY 12&13 SOLUTION:*

1.BOOKS.JAVA

package com.app.core;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Books implements Comparable<Books>{

private int isbn;

private String title;

private String author;

private double price;

private Category category;

private int quantity;

private Date publishDate;

public static SimpleDateFormat sdf;

static {

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

}

public Books(int isbn, String title, String author, double price, Category category, int quantity,

Date publishDate) {

super();

this.isbn = isbn;

this.title = title;

this.author = author;

this.price = price;

this.category = category;

this.quantity = quantity;

this.publishDate = publishDate;

}

public Books(int isbn) {

super();

this.isbn = isbn;

}

public String toString()

{

return "Book isbn: "+isbn+"| title: "+title+"| author: "+author+

"| price: "+price+"| category: "+category+"| quantity: "+quantity+

"| publish date: "+sdf.format(publishDate)+"|";

}

public boolean equals(Object obj)

{

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

if(obj instanceof Books)

return this.isbn==((Books)obj).isbn;

return false;

}

public int compareTo(Books anotherBook)

{

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

//sorting criteria : book isbn

if(this.isbn < anotherBook.getIsbn())

return -1;

if(this.isbn == anotherBook.getIsbn())

return 0;

return 1;

}

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;

}

public void setIsbn(int isbn) {

this.isbn = isbn;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public Date getPublishDate() {

return publishDate;

}

public String getTitle() {

return title;

}

public int getIsbn() {

return isbn;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public static SimpleDateFormat getSdf() {

return sdf;

}

}

2.CATEGORY.JAVA

**package** com.app.core;

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

***SCIENCE*** , ***COMICS*** , ***LANGUAGES*** , ***SPORTS***

}

3.BOOKEXCEPTION HANDLING:

**package** exception\_handling;

@SuppressWarnings("serial")

**public** **class** BookExceptionHandling **extends** Exception{

**public** BookExceptionHandling ( String msg)

{

**super**(msg);

}

}

4.COLLECTIONUTILS.JAVA

package utils;

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

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

import java.text.ParseException;

import java.util.ArrayList;

import com.app.core.Books;

public class CollectionUtils {

public static ArrayList<Books> populateBooksdata() throws ParseException

{

ArrayList<Books> list = new ArrayList<>();

list.add(new Books(1010,"Physics","Newton",200,SCIENCE,15,sdf.parse("10-10-2010")));

list.add(new Books(1005,"Chemistry","Bernolis",300,SCIENCE,10,sdf.parse("10-12-2012")));

list.add(new Books(1001,"Cricket","Sachin",500,SPORTS,25,sdf.parse("10-12-2012")));

list.add(new Books(1003,"English","Barrens",400,LANGUAGES,20,sdf.parse("10-10-2010")));

list.add(new Books(1007,"Marvel","Stark",600,COMICS,5,sdf.parse("10-10-2019")));

return list;

}

}

5.BOOKMANAGEMENT.JAVA

package tester;

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

import static utils.CollectionUtils.populateBooksdata;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.Date;

import java.util.Scanner;

import com.app.core.Books;

import exception\_handling.BookExceptionHandling;

public class BookManagement {

public static void main(String[] args) {

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

// create suitable D.S

ArrayList<Books> book = populateBooksdata();

boolean exit = false;

while (!exit) {

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

System.out.println("Options : 1.Display All Available Books Details 2.Issue Out Books 3. Return Books"

+ " 4. Discounted Books 5. Sort All Books By ISBN "

+ "6. Sort All Books By Publish Date And Author 100. Exit");

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

try {

switch (sc.nextInt()) {

case 1:

System.out.println("Book Details: ");

for(Books b : book)

{

System.out.println(b);

}

break;

case 2:

System.out.println("Enter Book isbn & Quantity which you want to issue..??");

int in = sc.nextInt();

int quant = sc.nextInt();

int index = book.indexOf(new Books(in));

if (index == -1)

throw new BookExceptionHandling("Invalid Book isbn : Book not found!!!!!");

Books bk = book.get(index);

if(bk.getQuantity()<quant)

{

System.out.println("Not in stock..!!!");

}

else

{

bk.setQuantity(bk.getQuantity()-quant);

System.out.println(bk.getTitle()+" Book issued "+"Now Quantity is "+bk.getQuantity());

}

break;

case 3:

System.out.println("Enter Book isbn & Quantity which you want to return..??");

int id = sc.nextInt();

int quants = sc.nextInt();

index = book.indexOf(new Books(id));

if (index == -1)

throw new BookExceptionHandling("Invalid Book isbn : Book not found!!!!!");

Books bks = book.get(index);

bks.setQuantity(bks.getQuantity()+quants);

System.out.println(bks.getTitle()+" Book return "+" Now Quantity is "+bks.getQuantity());

break;

case 4:

System.out.println("Enter Book Publish Date & Discount ..??");

Date dt = sdf.parse(sc.next());

double disc = sc.nextDouble();

for(Books b:book)

{

if(b.getPublishDate().before(dt))

b.setPrice(b.getPrice()-disc);

System.out.println(b);

}

break;

case 5:

Collections.sort(book);

System.out.println("sorted list as per isbn of Books ..");

for(Books b : book)

System.out.println(b);

break;

case 6:

Collections.sort(book, new Comparator<Books>() {

@Override

public int compare(Books b1, Books b2) {

int res = b1.getPublishDate().compareTo(b2.getPublishDate());

if(res == 0)

{

return b1.getAuthor().compareTo(b2.getAuthor());

}

return res;

}

});

System.out.println("Sorted list as per publish date & author ...");

for(Books b : book)

{

System.out.println(b);

}

break;

case 100:

exit = true;

break;

}

}

catch (Exception e) {

// e.printStackTrace();

System.out.println(e);

}

// clear off pending inputs from scanner

sc.nextLine();

}

} catch (Exception e) {

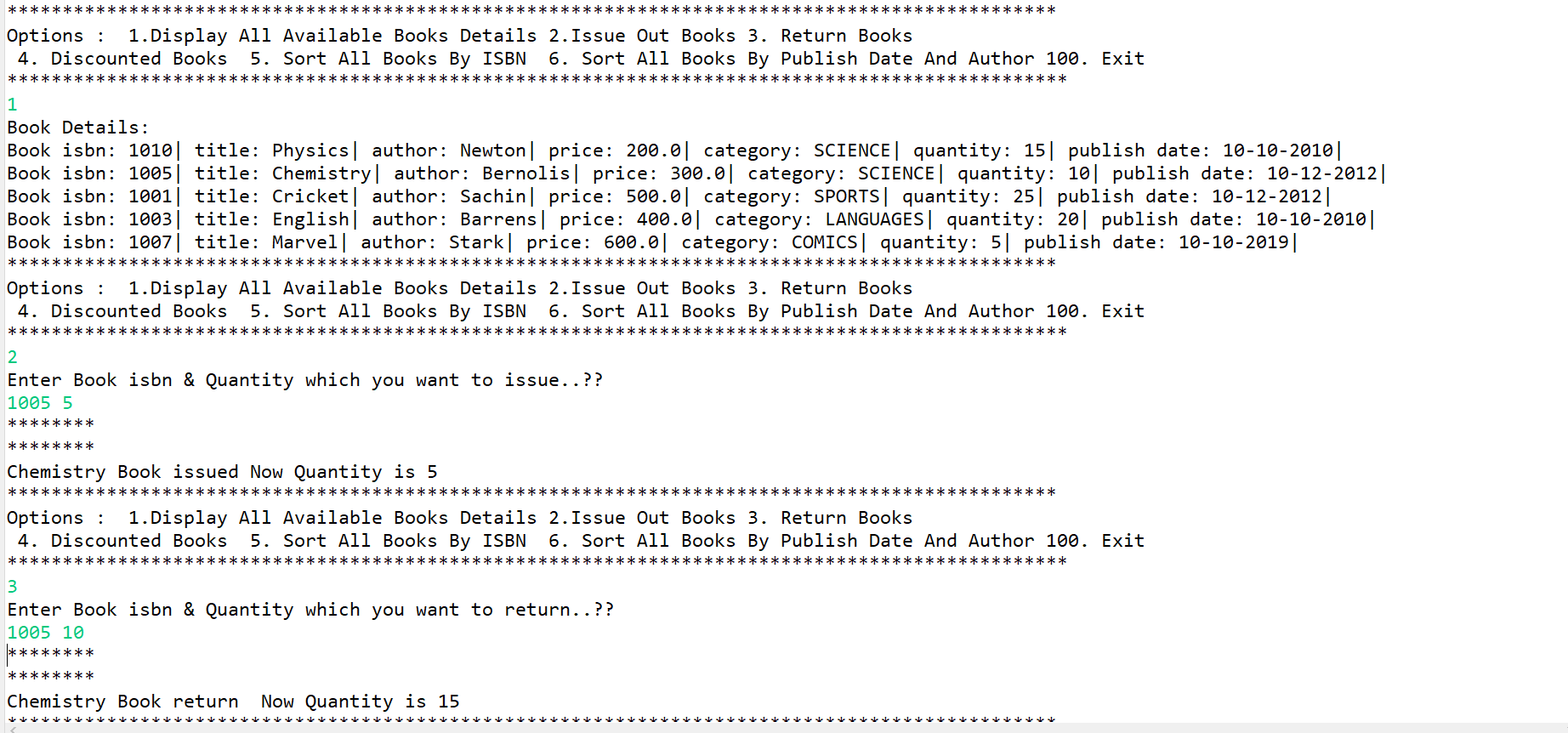
e.printStackTrace();

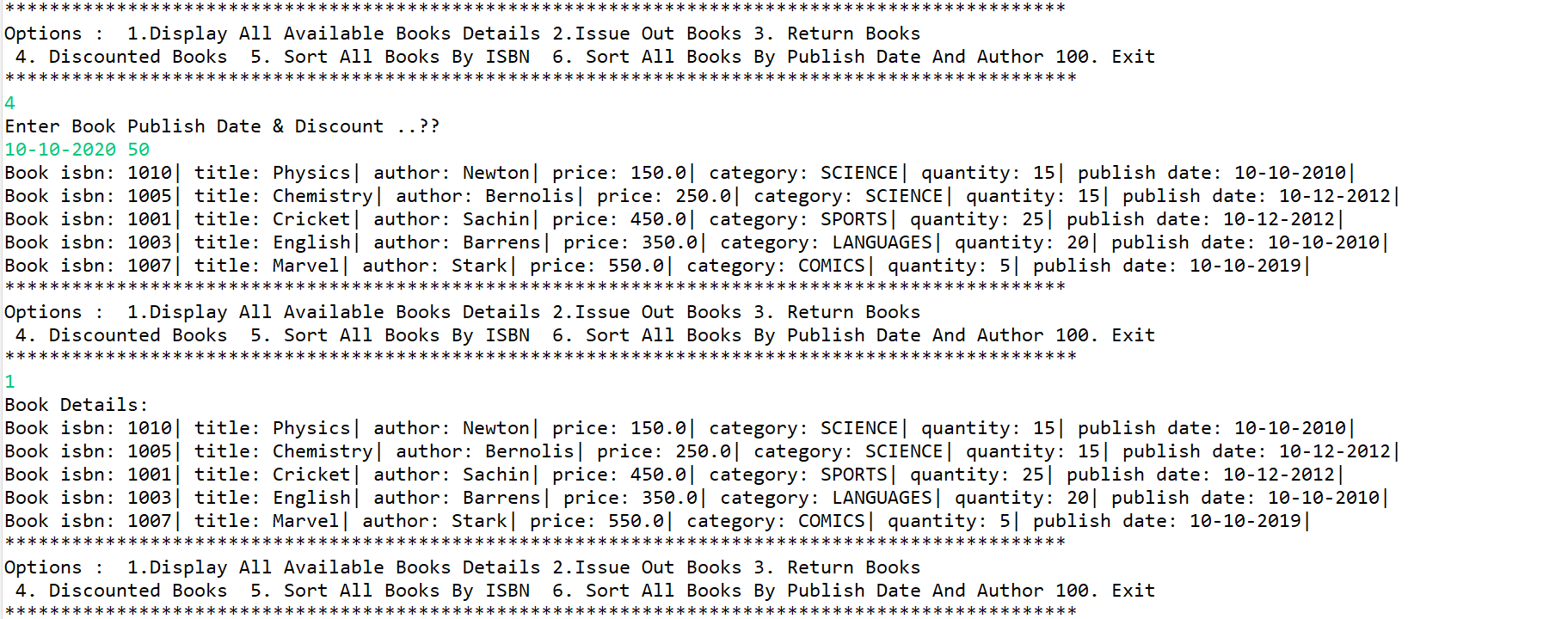
}

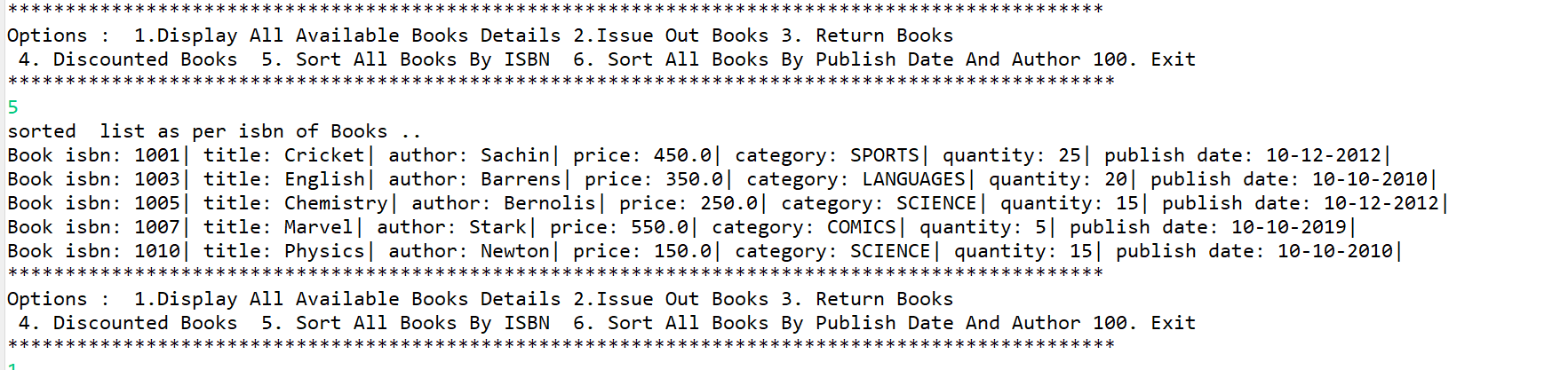
}

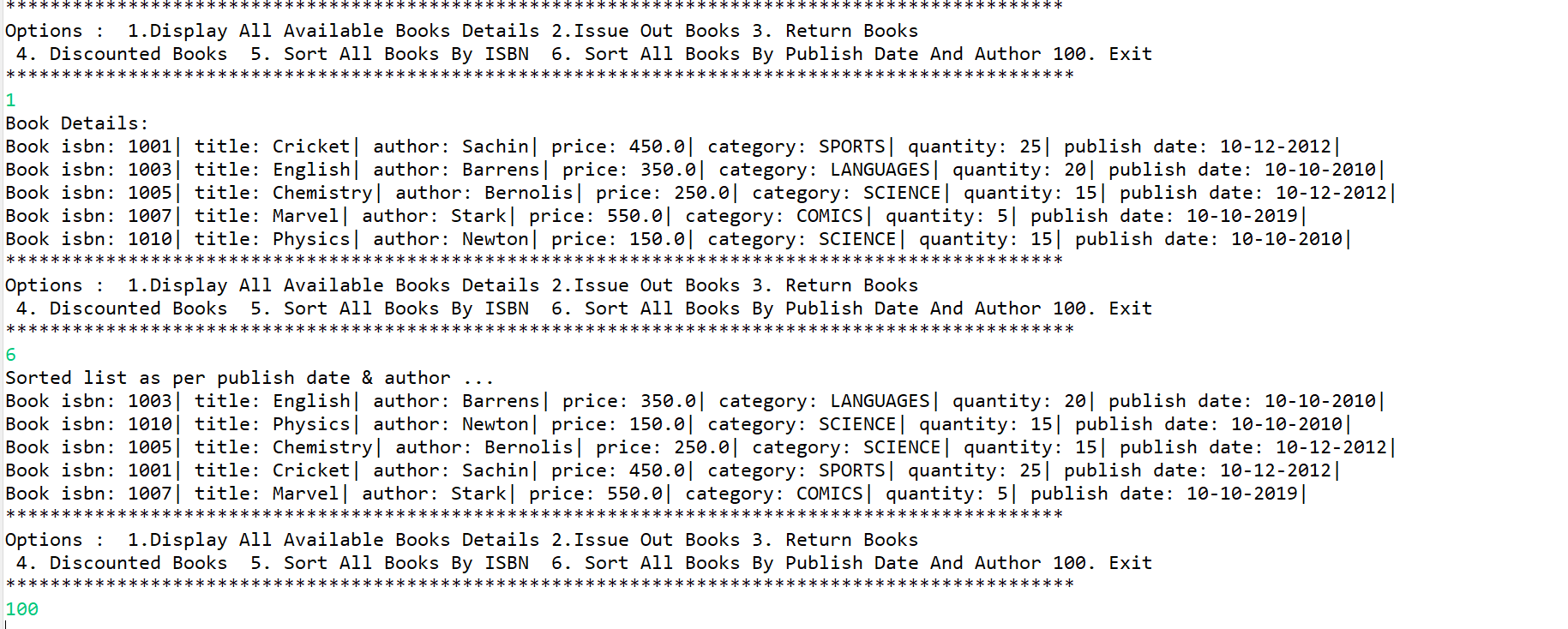
}

OUTPUT:









DAY 14:

Solve day12 n day13 assignment options, by replacing ArrayList by Map.

Replace Date by LocalDate

1.BOOKS.JAVA

**package** com.app.core;

**import** java.time.LocalDate;

//Book : isbn or title(string) , author(string) , price(double), category(enum) , quantity(int) , publish date(LocalDate)

**public** **class** Book {

**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.BOOKEXCEPTION HANDLING:

**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("1999-11-01")));

map.put("book8", new Book("book8", "Ramesh",510 ,SCIENCE, 30,parse("2020-01-01")));

return map;

}

}

5.BOOKMANAGEMENT.JAVA

package tester;

import static java.time.LocalDate.parse;

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;

public class BookLibrary {

public static void main(String[] args) {

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

// get lib books

Map<String, Book> books = populateBooks();

boolean exit = false;

while (!exit) {

System.out.println("1. Display all books 2.Issue out Books 3.Return Book 4.Discount on books before date 5.Sorted list as per isbn 6.sort by publish date 100.Exit");

try {

switch (sc.nextInt()) {

case 1:

System.out.println("library books: ");

// for(Book b:books.values())

// {

// System.out.println(b);

// }

books.forEach((isbn,book)->System.out.println(book));

//lambda func.

break;

case 2:

System.out.println("Enter isbn no and quantity : ");

String isbn = sc.next();

int quant = sc.nextInt();

Book book = books.get(isbn);

if(book == null)

throw new BookHandlingException("Book isbn "+isbn+" not found");

if(book.getQuantity()>quant) {

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

System.out.println("Book with isbn"+isbn+"issued");

}

else

throw new BookHandlingException("insufficient qty for book");

break;

case 3:

System.out.println("Enter isbn and quantity to return book: ");

isbn = sc.next();

quant = sc.nextInt();

book = books.get(isbn);

if(book==null)

throw new BookHandlingException("Book with isbn "+isbn+ " not found");

book.setQuantity(book.getQuantity()+quant);

System.out.println("Book returned successfully");

break;

case 4:

System.out.println("Enter Book publish date and Discount ");

LocalDate date = parse(sc.next());

double disc = sc.nextDouble();

//

// for(Book b:books.values())

// if(b.getPublishDate().isBefore(date))

// b.setPrice(b.getPrice()-disc);

books.forEach((isbn1,bk)->{

if(bk.getPublishDate().isBefore(date))

bk.setPrice(bk.getPrice()-disc);

});

System.out.println("Applied Discount..!! ");

break;

case 5:

System.out.println("sorted list as per isbn ");

TreeMap<String, Book> sortbooks = new TreeMap<>(books);//string class compareto method

sortbooks.forEach((isbn1,bk)->System.out.println(bk));

break;

case 6:

//Collection<Book>booklist1=books.values();

ArrayList<Book>list=new ArrayList<Book>(books.values());

//

//

//

// Collections.sort(list, new Comparator<Book>() {

//

// @Override

// public int compare(Book b1, Book b2) {

//

// int res = b1.getPublishDate().compareTo(b2.getPublishDate());

// if(res == 0)

// {

// return b1.getAuthor().compareTo(b2.getAuthor());

// }

// return res;

// }

// });

//function literals

Comparator<Book> bookComparator = (b1,b2)->{

int res = b1.getPublishDate().compareTo(b2.getPublishDate());

if(res == 0)

return b1.getAuthor().compareTo(b2.getAuthor());

return res;

};

Collections.sort(list,bookComparator);

System.out.println("sorted list ");

list.forEach(b->System.out.println(b));

break;

case 100:

exit = true;

break;

}

} catch (Exception e) {

e.printStackTrace();

}

sc.nextLine();

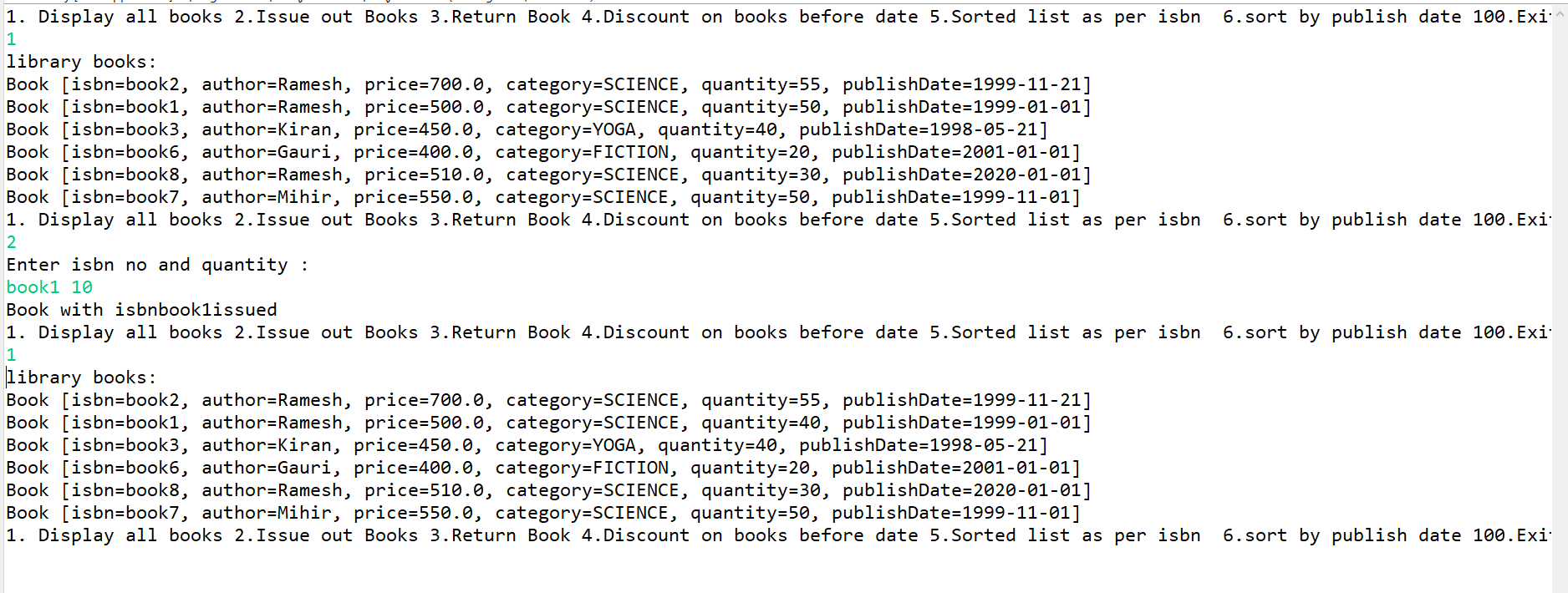
}

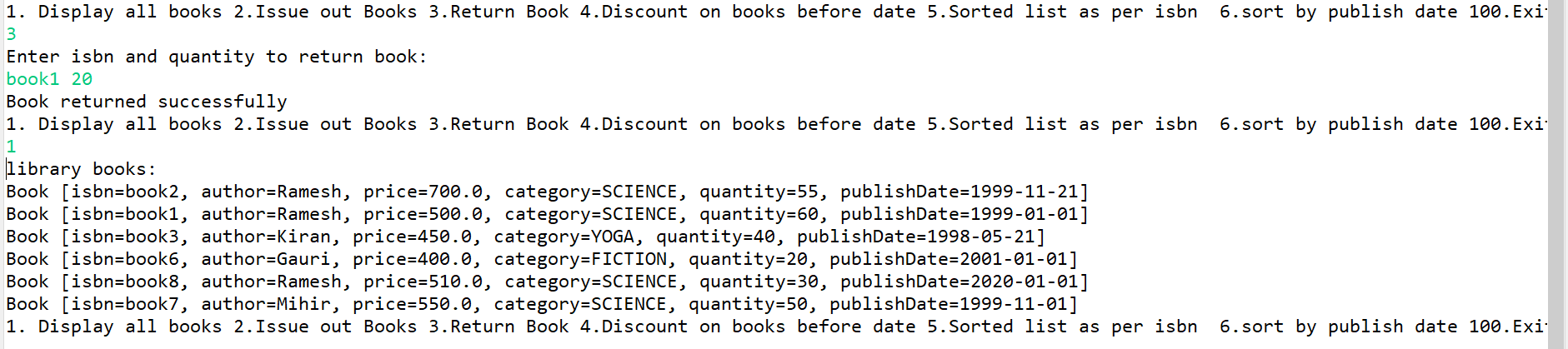
}

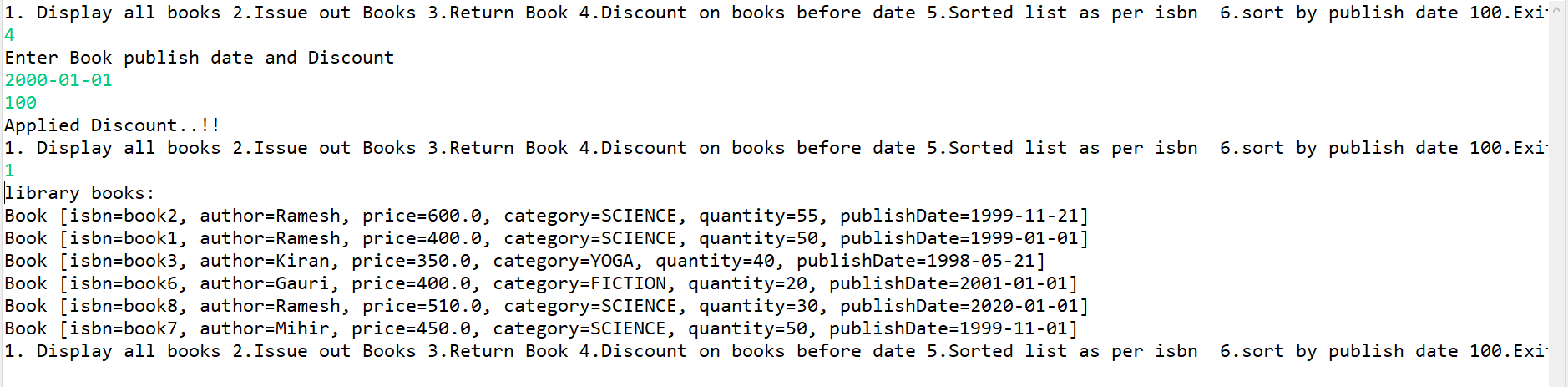
}

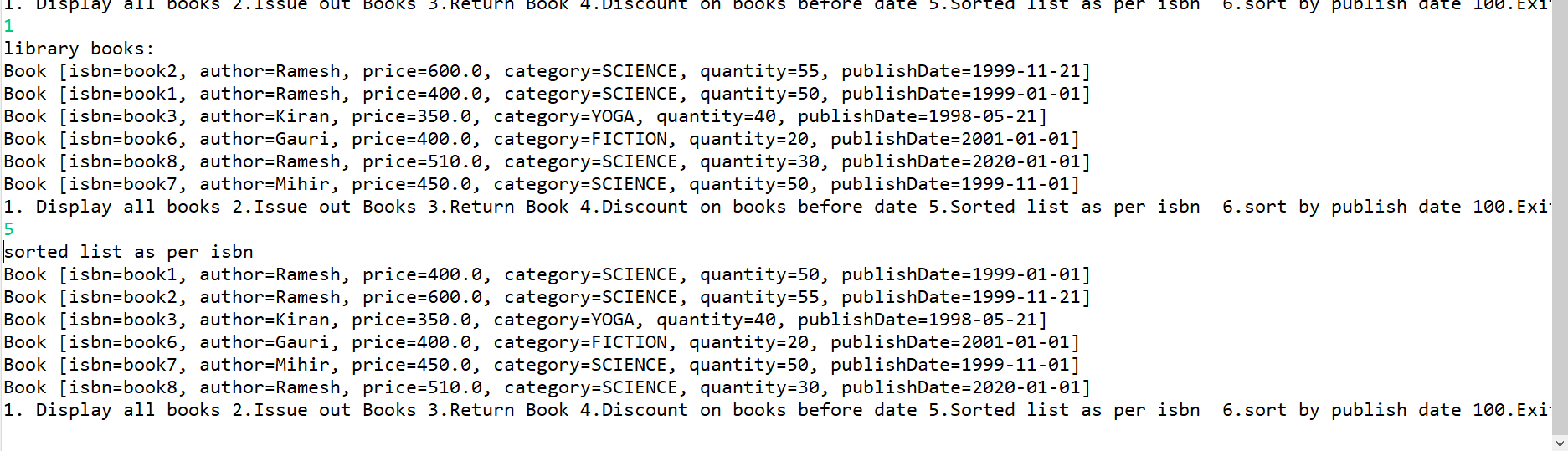
}

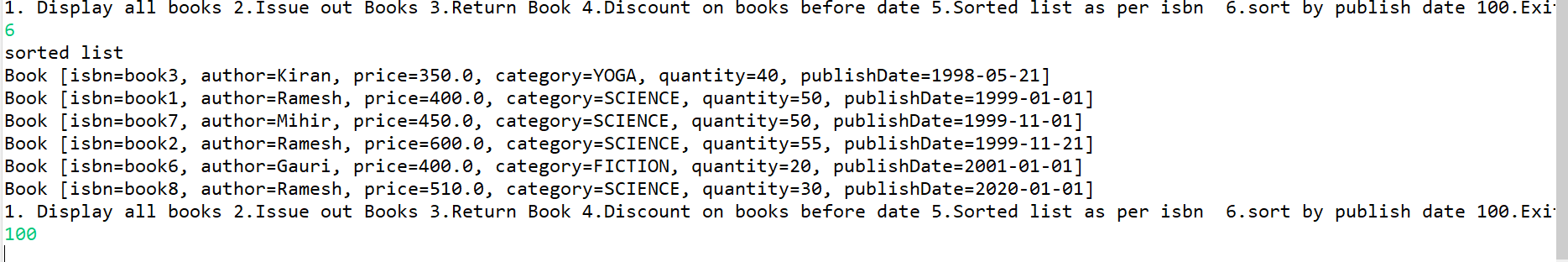
OUTPUT:











Day 15:

No new assignments !

Day 16:

Solve this

8.1 Create Student class -- rollNo(string),name,dob(LocalDate),subject(enum),gpa(double)

Create Subject enum --JAVA,DBT,ANGULAR,REACT,SE

Add constr & to string & getters.

8.2 Create CollectionUtils ---to ret populated FIXED size list of students (4 records)

*SOLUTION:*

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;

}

}

SUBJECT.JAVA

**package** com.app.core;

**public** **enum** Subject {

***JAVA***, ***DBT***, ***ANGULAR***, ***REACT***, ***SE***, ***DEVOPS***,***MERN***

}

3.COLLECTIONUTILS:

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, 7.5));

students.add(new Student("dac-005", "Rekha", parse("1997-11-06"), JAVA, 8.6));

students.add(new Student("dac-003", "Anish", parse("1999-01-21"), SE, 9.1));

students.add(new Student("dac-004", "Kirti", parse("2000-01-01"), DEVOPS, 6.1));

students.add(new Student("dac-008", "Anshuman", parse("1998-06-07"), JAVA, 5.1));

students.add(new Student("dac-006", "Timir", parse("1999-12-21"), DBT, 5.2));

students.add(new Student("dac-007", "Tarun", parse("1997-03-21"), ANGULAR, 4.8));

students.add(new Student("dac-010", "Priti", parse("1999-01-08"), JAVA, 8.3));

return students;

}

}

4. 1. Display all student details

package tester;

import static utils.StudentCollectionUtils.populateList;

import java.util.List;

import com.app.core.Student;

public class Test1 {

public static void main(String[] args) {

// Display all student details

List<Student> students = populateList();

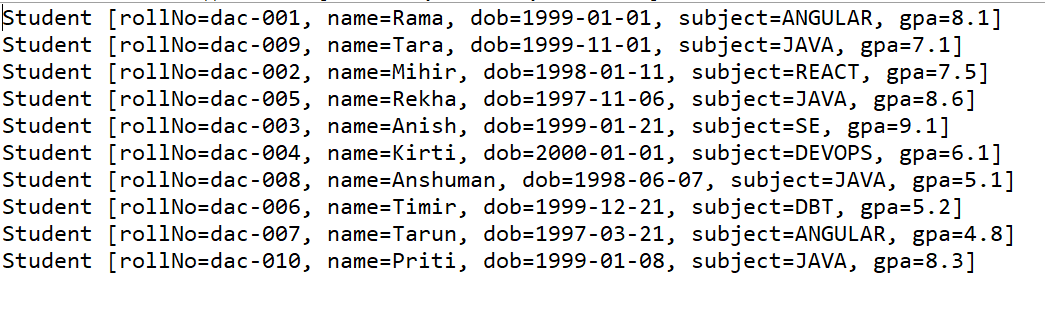
// students.forEach(student -> System.out.println(student));

students.forEach(System.out::println);

}

}

OUTPUT:



4.1.5 Display all student details sorted as per GPA

package tester;

import static utils.StudentCollectionUtils.populateList;

import java.util.Collections;

import java.util.Comparator;

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();

// Comparator<Student> comp=(s1,s2) -> ((Double)s1.getGpa()).compareTo(s2.getGpa());

// Collections.sort(students,comp);

// students.forEach(System.out::println);

//OR

// students.stream(). //Stream<Student> : un sorted

// sorted(comp). //Stream<Student> : sorted as per gpa

// forEach(System.out::println);

students.stream(). //Stream<Student> : un sorted

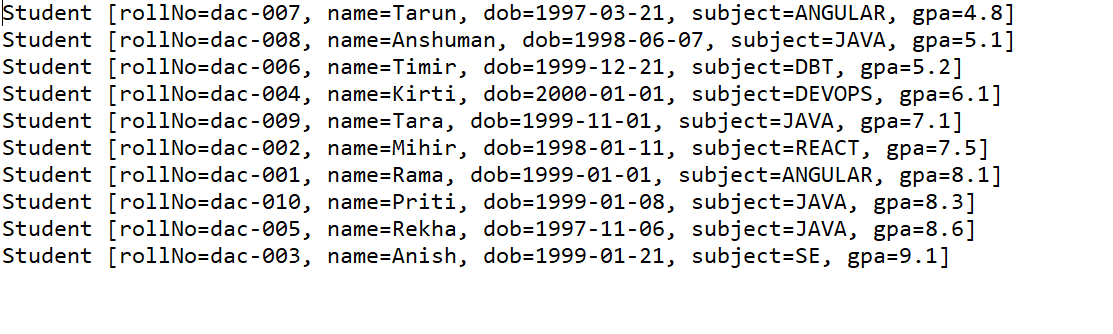
sorted(Comparator.comparing(Student::getGpa)). // s -> s.getGpa() Stream<Student> : sorted as per gpa

forEach(System.out::println);

}

}

OUTPUT:



2. Write a tester to print sum of gpa of students for the specified subject

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 Test4 {

public static void main(String[] args) {

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 sumOfGPA=students.stream().

filter(s -> s.getSubject()==subject).

mapToDouble(Student :: getGpa).

sum();

System.out.println(sumOfGPA);

} catch (Exception e) {

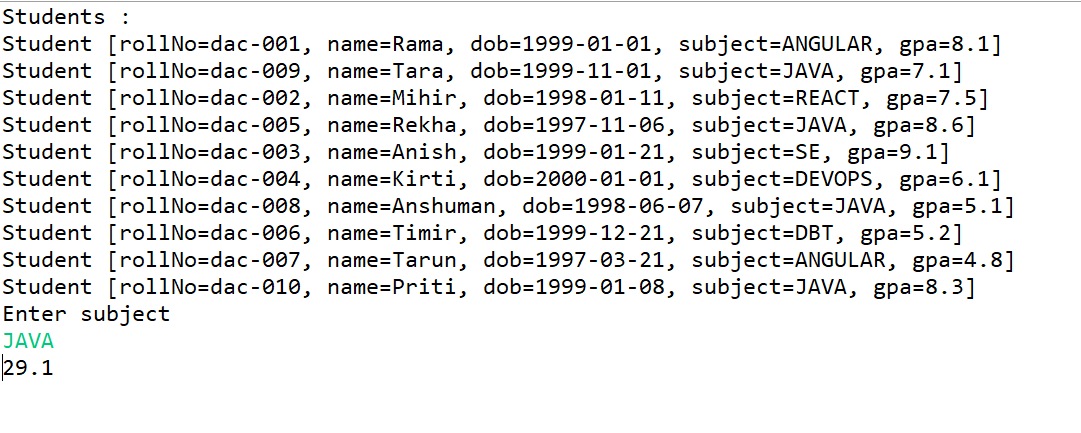
e.printStackTrace();

}

}

}

OUTPUT:



3. Write a tester to print average of gpa of students for the specified subject

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) {

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 avgGPA=students.stream().

filter(s -> s.getSubject()==subject).

mapToDouble(Student :: getGpa).

average().orElse(-1);//If DoubleStream is empty : rets -1 or rets actual avg of GPA of filtered students

System.out.println(avgGPA);

} catch (Exception e) {

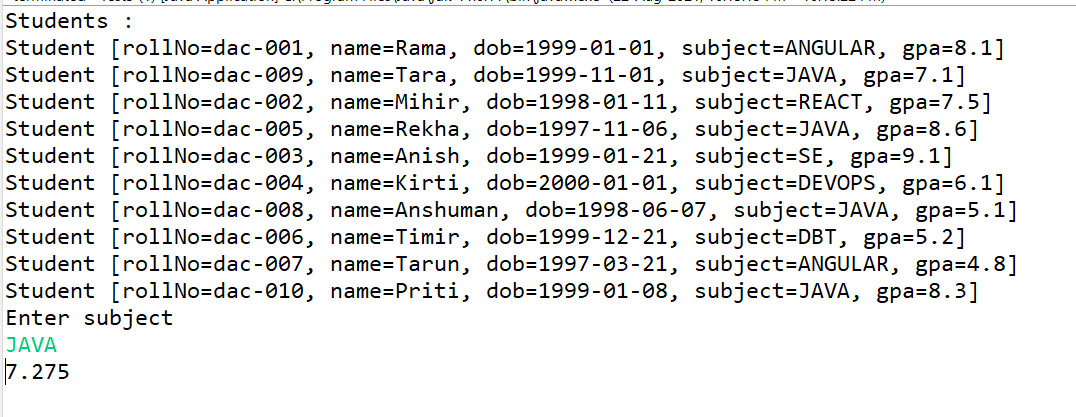
e.printStackTrace();

}

}

}

OUTPUT:



4 Print name of specified subject topper

**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) {

**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> optionalStudent=students.stream().

filter(s -> s.getSubject() == subject).

max(Comparator.*comparing*(Student::getGpa));

**if**(optionalStudent.isPresent())

System.***out***.println(optionalStudent.get().getName());

} **catch** (Exception e) {

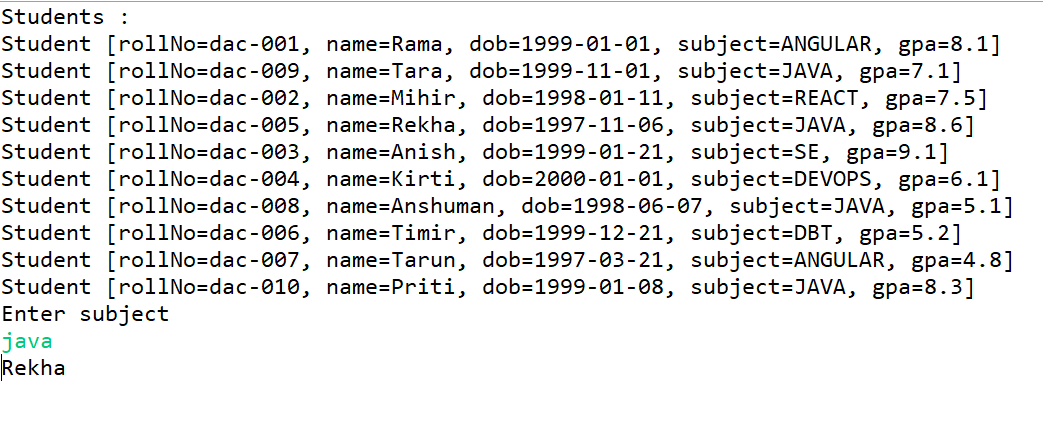
e.printStackTrace();

}

}

}

Output:



DAY 17:

1. Continue with BookLibrary management application.

Add following functionality

Whenever application is launched , in the init phase of the application , try to restore the library book details from bin file , using de-serialization. In case of any exc(file doesn't exist or reading error) , populate the Map using sample data.

Before terminating the application , save all library books in the bin file using serialization

*SOLUTION:*

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;

}

}

**package** com.app.core;

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

***TECHNOLOGY***, ***SCIENCE***, ***SELF\_HELP***, ***YOGA***, ***MUSIC***,***FICTION***

}

**package** custom\_exception;

@SuppressWarnings("serial")

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

**public** BookHandlingException(String mesg) {

**super**(mesg);

}

}

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;

}

}

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 {

//add a static method to store library books using serialization

//method : public void writeObject(Object o) throws IOExc.

static void storeBooks(Map<String,Book> books,String fileName) throws IOException

{

//Java App ---> OOS ---> FOS(bin file)

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

{

out.writeObject(books);

}

}

//add a static method to restore books : from bin file

//in case of any exc --return map populated with sample data

//in case of no err --return map populated with data from bin file.

//method : public Object readObject() throws IOExc,ClassNotFoundException

@SuppressWarnings("unchecked")

static Map<String,Book> restoreBooks(String fileName)

{

//chain : Java App <------OIS <---- FIS(bin file)

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

{

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

}catch (Exception e) {

//de -serial failed : so ret sample data map

return populateBooks();

}

}

}

package tester;

import static utils.SerIOUtils.\*;

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;

public class BookLibrary {

public static void main(String[] args) {

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

System.out.println("Enter file name to restore the books");

String fileName=sc.nextLine();

// get lib books

Map<String, Book> books = restoreBooks(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);

break;

case 100:

exit = true;

//store the books in bin file

storeBooks(books,fileName);

break;

}

} catch (Exception e) {

e.printStackTrace();

}

sc.nextLine();

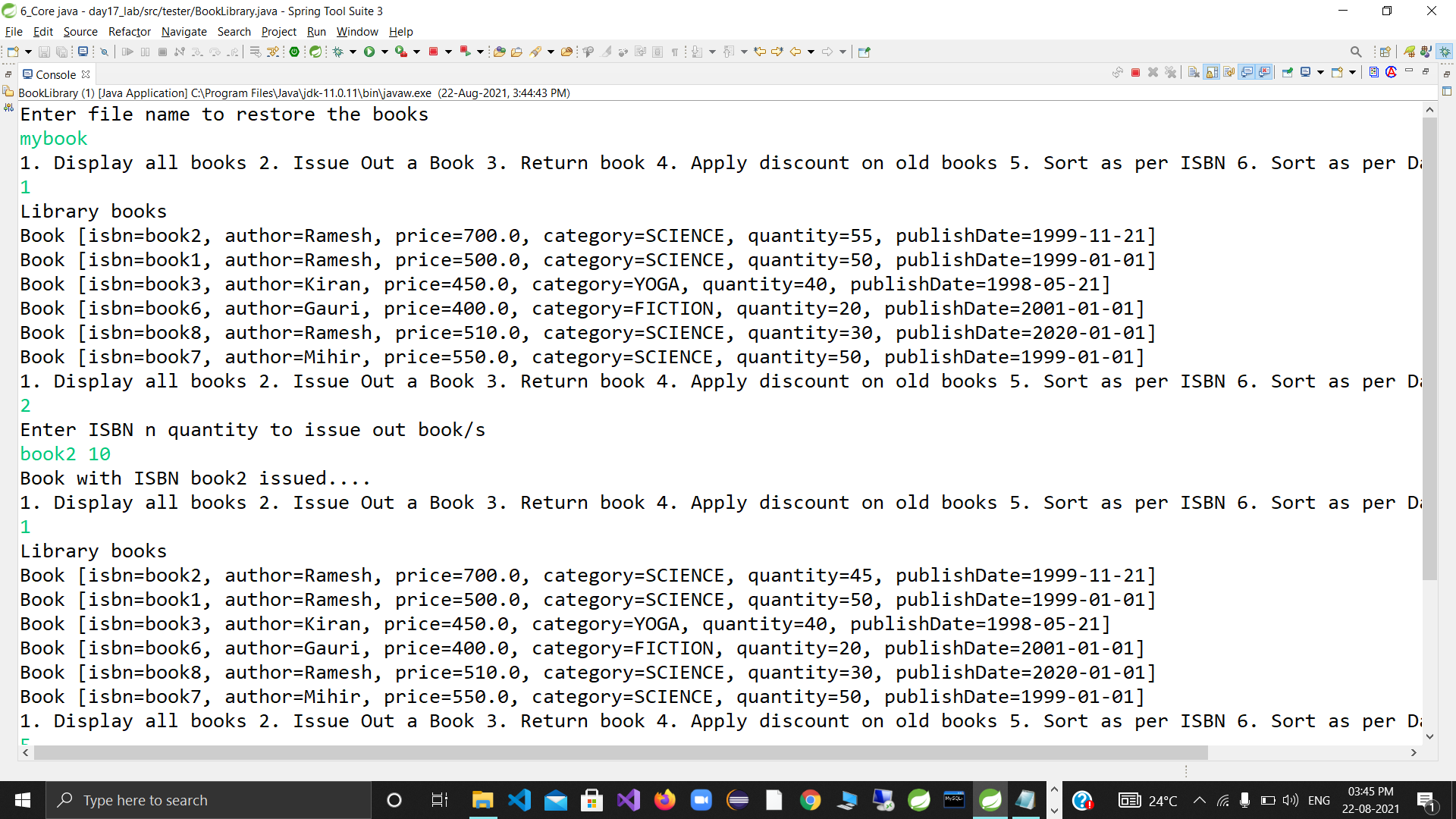
}

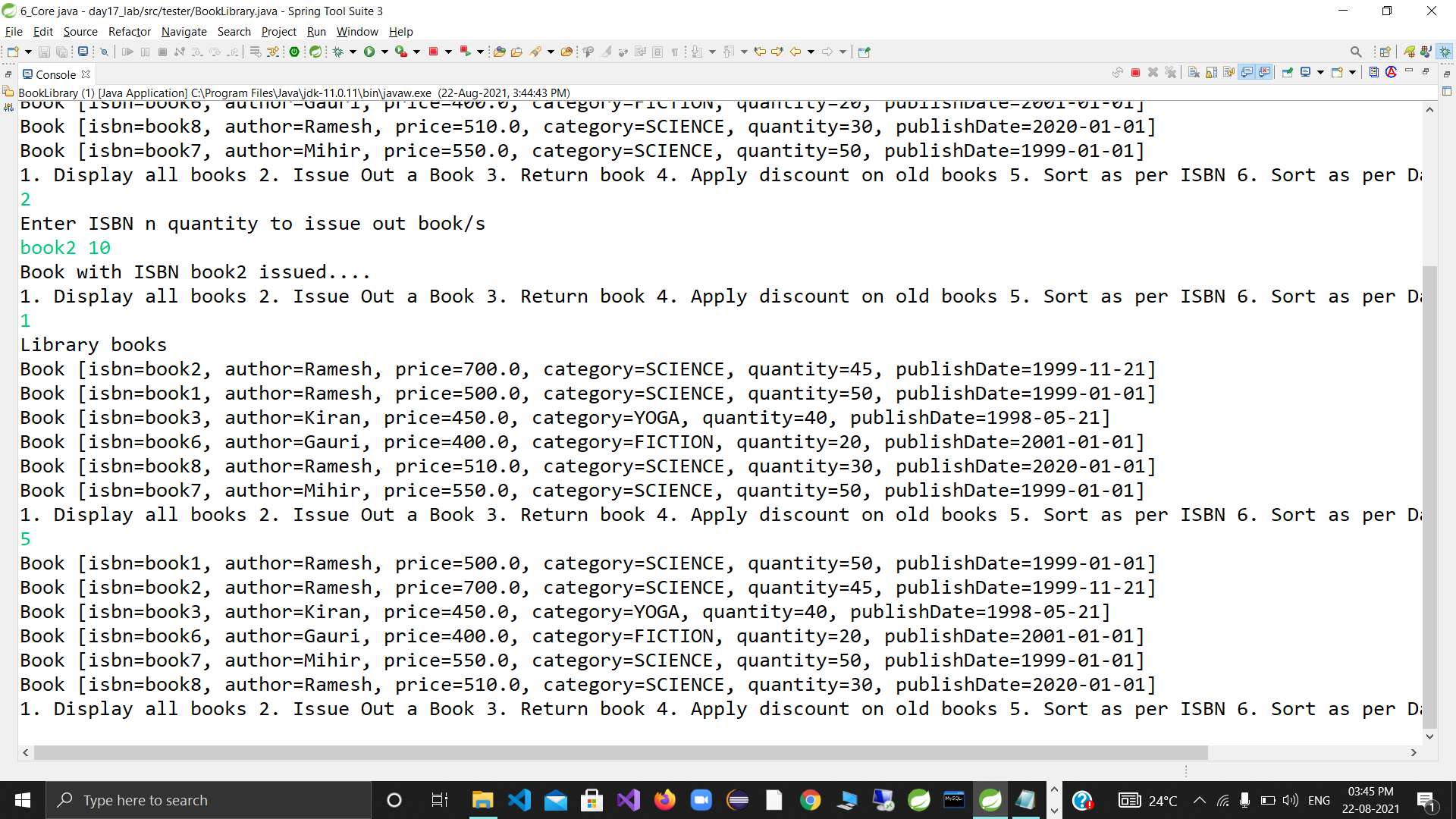
}

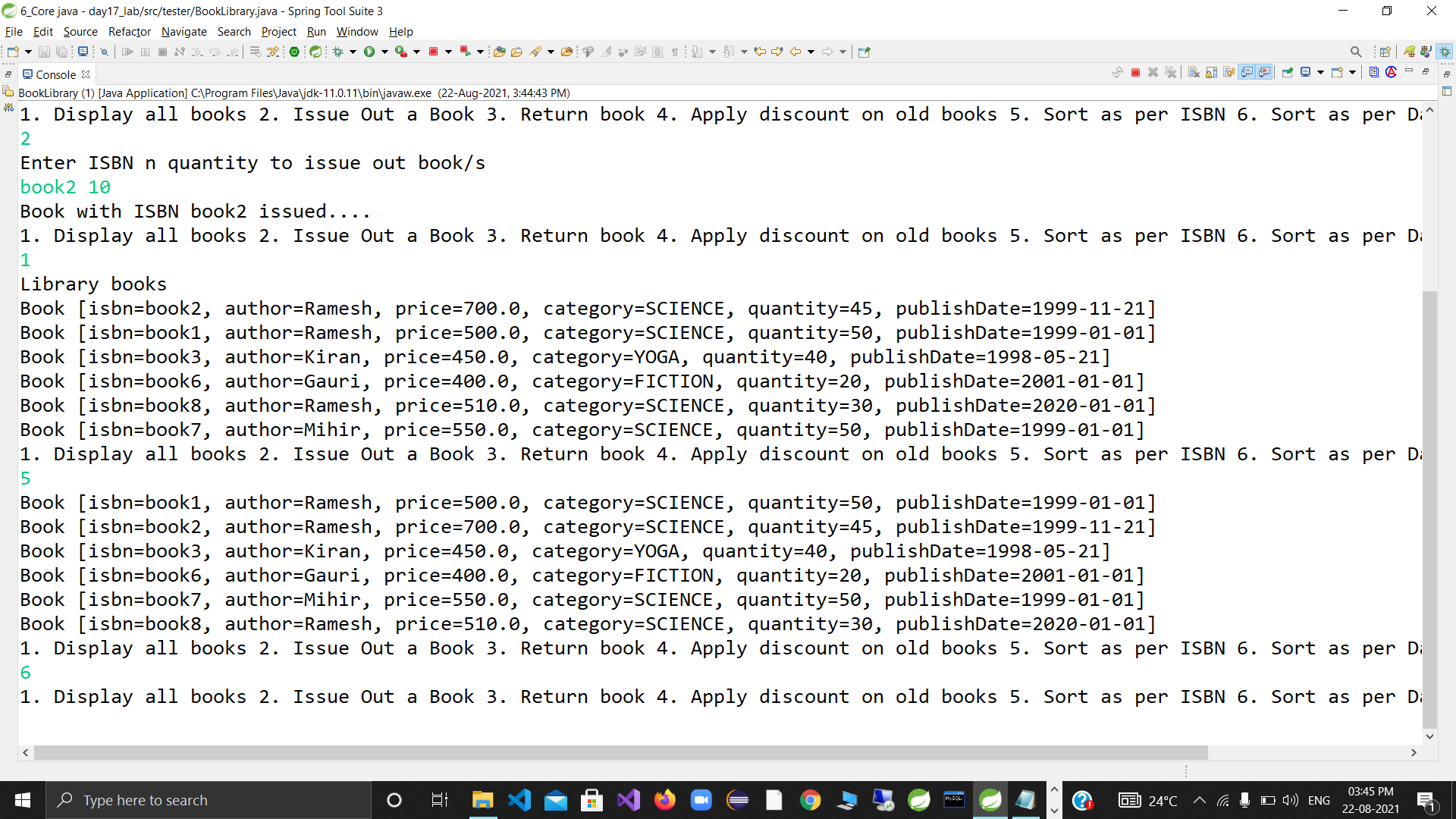
}

}

Output:







DAY 18:

Create multi threaded java application --- having 3 threads

main , even , odd.

Create EvenPrinterTask & OddPrinterTask

Accept from user (in main thread)-- start & end values.

Pass these values from main to child threads

Even no printer thread should print even nos within range, with small delay (sleep)

Odd no printer thread should print odd nos within range , with small delay (sleep)

Ensure no orphans.

Solve above using implements Runnable scenario.

**package** runnable\_tasks;

**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"+Thread.*currentThread*().getName());

}

@Override

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

{

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

**try** {

**for**(**int** i=start ;i<=end ;i++)

{

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

System.***out***.println("Even number "+ i + "exe thread"+Thread.*currentThread*().getName());

Thread.*sleep*(35);

}

}**catch** (Exception e) {

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

}

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

}

}

**package** runnable\_tasks;

**public** **class** OddPrinterTask2 **implements** Runnable{

**private** **int** start,end;

**public** OddPrinterTask2( **int** start, **int** end) {

**super**();

//this.num = num;

**this**.start = start;

**this**.end = end;

System.***out***.println("constrctor of"+getClass().getName()+"invoked by"+Thread.*currentThread*().getName());

}

@Override

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

{

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

**try** {

**for**(**int** i=start ;i<=end ;i++)

{

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

System.***out***.println("Odd number "+ i + "exe thread"+Thread.*currentThread*().getName());

Thread.*sleep*(50);

}

}**catch** (Exception e) {

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

}

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

}

}

package tester;

import java.util.Scanner;

import runnable\_tasks.EvenPrinterTask;

import runnable\_tasks.OddPrinterTask2;

public class TestEvenOddThread {

public static void main(String[] args) {

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

{

System.out.println("Enter the num begin and end values range");

int start=sc.nextInt();

int end=sc.nextInt();

Thread e1=new Thread(new EvenPrinterTask(start,end), "Even");

Thread e2= new Thread(new OddPrinterTask2(start,end), "Odd");

e1.start();

e2.start();

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

e1.join();

e2.join();

System.out.println("child thrds over....");

System.out.println("main thread getting over...");

} catch (Exception e) {

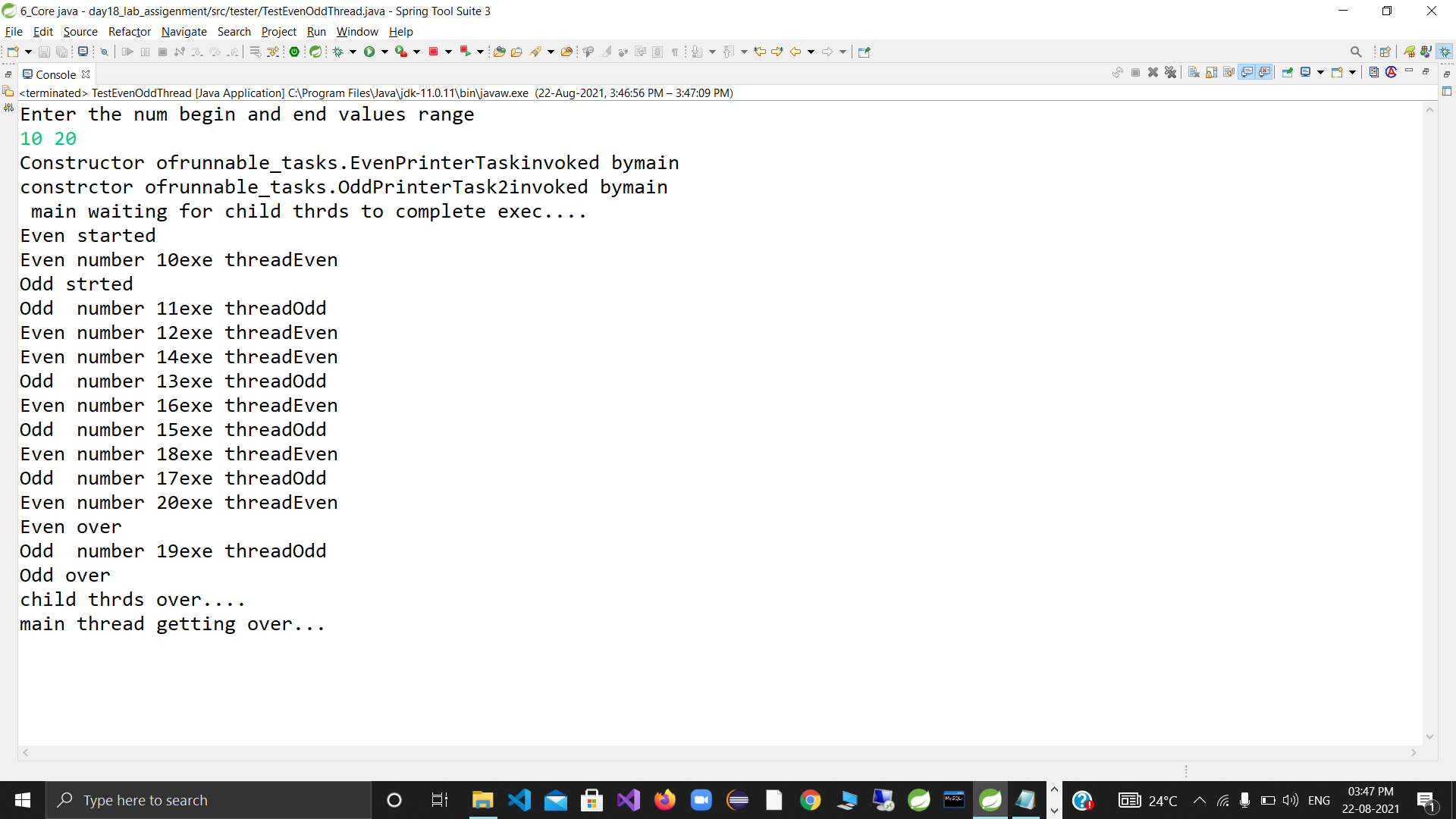
e.printStackTrace();

}

}

}

OUTPUT:



**package** thrd\_unsafe\_coll;

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

**public** **class** ThreadUnsafeCollections {

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

List<Integer> list = **new** ArrayList<>(10000);

**for** (**int** i = 0; i < 10000; i++)

list.add(i);

// Thread(Runnable instance,String name)

Thread reader = **new** Thread(() -> {

list.forEach(i -> {

**try** {

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

Thread.*sleep*(13);

} **catch** (Exception e) {

System.***out***.println("err in thrd " + Thread.*currentThread*().getName() + " exc " + e);

}

}

);// forEach : for-each : implicit Iterator(fail-fast)

}, "t1");

Thread writer = **new** Thread(() -> {

**try** {

**for** (**int** i = 10000; i < 20000; i++) {

list.add(i);

Thread.*sleep*(17);

}

} **catch** (Exception e) {

System.***out***.println("err in thrd " + Thread.*currentThread*().getName() + " exc " + e);

}

}, "t2");

reader.start();

writer.start();

reader.join();

writer.join();

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

}

}

**package** utils;

**import** java.util.Random;

**public** **class** SynchroUtils {

**private** **static** Random *r1* = **new** Random();

**public** **synchronized** **void** test() {

System.***out***.println("entered test "

+ Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(40) + 100);//100---139

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited test "

+ Thread.*currentThread*().getName());

}

**public** **synchronized** **void** testMe() {

System.***out***

.println("entered testMe " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(10) + 20);//20---29

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited testMe " + Thread.*currentThread*().getName());

}

**public** **void** testMeAgain() {

System.***out***.println("entered testMeagain "

+ Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(50) + 1);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited testMeagain "

+ Thread.*currentThread*().getName());

}

**public** **synchronized** **static** **void** test1() {

System.***out***.println("entered test1 " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(10) + 1);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited test1 " + Thread.*currentThread*().getName());

}

**public** **synchronized** **static** **void** test2() {

System.***out***.println("entered test2 " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(10) + 1);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited test2 " + Thread.*currentThread*().getName());

}

**public** **static** **void** test3() {

System.***out***.println("entered test3 " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(10) + 1);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

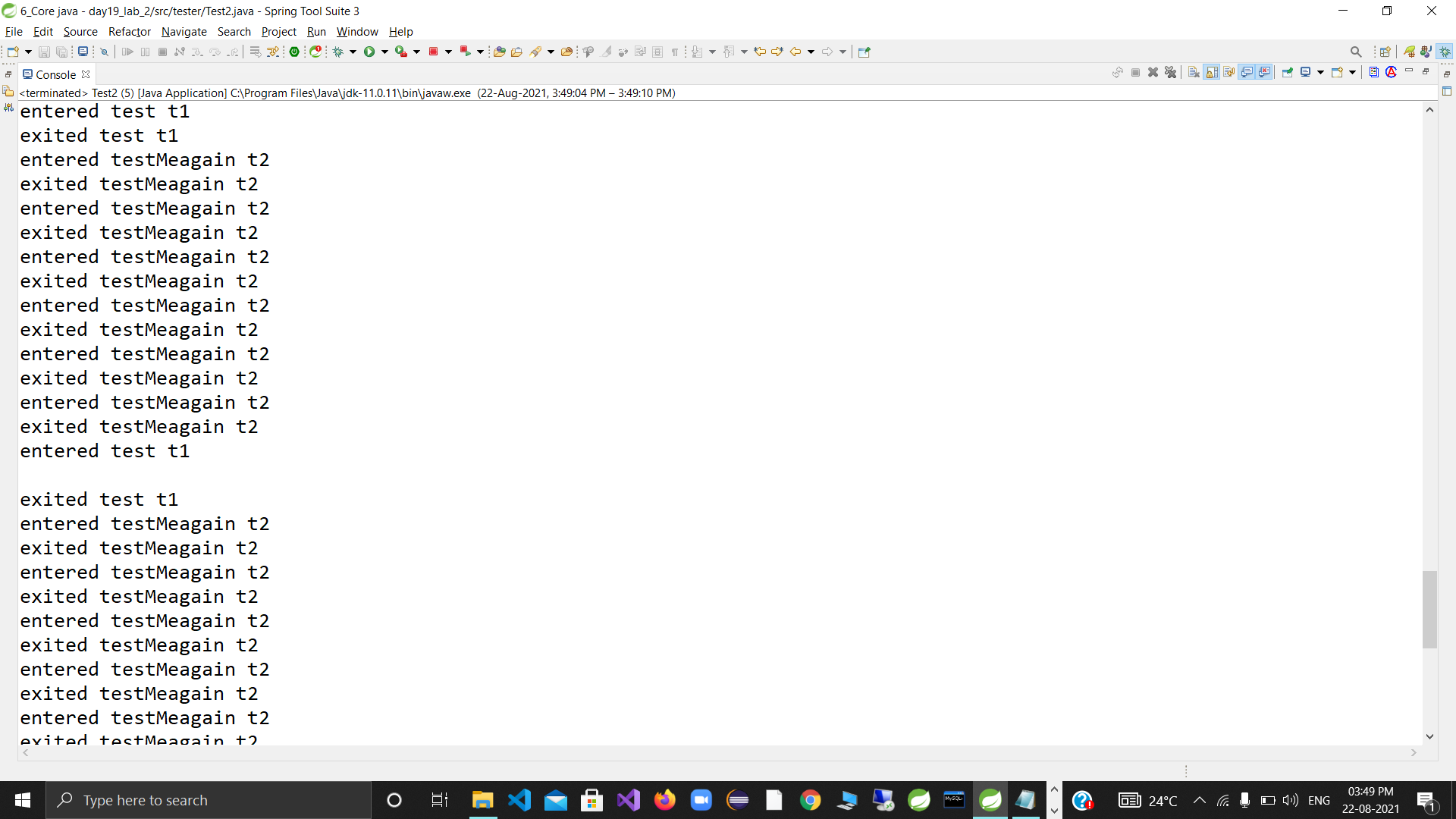
}

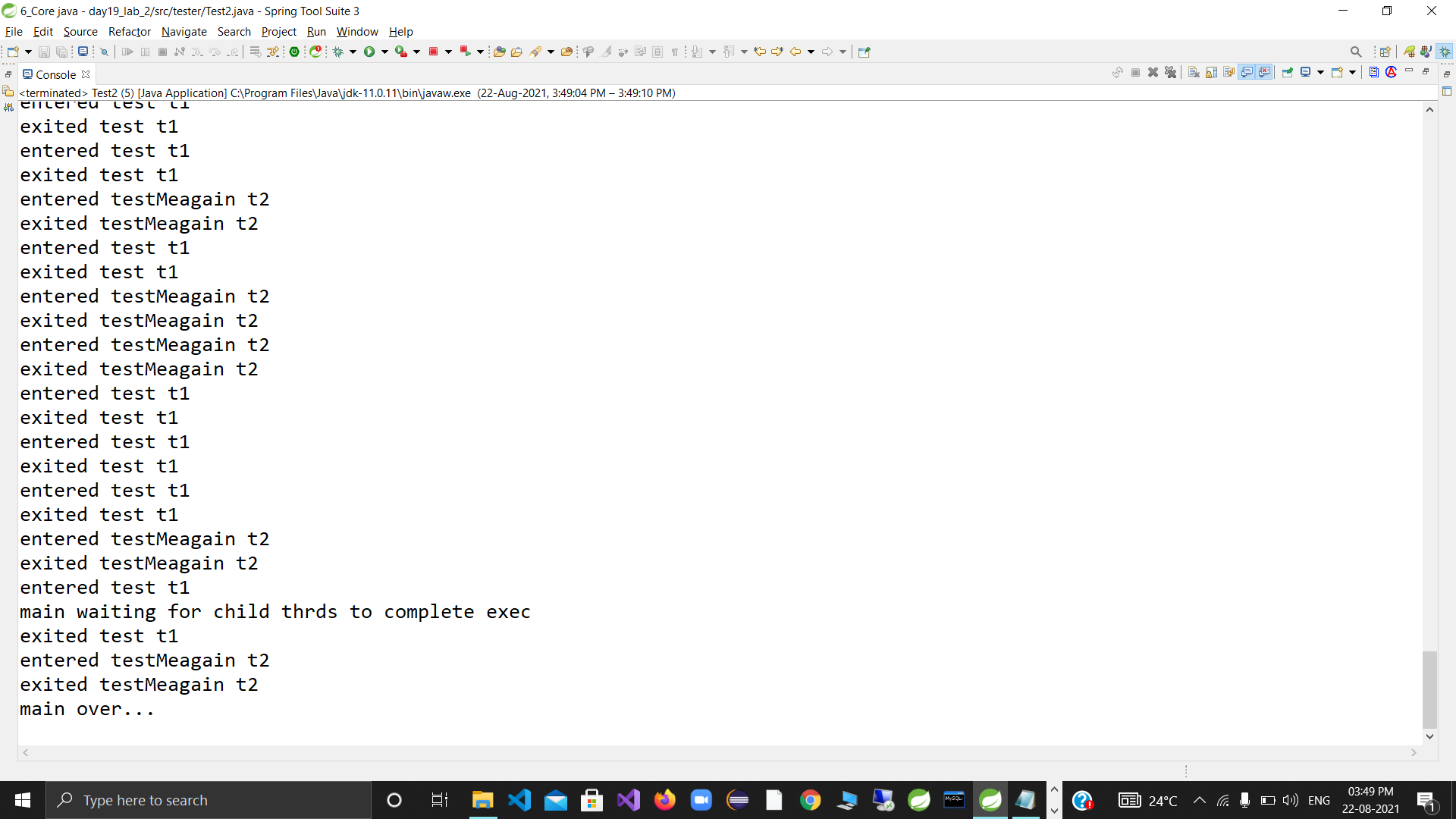
System.***out***.println("exited test3 " + Thread.*currentThread*().getName());

}

}

OUTPUT:





private int isbn;

private String title;

private String author;

private double price;

private int quantity;

private Category cat;

private Date publishedOn;

public static SimpleDateFormat sdf;

static {

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

}

public Book(int isbn, String title, String author, double price, int quantity, Category cat, Date publishedOn) {

this.isbn = isbn;

this.title = title;

this.author = author;

this.price = price;

this.quantity = quantity;

this.cat = cat;

this.publishedOn = publishedOn;

}

public int getIsbn() {

return isbn;

}

public void setIsbn(int isbn) {

this.isbn = isbn;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public Date getPublishedOn() {

return publishedOn;

}

public void setPublishedOn(Date publishedOn) {

this.publishedOn = publishedOn;

}

public Book(int isbn) {

this.isbn = isbn;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

@Override

public String toString() {

return "ISBN=" + isbn + ", Title=" + title + ", Author=" + author + ", Price=" + price + ", Quantity="

+ quantity + ", Catagory=" + cat + ", Published On =" + sdf.format(publishedOn);

}

@Override

public boolean equals(Object o)

{

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

if (o instanceof Book) {

return this.isbn == ((Book) o).isbn;

}

return false;

}

@Override

public int compareTo(Book anotherBook)

{

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

//sorting criteria : emp id

if(this.isbn < anotherBook.getIsbn())

return -1;

if(this.isbn == anotherBook.getIsbn())

return 0;

return 1;

//if you are sorting basic sortiong criteria on title

//return this.title.compareTo(anotherBook.title);

}

}

**package** com.app.core;

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

***PROGRAMMING***, ***MATHS***, ***SCIENCE***, ***POLITICS***, ***DATABASE***

}

**package** custom\_exception;

@SuppressWarnings("serial")

**public** **class** BookException **extends** Exception{

**public** BookException(String msg) {

**super**(msg);

}

}

package utils;

import java.text.ParseException;

import java.util.ArrayList;

import com.app.core.Book;

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

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

public class CollectionUtils {

public static ArrayList<Book> populatebook() throws ParseException {

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

list.add(new Book(347062, "LET US C", "Yashwant kanitkar", 300.00, 100, PROGRAMMING, sdf.parse("01-12-2009")));

list.add(new Book(347034, "CPP", "Denis", 254.00, 2000, DATABASE, sdf.parse("01-10-2011")));

list.add(new Book(347012, "Chemistry", "Smith Gorge", 294.00, 300, SCIENCE, sdf.parse("01-01-2010")));

list.add(new Book(347045, "Integration ", "Matyu Garga", 400.00, 400, MATHS,

sdf.parse("01-08-2012")));

list.add(new Book(347078, " Money Of Country", " Ramchatrandra ", 1700.00, 500, POLITICS,

sdf.parse("11-11-2016")));

return list;

}

}

package book\_library\_tester;

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

import java.text.ParseException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Date;

import java.util.Iterator;

import java.util.Scanner;

import com.app.core.Book;

import com.app.core.Category;

import custom\_exception.BookException;

import static utils.CollectionUtils.populatebook;

public class BookShop {

public static void main(String[] args) throws BookException, ParseException {

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

ArrayList<Book> books = populatebook();

boolean exit = false;

while (!exit) {

System.out.println("Options");

System.out.println("1. Display All Book Details");

System.out.println("2. Issue out book");

System.out.println("3. Return book");

System.out.println("4. Apply discount on old books ");

System.out.println("5. Sort the books as per isbn");

System.out.println("100. Exit");

try {

switch (sc.nextInt())

{

case 1:

System.out.println("All Book Details");

for (Book b : books)

{

System.out.println(b);

}

break;

case 2:

System.out.println("Enter ISBN number and Quantity of Books To be Issued");

int index = books.indexOf(new Book(sc.nextInt()));

int quantity = sc.nextInt();

Book b1 = books.get(index);

if (index == -1) {

throw new BookException("Invalid ISBN id : BOOK not found!!!!!");

}

else

{

if (b1.getQuantity() < quantity) {

System.out.println("insufficient quantity of book Available");

}

else if (b1.getQuantity() > quantity)

{

b1.setQuantity(b1.getQuantity() - quantity);

}

System.out.println("Books Issued: " + books.get(index) + " Quantity: " + quantity);

}

break;

case 3:

System.out.println("Enter ISBN number and Quantity of Books To be Returned ");

int index1 = books.indexOf(new Book(sc.nextInt()));

int quantity1 = sc.nextInt();

Book b2 = books.get(index1);

if (index1 == -1)

{

throw new BookException("Invalid ISBN id : BOOK not found!!!!!");

}

else

{

b2.setQuantity(b2.getQuantity() + quantity1);//90=10=100

System.out.println("Books Returned: " + books.get(index1) + " Quantity: " + quantity1);

}

break;

case 4:

System.out.println("Enter DATE and Discount amount To be Applied ");

Date thresholddate = sdf.parse(sc.next());

double discount = sc.nextDouble();

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

while (itr.hasNext())

{

Book b3 = itr.next();

if (b3.getPublishedOn().before(thresholddate))

{

b3.setPrice(b3.getPrice() - discount);

}

}

break;

case 5:

Collections.sort(books);

for (Book b4 : books) {

System.out.println(b4);

}

break;

case 100:

exit = true;

break;

}

} catch (Exception e) {

System.out.println(e);

}

sc.nextLine();

}

} catch (Exception e) {

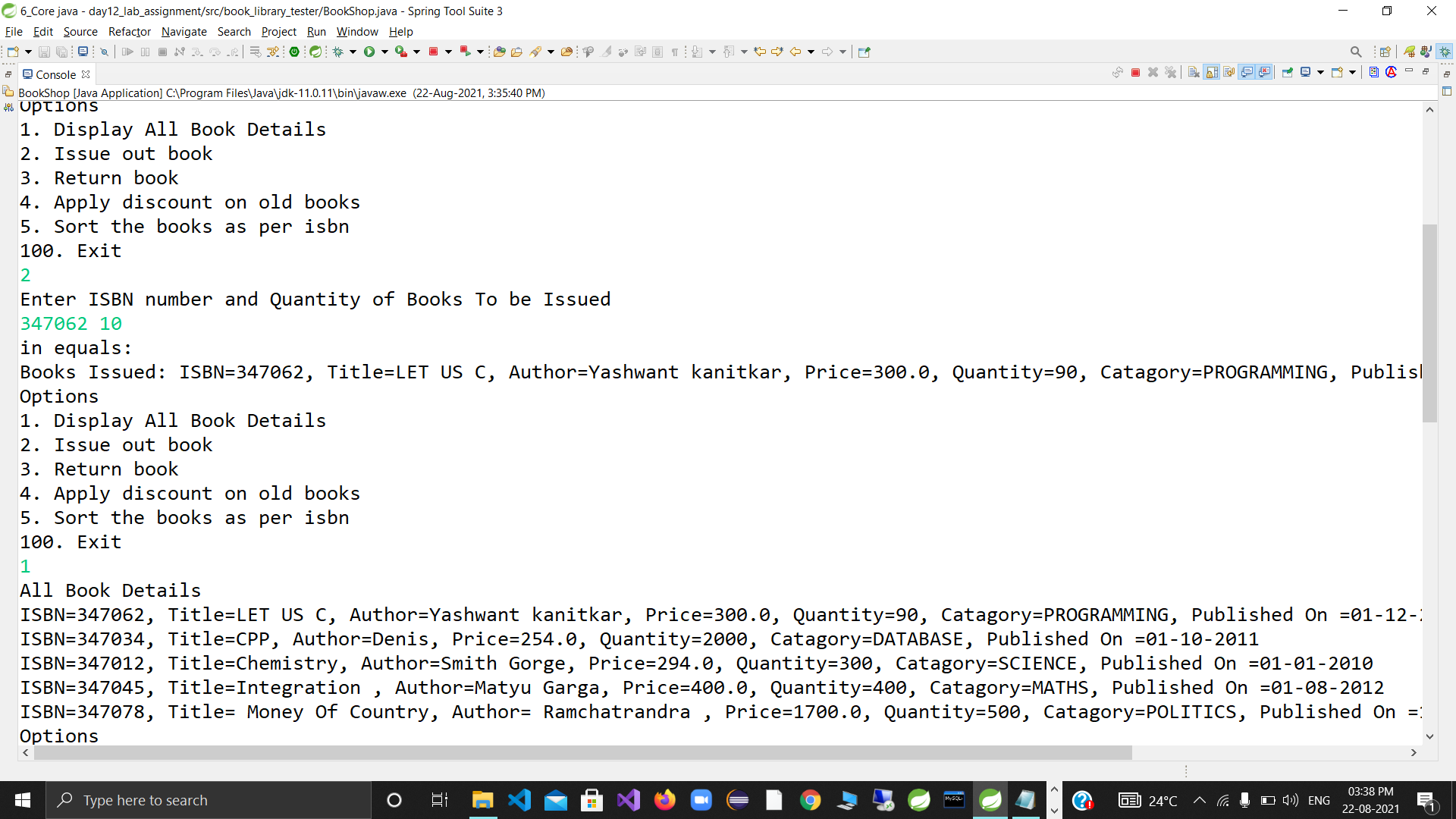
System.out.println(e);

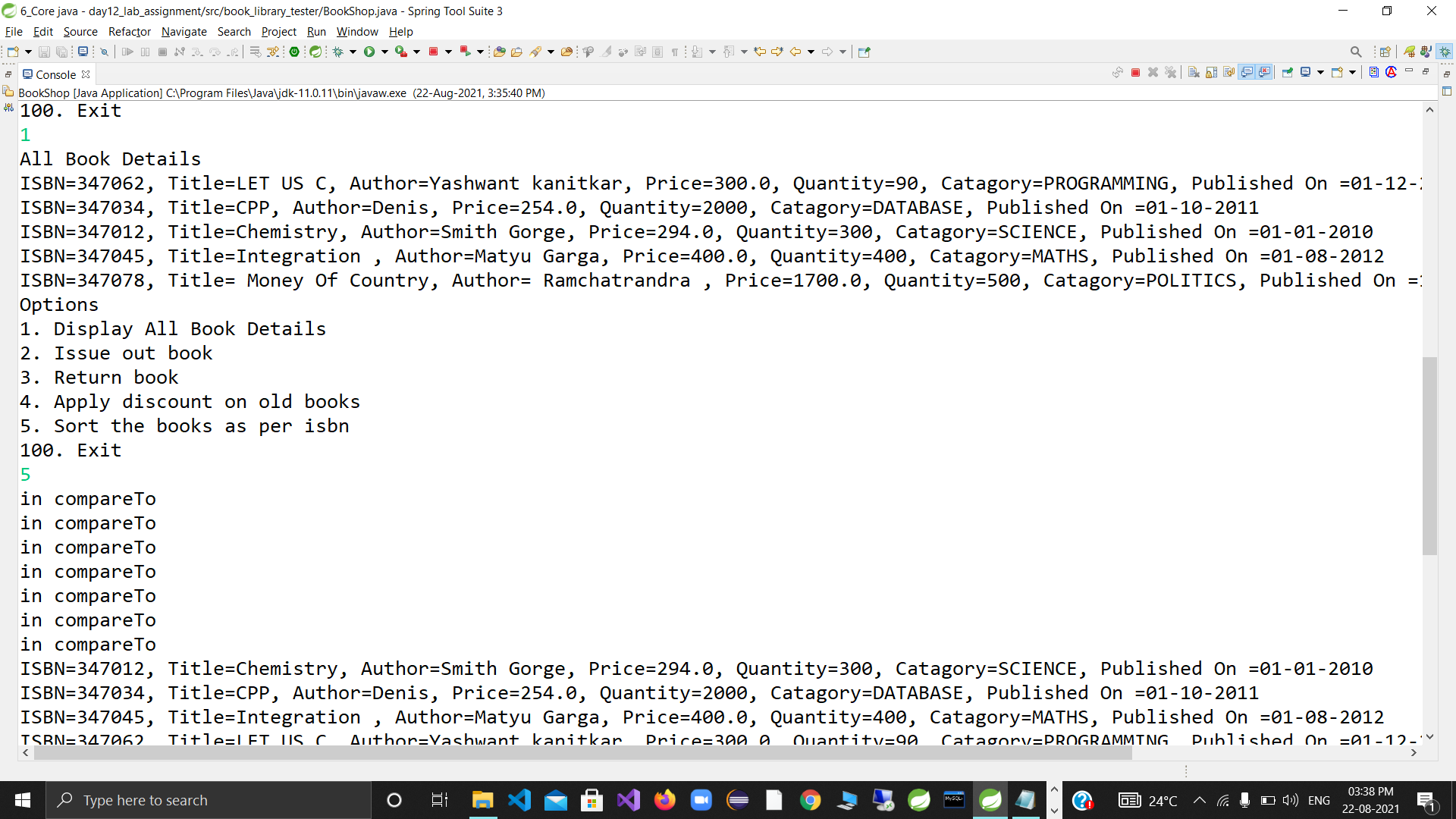
}

}

}







package com.app.core;

import java.time.LocalDate;

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 LocalDate joinDate;

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

super();

this.id = id;

this.name = name;

this.salary = salary;

this.dept = dept;

this.joinDate = joinDate;

}

@Override

public String toString() {

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

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

+joinDate;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public Department getDept() {

return dept;

}

public void setDept(Department dept) {

this.dept = dept;

}

public LocalDate getJoinDate() {

return joinDate;

}

public void setJoinDate(LocalDate joinDate) {

this.joinDate = joinDate;

}

}

**package** com.app.core;

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

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

}

**package** cust\_exection;

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

**public** EmpHandlingException (String msg){

**super**(msg);

}

}

package utils;

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

import static java.time.LocalDate.parse;

import java.util.HashMap;

import java.util.Map;

import com.app.core.Department;

import com.app.core.Emp;

public interface CollectionUtils {

static Map<Integer, Emp> populateEmpData()

{

Map<Integer,Emp> map=new HashMap<>();

map.put(101, new Emp(101,"Sagar",20000,FINANCE,parse("1999-01-01")));

map.put(201, new Emp(201,"Mahesh",30000,HR,parse("2000-07-11")));

map.put(301, new Emp(301,"Kajal",40000,MARKETING,parse("2001-05-11")));

map.put(401, new Emp(401,"Mayur",25000,FINANCE,parse("2001-03-04")));

map.put(4501, new Emp(401,"Vivek",35000,SALES,parse("2003-06-01")));

return map;

}

}

package tester;

import java.util.Map;

import java.util.Scanner;

import static utils.CollectionUtils.populateEmpData;

import com.app.core.Emp;

import cust\_exection.EmpHandlingException;

public class Emp\_tester {

public static void main(String[] args) {

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

Map<Integer, Emp> emps=populateEmpData();

boolean exit = false;

while(!exit)

{

System.out.println("1 display all emp 2 get emp details through emp id");

try {

switch (sc.nextInt()) {

case 1:

System.out.println("All emp Details:");

emps.forEach((id,emp)->System.out.println(emp));

break;

case 2:

System.out.println("Enter emp id , to fecth details");

int eid=sc.nextInt();

Emp emp=emps.get(eid);

if(emp==null)

throw new EmpHandlingException("Invalid emp id");

System.out.println("emp detials"+emp);

case 100:

exit=true;

break;

}

}

catch (Exception e) {

e.printStackTrace();

}

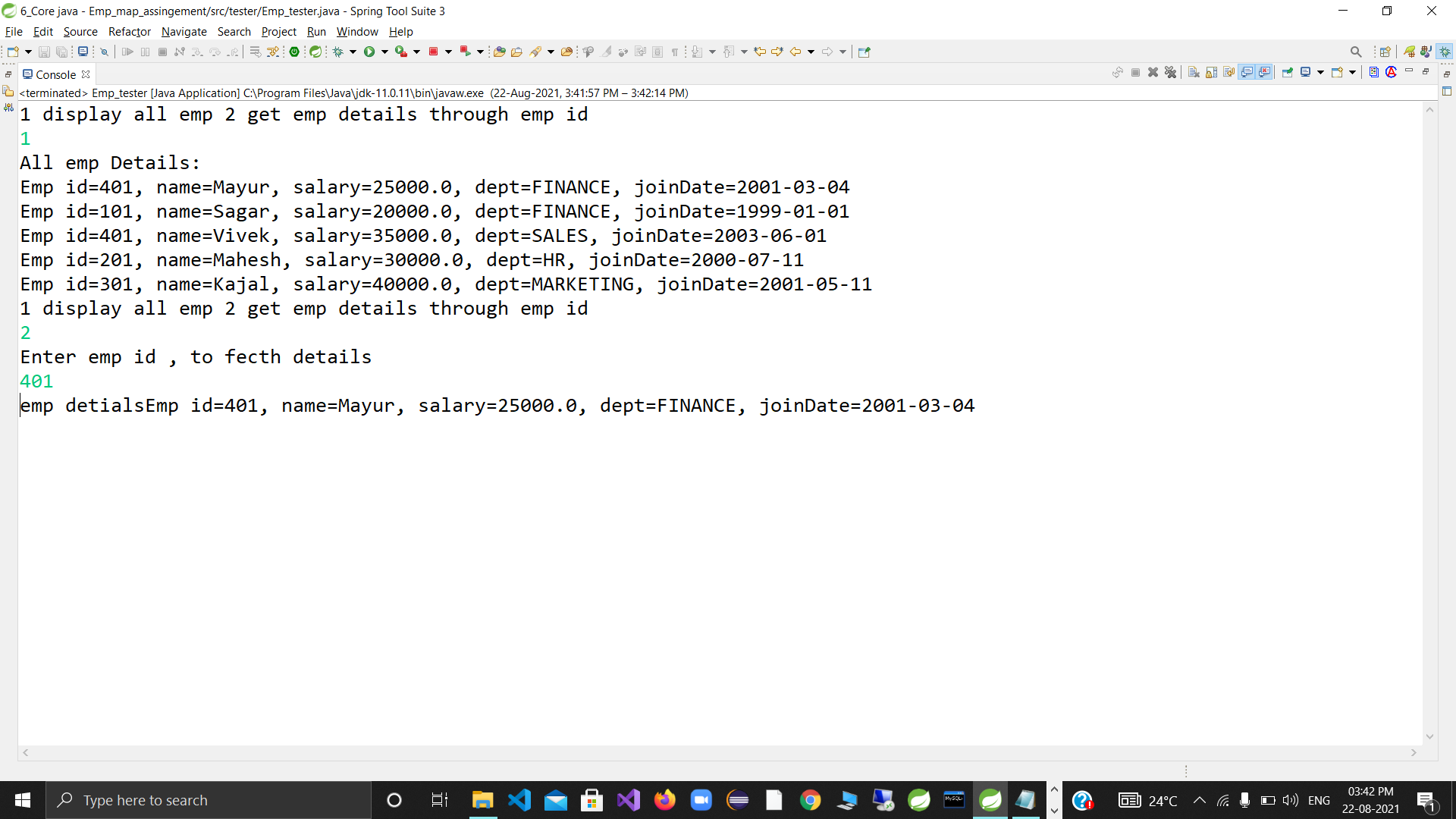
sc.nextLine();

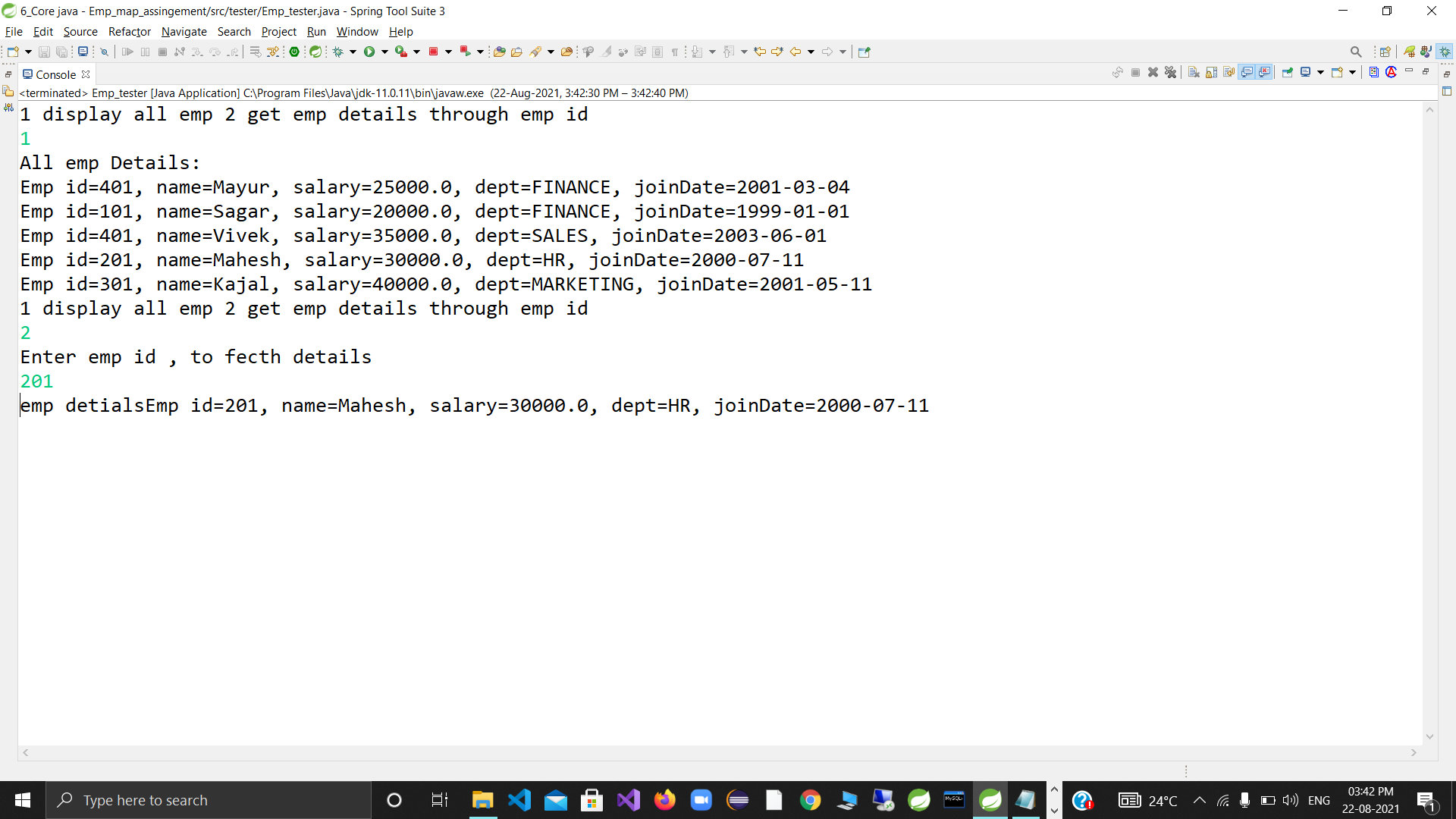
}

}

}

}





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;

}

}

**package** com.app.core;

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

***TECHNOLOGY***, ***SCIENCE***, ***SELF\_HELP***, ***YOGA***, ***MUSIC***,***FICTION***

}

**package** custom\_exception;

@SuppressWarnings("serial")

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

**public** BookHandlingException(String mesg) {

**super**(mesg);

}

}

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;

}

}

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 {

//add a static method to store library books using serialization

//method : public void writeObject(Object o) throws IOExc.

static void storeBooks(Map<String,Book> books,String fileName) throws IOException

{

//Java App ---> OOS ---> FOS(bin file)

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

{

out.writeObject(books);

}

}

//add a static method to restore books : from bin file

//in case of any exc --return map populated with sample data

//in case of no err --return map populated with data from bin file.

//method : public Object readObject() throws IOExc,ClassNotFoundException

@SuppressWarnings("unchecked")

static Map<String,Book> restoreBooks(String fileName)

{

//chain : Java App <------OIS <---- FIS(bin file)

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

{

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

}catch (Exception e) {

//de -serial failed : so ret sample data map

return populateBooks();

}

}

}

package tester;

import static utils.SerIOUtils.\*;

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;

public class BookLibrary {

public static void main(String[] args) {

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

System.out.println("Enter file name to restore the books");

String fileName=sc.nextLine();

// get lib books

Map<String, Book> books = restoreBooks(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);

break;

case 100:

exit = true;

//store the books in bin file

storeBooks(books,fileName);

break;

}

} catch (Exception e) {

e.printStackTrace();

}

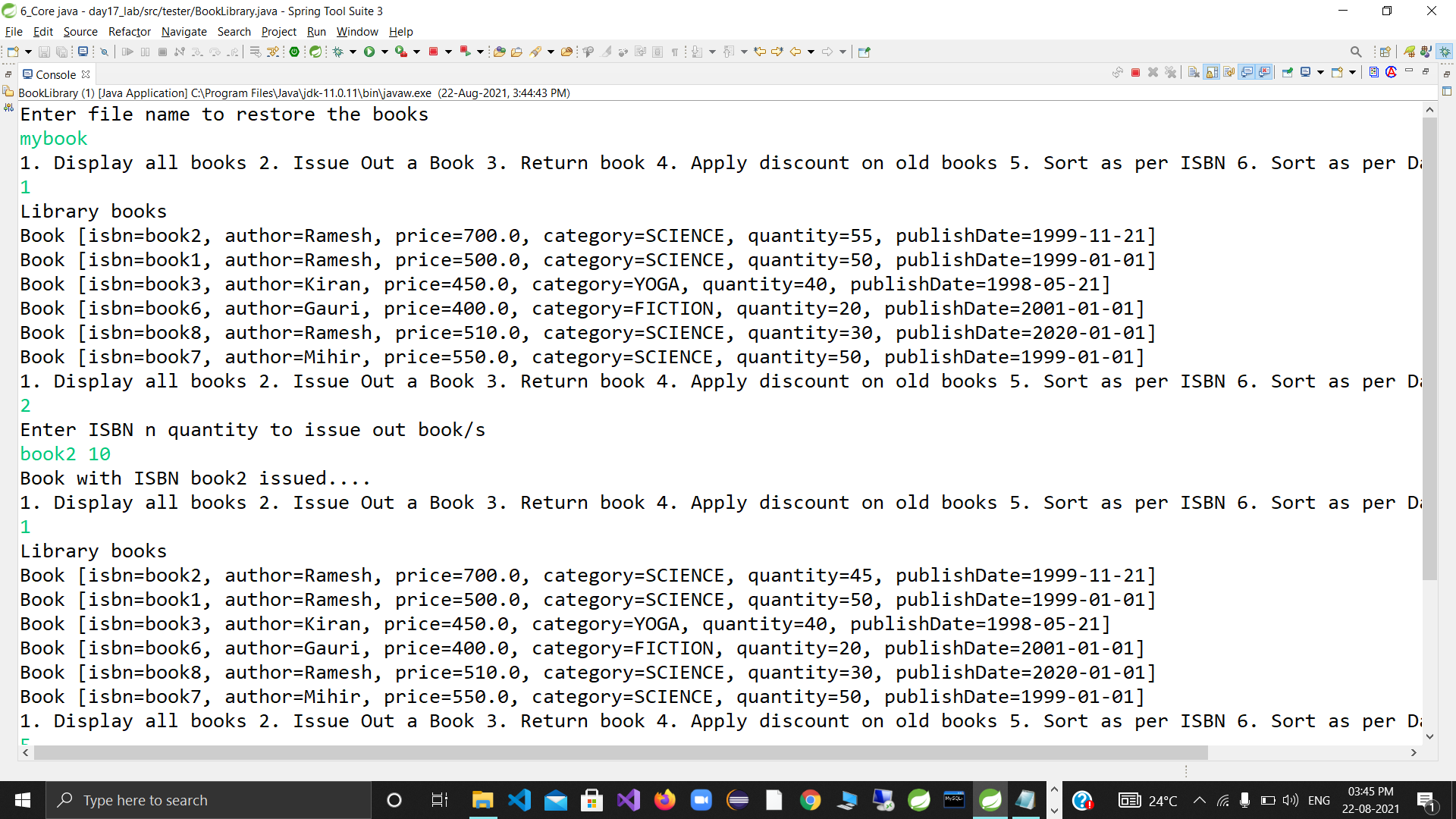
sc.nextLine();

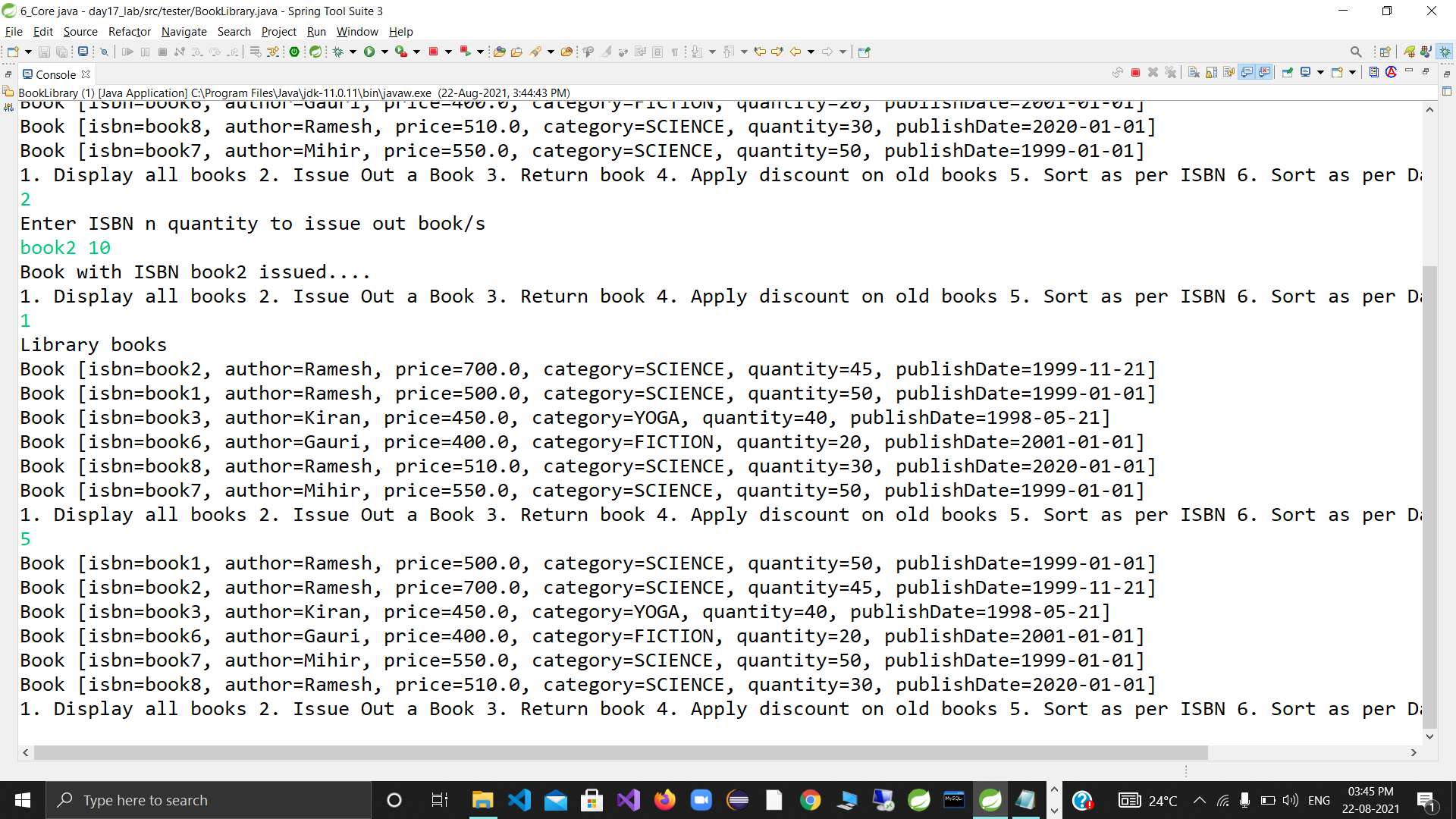
}

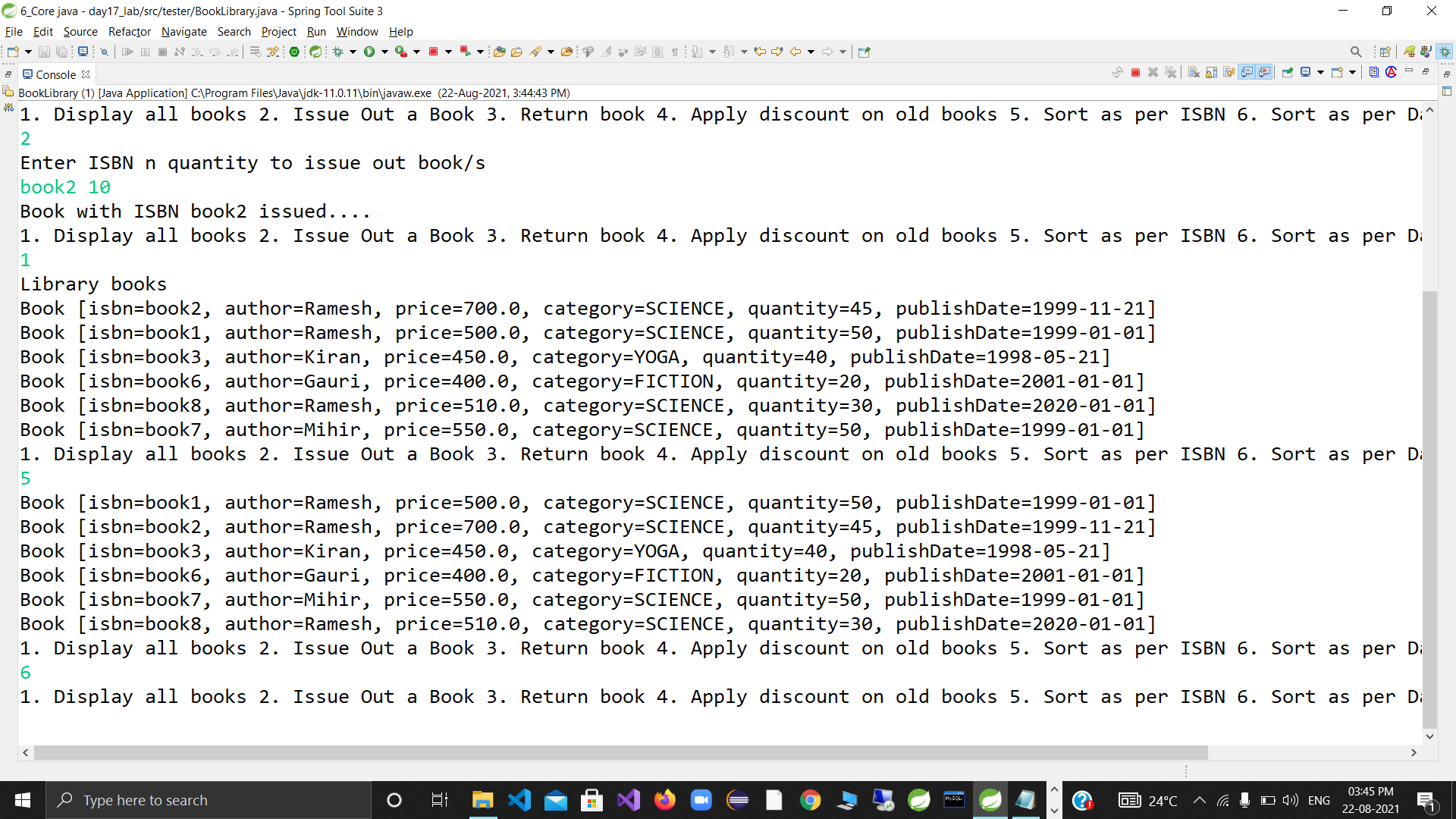
}

}

}







Day 18

**package** runnable\_tasks;

**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"+Thread.*currentThread*().getName());

}

@Override

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

{

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

**try** {

**for**(**int** i=start ;i<=end ;i++)

{

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

System.***out***.println("Even number "+ i + "exe thread"+Thread.*currentThread*().getName());

Thread.*sleep*(35);

}

}**catch** (Exception e) {

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

}

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

}

}

**package** runnable\_tasks;

**public** **class** OddPrinterTask2 **implements** Runnable{

**private** **int** start,end;

**public** OddPrinterTask2( **int** start, **int** end) {

**super**();

//this.num = num;

**this**.start = start;

**this**.end = end;

System.***out***.println("constrctor of"+getClass().getName()+"invoked by"+Thread.*currentThread*().getName());

}

@Override

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

{

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

**try** {

**for**(**int** i=start ;i<=end ;i++)

{

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

System.***out***.println("Odd number "+ i + "exe thread"+Thread.*currentThread*().getName());

Thread.*sleep*(50);

}

}**catch** (Exception e) {

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

}

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

}

}

package tester;

import java.util.Scanner;

import runnable\_tasks.EvenPrinterTask;

import runnable\_tasks.OddPrinterTask2;

public class TestEvenOddThread {

public static void main(String[] args) {

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

{

System.out.println("Enter the num begin and end values range");

int start=sc.nextInt();

int end=sc.nextInt();

Thread e1=new Thread(new EvenPrinterTask(start,end), "Even");

Thread e2= new Thread(new OddPrinterTask2(start,end), "Odd");

e1.start();

e2.start();

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

e1.join();

e2.join();

System.out.println("child thrds over....");

System.out.println("main thread getting over...");

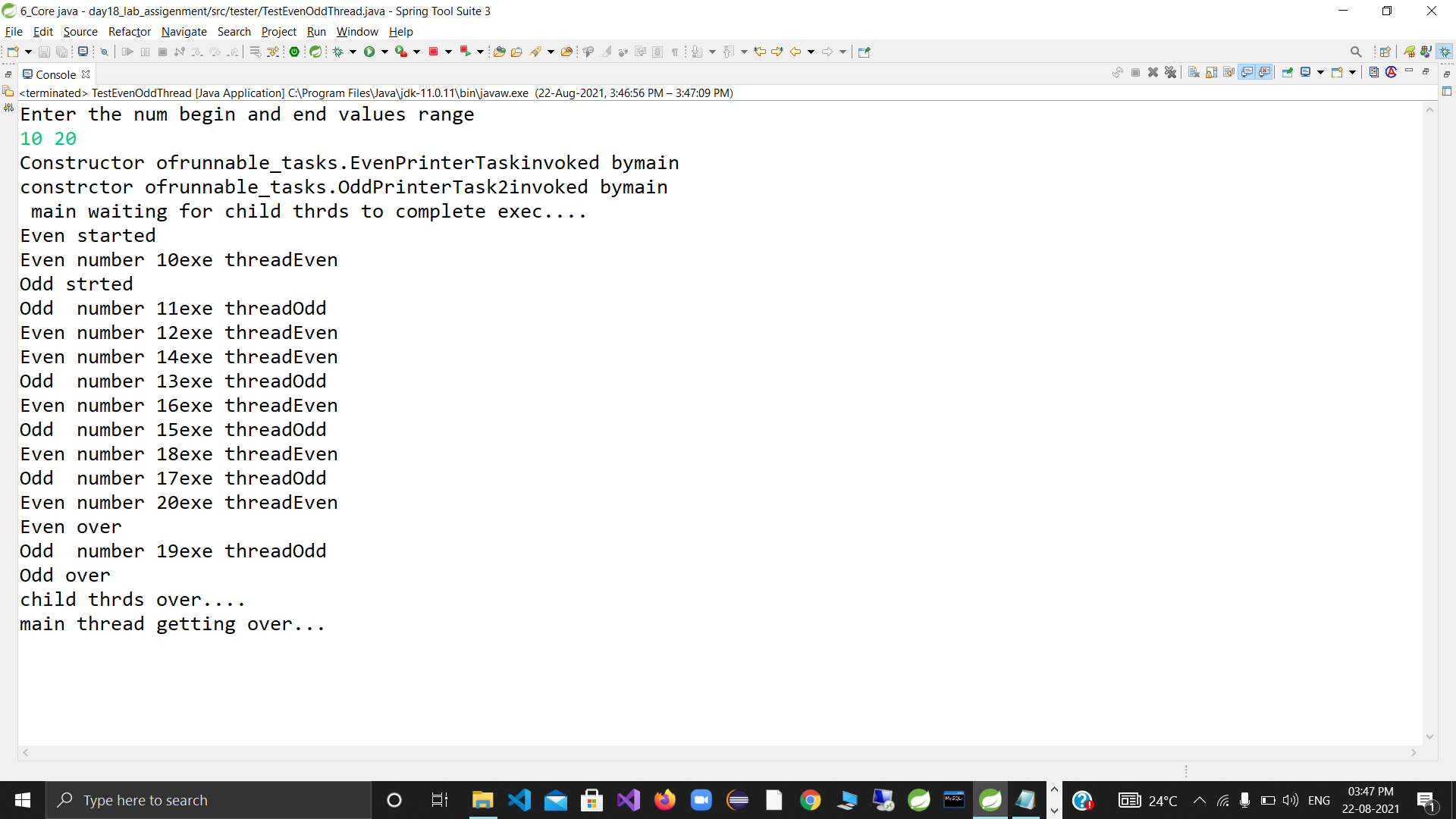
} catch (Exception e) {

e.printStackTrace();

}

}

}



Day 19

**package** thrd\_unsafe\_coll;

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

**public** **class** ThreadUnsafeCollections {

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

List<Integer> list = **new** ArrayList<>(10000);

**for** (**int** i = 0; i < 10000; i++)

list.add(i);

// Thread(Runnable instance,String name)

Thread reader = **new** Thread(() -> {

list.forEach(i -> {

**try** {

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

Thread.*sleep*(13);

} **catch** (Exception e) {

System.***out***.println("err in thrd " + Thread.*currentThread*().getName() + " exc " + e);

}

}

);// forEach : for-each : implicit Iterator(fail-fast)

}, "t1");

Thread writer = **new** Thread(() -> {

**try** {

**for** (**int** i = 10000; i < 20000; i++) {

list.add(i);

Thread.*sleep*(17);

}

} **catch** (Exception e) {

System.***out***.println("err in thrd " + Thread.*currentThread*().getName() + " exc " + e);

}

}, "t2");

reader.start();

writer.start();

reader.join();

writer.join();

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

}

}

**package** utils;

**import** java.util.Random;

**public** **class** SynchroUtils {

**private** **static** Random *r1* = **new** Random();

**public** **synchronized** **void** test() {

System.***out***.println("entered test "

+ Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(40) + 100);//100---139

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited test "

+ Thread.*currentThread*().getName());

}

**public** **synchronized** **void** testMe() {

System.***out***

.println("entered testMe " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(10) + 20);//20---29

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited testMe " + Thread.*currentThread*().getName());

}

**public** **void** testMeAgain() {

System.***out***.println("entered testMeagain "

+ Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(50) + 1);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited testMeagain "

+ Thread.*currentThread*().getName());

}

**public** **synchronized** **static** **void** test1() {

System.***out***.println("entered test1 " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(10) + 1);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited test1 " + Thread.*currentThread*().getName());

}

**public** **synchronized** **static** **void** test2() {

System.***out***.println("entered test2 " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(10) + 1);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited test2 " + Thread.*currentThread*().getName());

}

**public** **static** **void** test3() {

System.***out***.println("entered test3 " + Thread.*currentThread*().getName());

**try** {

Thread.*sleep*(*r1*.nextInt(10) + 1);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("exited test3 " + Thread.*currentThread*().getName());

}

}

