**1. Create Java classes having suitable attributes for Library management system.Use OOPs concepts in your design.Also try to use interfaces and abstract classes.**

**Solution**

**package ttn.library;**

**public class Book {**

**int id;**

**String title;**

**String authorName;**

**int countAvailable;**

**int rackNumber;**

**public Book(int id, String title, String authorName, int countAvailable, int rackNumber) {**

**this.id = id;**

**this.title = title;**

**this.authorName = authorName;**

**this.countAvailable = countAvailable;**

**this.rackNumber = rackNumber;**

**}**

**public int getId() {**

**return id;**

**}**

**public void setId(int id) {**

**this.id = id;**

**}**

**public String getTitle() {**

**return title;**

**}**

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

**this.title = title;**

**}**

**public String getAuthorName() {**

**return authorName;**

**}**

**public void setAuthorName(String authorName) {**

**this.authorName = authorName;**

**}**

**public int getCountAvailable() {**

**return countAvailable;**

**}**

**public void setCountAvailable(int countAvailable) {**

**this.countAvailable = countAvailable;**

**}**

**public int getRackNumber() {**

**return rackNumber;**

**}**

**public void setRackNumber(int rackNumber) {**

**this.rackNumber = rackNumber;**

**}**

**}**

**package ttn.library;**

**import java.util.Date;**

**import java.util.List;**

**public class BookIssue {**

**int issueId;**

**List<Book> bookList;**

**Date issueDate;**

**Date dueDate;*//one issue id will have one issue date***

***//Constructor and getter setter***

**}**

**package ttn.library;**

**import java.util.List;**

**public abstract class Client {**

**private String name;**

***//Constructor and getter setter here***

**public abstract void requestForBooksIssue();**

**public void returnBooks(List<Book>bookList){**

**System.*out*.println("execute the return flow");**

**}**

**}**

**package ttn.library;**

**import java.util.List;**

**public class Librarian {**

**String name;**

**public boolean checkCardValidity(Member member)**

**{**

**System.*out*.println("Calls member.isCardValid");**

**return false;**

**}**

**public int issueBooks(List<Book>bookList)**

**{**

**System.*out*.println("1.Check if it's a member or non member, and limit on how many books can client issue");**

**System.*out*.println("In case of member,checks for card vailidity using checkCardValidity,in case of non-membber generates temporary card");**

**System.*out*.println("check availability of book");**

**System.*out*.println("creates an object of BookIssueAccordingly");**

**System.*out*.println("Issued books are saved in system with client");**

**System.*out*.println("Returns valid issueId in case of success");**

**return 0;**

**}**

**public void takeBookReturns(BookIssue bookIssue)**

**{**

**System.*out*.println("Check for history..if exceeds due date calculate fine,else take return");**

**}**

**public float calculateFine(BookIssue issue)**

**{**

**System.*out*.println("calculates fine based on number of books and due date");**

**return 1.0f;**

**}**

**public int issueTemporaryCard(NonMember nonMember)**

**{**

**System.*out*.println("return temporary card id for a non-member valid for fix time");**

**return 1;**

**}**

**public boolean checkAvailabilityOfBook(String title){**

**System.*out*.println("Checks for availability of book by title of book and returns status");**

**return false;**

**}**

**public int findBook(String title)**

**{**

**System.*out*.println("Gets the rack number of a book where it is placed, using title");**

**return 0;**

**}**

**}**

**package ttn.library;**

**import java.util.Date;**

**public class Member extends Client {**

**long phoneNumber;**

**String address;**

**Date cardIssueDate;**

**Date cardExpiryDate;**

**@Override**

**public void requestForBooksIssue() {**

**System.*out*.println("Show card to librarian");**

**System.*out*.println("librarian checks for validity of card in issueBooks()");**

**System.*out*.println("If valid provide names of books");**

**}**

**public boolean isCardValid()**

**{**

**System.*out*.println("Check if expiry date < current date return true/false");**

**return false;**

**}**

**}**

**package ttn.library;**

**public class NonMember extends Client {**

**@Override**

**public void requestForBooksIssue() {**

**System.*out*.println("Requests for books");**

**System.*out*.println("Librarian issues a temporary cardId valid only for 15 days");**

**System.*out*.println("Books are issued");**

**}**

**}**

**2. WAP to sorting string without using string Methods?.**

**Solution**

**import java.util.Arrays;**

**import java.util.Scanner;**

**public class SortStrings {**

**static Scanner *in*;**

***//utility function to read an array***

**static void readArray(String[]inputArray,int eleCount)**

**{**

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

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

**{**

**System.*out*.println("Enter"+(i+1)+" th string");**

**inputArray[i]=*in*.next();**

**}**

**}**

**static int compareStrings(String first,String second)**

**{**

**int firstLength = first.length();**

**int secondLength = second.length();**

**int min = firstLength<secondLength?firstLength:secondLength;**

**for (int i = 0; i < min; i++) {**

**char c1 = first.charAt(i);**

**char c2 = second.charAt(i);**

**if (c1 != c2) {**

**return c1 - c2;**

**}**

**}**

**return firstLength-secondLength;**

**}**

**static void sortStringArray(String[]inputArray)**

**{**

**for(int i=0;i<inputArray.length-1;i++)**

**{**

**int minIndex=i;**

**for(int j=i+1;j<inputArray.length;j++)**

**{**

**int res=*compareStrings*(inputArray[j],inputArray[minIndex]);**

**if(res<0)**

**minIndex=j;**

**}**

**String temp=inputArray[minIndex];**

**inputArray[minIndex]=inputArray[i];**

**inputArray[i]=temp;**

**}**

**}**

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

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

**System.*out*.println("Enter number of elements in array");**

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

**String[] inputArray=new String[eleCount];**

***readArray*(inputArray,eleCount);**

**System.*out*.println("Array before sorting");**

**System.*out*.println(Arrays.*toString*(inputArray));**

***sortStringArray*(inputArray);**

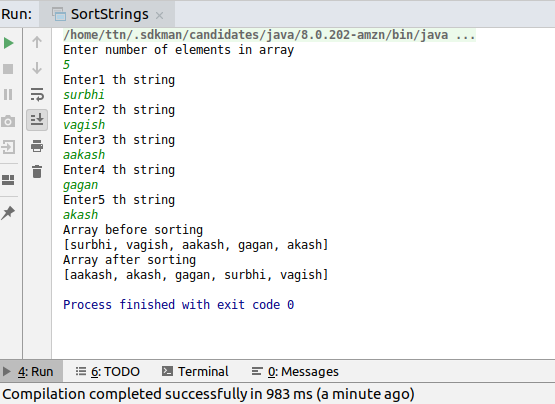
**System.*out*.println("Array after sorting");**

**System.*out*.println(Arrays.*toString*(inputArray));**

**}**

**}**

**Output**



**3. WAP to produce NoClassDefFoundError and ClassNotFoundException exception.**

**Solution**

public class ErrorException {

public static void main(String[] args) {

Greeter greeter=new Greeter();

greeter.greet();

try {

Class.forName("oracle.jdbc.driver.OracleDriver");

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

}

public class Greeter

{

public void greet()

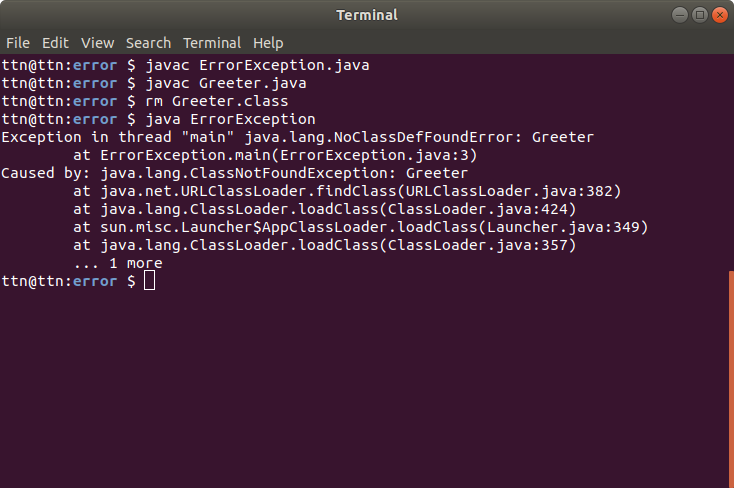
{

System.out.println("Hello");

}

}

**Execution to produce both error and exception**



**4. WAP to create singleton class.**

**Solution**

**public class Singleton {**

***/\*Three most common things to make class singleton\*/***

***//1private constructor so that it can't be instantiated outside***

**private Singleton()**

**{**

**}**

***//2private static variable of same class--->only one object possible***

**private static Singleton *singleton*;**

***//3public static method to get instance--global access point***

***//double locking--to avoid extra overhead of synchronized method***

**public static Singleton getInstance()**

**{**

**if(*singleton*==null)**

**{**

**synchronized (Singleton.class) {**

**if(*singleton*==null)**

***singleton* = new Singleton();**

**}**

**}**

**return *singleton*;**

**}**

**}**

**5. WAP to show object cloning in java using cloneable and copy constructor both.**

**Solution**

**Person.java**

**public class Person implements Cloneable {**

**int personId;**

**String name;**

**Address address;**

**//Copy constructor**

**public Person(Person source)**

**{**

**this.personId=source.personId;**

**this.name=source.name;**

**this.address=new Address(source.address.getPlace(),source.address.getZip());**

**}**

**@Override**

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

**return super.clone();**

**}**

**public Person(int personId, String name, Address address) {**

**this.personId = personId;**

**this.name = name;**

**this.address = address;**

**}**

**public int getPersonId() {**

**return personId;**

**}**

**public void setPersonId(int personId) {**

**this.personId = personId;**

**}**

**public String getName() {**

**return name;**

**}**

**@Override**

**public String toString() {**

**return "Person{" +**

**"personId=" + personId +**

**", name='" + name + '\'' +**

**", address=" + address +**

**'}';**

**}**

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

**this.name = name;**

**}**

**}**

**Address.java**

**public class Address {**

**String place;**

**int zip;**

**@Override**

**public String toString() {**

**return "Address{" +**

**"place='" + place + '\'' +**

**", zip=" + zip +**

**'}';**

**}**

**public String getPlace() {**

**return place;**

**}**

**public void setPlace(String place) {**

**this.place = place;**

**}**

**public int getZip() {**

**return zip;**

**}**

**public void setZip(int zip) {**

**this.zip = zip;**

**}**

**public Address(String place, int zip) {**

**this.place = place;**

**this.zip = zip;**

**}**

**}**

**ObjectClone.java**

**public class** ObjectClone {

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

Person person=**new** Person(1,**"Surbhi"**,**new** Address(**"janakpuri"**,110046));

**try** {

*//using clone*

Person person2=(Person)person.clone();

System.***out***.println(**"After cloning using clone"**);

System.***out***.println(**"First Object "**+person);

System.***out***.println(**"Second Object "**+person2);

person.**address**.setPlace(**"tilak nagar"**);

System.***out***.println(**"Modification in an object in case of clone"**);*//shallow copy*

System.***out***.println(**"First Object "**+person);

System.***out***.println(**"Second Object "**+person2);

} **catch** (CloneNotSupportedException e) {

e.printStackTrace();

}

*//using constructor*

*//deep copy*

Person person2=**new** Person(person);

System.***out***.println(**"After cloning using copy constructor"**);

System.***out***.println(**"First Object "**+person);

System.***out***.println(**"Second Object "**+person2);

person.**address**.setPlace(**"subhash nagar"**);

System.***out***.println(**"Modification in an object in case of copy constructor"**);*//shallow copy*

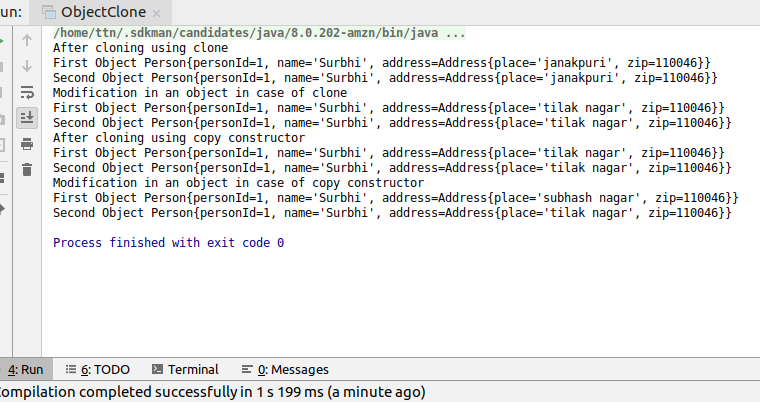
System.***out***.println(**"First Object "**+person);

System.***out***.println(**"Second Object "**+person2);

}

}

**Output**



**6. WAP showing try, multi-catch and finally blocks**.

**Solution**

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.util.Scanner;

**public class** TryCatch {

**static int** divide(**int** numerator,**int** denominator)

{

**return** numerator/denominator;

}

**static** File openFile(**final** String path)

{

**return new** File(path);

}

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

**try** {

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

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

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

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

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

in.nextLine();

System.***out***.println(**"Enter file path"**);

String path=in.nextLine();

System.***out***.println(*divide*(numerator,denominator));

File result=*openFile*(path);

FileReader reader=**new** FileReader(result);

}

**catch** (ArithmeticException arithmeticException)

{

arithmeticException.printStackTrace();

}

**catch** (FileNotFoundException fileNotFoundException)

{

System.***out***.println(**"Exception"**+fileNotFoundException.getMessage());

}

**catch** (Exception e) {

e.printStackTrace();

} **finally** {

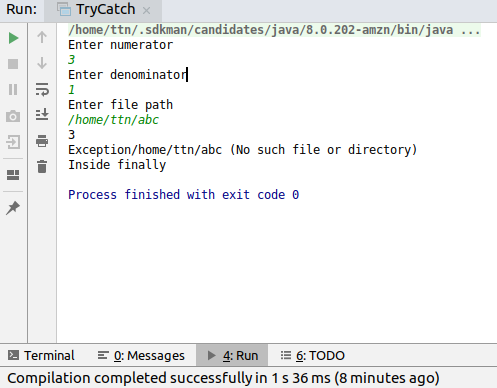
System.***out***.println(**"Inside finally"**);

}

}

}

**Output**



**7. WAP to convert seconds into days, hours, minutes and seconds.**

**Solution**

import java.util.Scanner;

public class ConvertSeconds {

public static void main(String[] args) {

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

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

int numberOfSeconds=in.nextInt();

int numberOfDays=numberOfSeconds/(24\*3600); *//1hour-->60mins-->60seconds--->3600seconds*

int numberOfHours=((numberOfSeconds%(24\*3600))/3600);

int numberofMinutes=(((numberOfSeconds%(24\*3600))%3600)/60);

int numberOfSecond=(((numberOfSeconds%(24\*3600))%3600)%60);

System.*out*.println(numberOfSeconds+" is equivalent to:- ");

System.*out*.println("Days: "+numberOfDays);

System.*out*.println("Hours: "+numberOfHours);

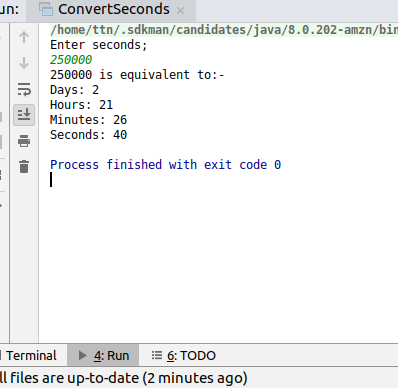
System.*out*.println("Minutes: "+numberofMinutes);

System.*out*.println("Seconds: "+numberOfSecond);

}

}

**Output**



**8. WAP to read words from the keyboard until the word done is entered. For each word except done, report whether its first character is equal to its last character. For the required loop, use a**

**a)while statement**

**b)do-while statement**

**Solution**

**import java.util.ArrayList;**

**import java.util.HashMap;**

**import java.util.Scanner;**

**public class ReadUntilDone {**

**static Scanner *in*;**

***//part-a***

**static HashMap<String,Boolean> readUntilDoneUsingWhile()**

**{**

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

**String inputString=null;**

**HashMap<String,Boolean>inputsReceived=new HashMap<String, Boolean>();**

**while(!(("done").equals(inputString)))**

**{**

**System.*out*.println("Enter a string");**

**inputString=*in*.next();**

**if(!inputString.equals("done"))**

**inputsReceived.put(inputString,(inputString.charAt(0)==inputString.charAt(inputString.length()-1)));**

**}**

**return inputsReceived;**

**}**

***//part-b***

**static HashMap<String,Boolean> readUntilDoneUsingDoWhile()**

**{**

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

**String inputString;**

**HashMap<String,Boolean>inputsReceived=new HashMap<String, Boolean>();**

**do**

**{**

**System.*out*.println("Enter a string");**

**inputString=*in*.next();**

**if(!inputString.equals("done"))**

**inputsReceived.put(inputString,(inputString.charAt(0)==inputString.charAt(inputString.length()-1)));**

**}while (!(inputString.equals("done")));**

**return inputsReceived;**

**}**

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

**HashMap<String,Boolean>inputCheckMap=*readUntilDoneUsingWhile*();**

**System.*out*.println("Items read using while");**

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

**inputCheckMap=*readUntilDoneUsingDoWhile*();**

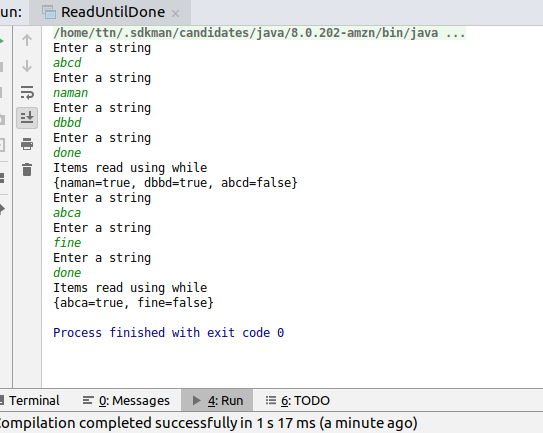
**System.*out*.println("Items read using while");**

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

**}**

**}**

**Output**



**9. Design classes having attributes for furniture where there are wooden chairs and tables, metal chairs and tables. There are stress and fire tests for each products.**

**Solution**

**Furniture.java**

**package** furniture;

**public abstract class** Furniture {

**private** String **color**;

**private float price**;

**private** furniture.Material **material**;

**public** Furniture(String color, **float** price,furniture.Material material) {

**this**.**color** = color;

**this**.**price** = price;

**this**.**material**=material;

}

**public abstract void** fireTest();

**public** String getColor() {

**return color**;

}

**public void** setColor(String color) {

**this**.**color** = color;

}

**public float** getPrice() {

**return price**;

}

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

**this**.**price** = price;

}

**public** furniture.Material getMaterial() {

**return material**;

}

**public void** setMaterial(furniture.Material material) {

**this**.**material** = material;

}

**public abstract void** stressTest();

}

**Chair.java**

**package** furniture;

**public class** Chair **extends** Furniture {

**boolean isReclinable**;

**boolean isFoldable**;

**public** Chair(String color, **float** price, furniture.Material material,**boolean** isReclinable,**boolean** isFoldable) {

**super**(color, price, material);

**this**.**isReclinable**=isReclinable;

**this**.**isFoldable**=isFoldable;

}

@Override

**public void** fireTest() {

System.***out***.println(**"Material of chair is:"**+**this**.getMaterial().getMaterialName()+**"Under fire it behaves like"**+**this**.getMaterial().getBehaviorFire());

}

@Override

**public void** stressTest() {

System.***out***.println(**"Material of chair is: "**+**this**.getMaterial().getMaterialName()+**"Under stress it behaves like"**+**this**.getMaterial().getBehaviorStress());

}

}

**Table.java**

**package furniture;**

**public class Table extends Furniture {**

**int numberOfLegs;**

**boolean hasWheels;**

**public Table(String color, float price, Material material, int numberOfLegs, boolean hasWheels) {**

**super(color, price, material);**

**this.numberOfLegs = numberOfLegs;**

**this.hasWheels = hasWheels;**

**}**

**@Override**

**public void fireTest() {**

**System.*out*.println("Material of table is:"+this.getMaterial().getMaterialName()+"Under fire it behaves like"+this.getMaterial().getBehaviorFire());**

**}**

**@Override**

**public void stressTest() {**

**System.*out*.println("Material of table is: "+this.getMaterial().getMaterialName()+"Under stress it behaves like"+this.getMaterial().getBehaviorStress());**

**}**

**}**

**Material.java**

**package furniture;**

***/\*This class is made so that even if new material of chair or table comes in picture***

***\* ,We don't need to create a concrete class for each type,***

***\* We just need to define the material and it's behavior under stress and fire and the code will run for***

***\* that type too.***

***\****

***\* \*/***

**public class Material {**

**private String materialName;**

**private String behaviorFire;**

**private String behaviorStress;**

**public Material(String materialName, String behaviorFire, String behaviorStress) {**

**this.materialName = materialName;**

**this.behaviorFire = behaviorFire;**

**this.behaviorStress = behaviorStress;**

**}**

**public String getMaterialName() {**

**return materialName;**

**}**

**public void setMaterialName(String materialName) {**

**this.materialName = materialName;**

**}**

**public String getBehaviorFire() {**

**return behaviorFire;**

**}**

**public void setBehaviorFire(String behaviorFire) {**

**this.behaviorFire = behaviorFire;**

**}**

**public String getBehaviorStress() {**

**return behaviorStress;**

**}**

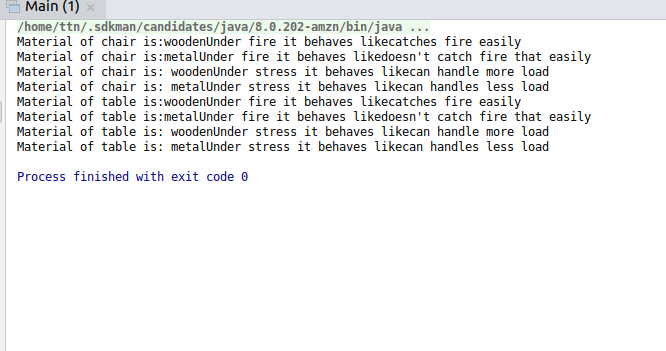
**public void setBehaviorStress(String behaviorStress) {**

**this.behaviorStress = behaviorStress;**

**}**

**}**

**Output**



**10. Design classes having attributes and method(only skeleton) for a coffee shop. There are three different actors in our scenario and i have listed the different actions they do also below**

**\* Customer**

**- Pays the cash to the cashier and places his order, get a token number back**

**- Waits for the intimation that order for his token is ready**

**- Upon intimation/notification he collects the coffee and enjoys his drink**

**( Assumption: Customer waits till the coffee is done, he wont timeout and cancel the order. Customer always likes the drink served. Exceptions like he not liking his coffee, he getting wrong coffee are not considered to keep the design simple.)**

**\* Cashier**

**- Takes an order and payment from the customer**

**- Upon payment, creates an order and places it into the order queue**

**- Intimates the customer that he has to wait for his token and gives him his token**

**( Assumption: Token returned to the customer is the order id. Order queue is unlimited. With a simple modification, we can design for a limited queue size)**

**\* Barista**

**- Gets the next order from the queue**

**- Prepares the coffee**

**- Places the coffee in the completed order queue**

**- Places a notification that order for token is ready**

**Solution**

**package com.ttn.barista;**

**public class Barista {**

**public void prepareAndProcessOrder()**

**{**

**System.*out*.println("1Call the getNextOrder fromPendingOrderQueue");**

**System.*out*.println("2. Iterate through list of products in that order and instruct the counter to make");**

**System.*out*.println("3.Notify customer with that orderId");**

**System.*out*.println("4. Place the order to completedQueue");**

**}**

**public void addOrderToCompletedQueue(Order order)**

**{**

**System.*out*.println("1. will be called by prepareAndProcessOrder");**

**System.*out*.println("2. Add Order to completedorderQueue");**

**}**

**public void notifyOrderReady(Order order)**

**{**

**System.*out*.println("will be called by prepareAndProcessOrder");**

**System.*out*.println("It will notify customer using the required mechanism");**

**}**

**}**

**package com.ttn.barista;**

**import java.util.List;**

**public class Cashier {**

**int employeeId;**

**String name;**

**public void collectPayment(float cash)**

**{**

**System.*out*.println("This method will be called in take order, after computing the total price of order" +**

**"Made this as collectPayment(), so that if any other mechanism than cash comes, we can process accordingly");**

**}**

**public int takeOrder(List<Product> productList)**

**{**

**int tokenNumber=0;*//any random token number generator logic here***

**System.*out*.println("This method will sum the price of all products and calls collectPayment by calculating sum" +**

**"and generate a toke number for that order");**

**System.*out*.println("this will also call placeOrderToPendingQueue with by creating an Order object");**

**return tokenNumber;**

**}**

**public void placeOrderToPendingQueue(Order order)**

**{**

**System.*out*.println("This method will add the order to a pending order queue" +**

**"by calling a method addToPendingOrdersof class PendingOrderQueue");**

**}**

**}**

**package com.ttn.barista;**

**import java.util.Queue;**

**public class CompleteOrderQueue {**

**static Queue<Order>*completeQueue*;**

**public void addToQueue(Order order)**

**{**

**System.*out*.println("As soon a this is called, we can store the order.");**

**}**

**}**

**package com.ttn.barista;**

**public class Customer {**

**private String name;**

**private long phoneNumber;**

**public Customer(String name, long phoneNumber) {**

**this.name = name;**

**this.phoneNumber = phoneNumber;**

**}**

**public String getName() {**

**return name;**

**}**

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

**this.name = name;**

**}**

**public long getPhoneNumber() {**

**return phoneNumber;**

**}**

**public void setPhoneNumber(long phoneNumber) {**

**this.phoneNumber = phoneNumber;**

**}**

**public void waitForToken() {**

**System.*out*.println("This method will keep customer waiting,until the token number is in completed queue,ie, barista raises notification" +**

**"after that signal the customer with chosen mechanism");**

**}**

**public void collectCofee()**

**{**

**System.*out*.println("After the customer collects coffee, the order is stored in order table");**

**}**

**}**

**package com.ttn.barista;**

**import java.util.List;**

**public class Order {**

**int tokenId;**

**List<Product> productList;**

**public Order(int tokenId, List<Product> productList) {**

**this.tokenId = tokenId;**

**this.productList = productList;**

**}**

**public int getTokenId() {**

**return tokenId;**

**}**

**public void setTokenId(int tokenId) {**

**this.tokenId = tokenId;**

**}**

**public List<Product> getProductList() {**

**return productList;**

**}**

**public void setProductList(List<Product> productList) {**

**this.productList = productList;**

**}**

**}**

**package com.ttn.barista;**

**import java.util.ArrayList;**

**import java.util.List;**

**import java.util.Queue;**

**public class PendingOrderQueue {**

**static Queue<Order>*pendingOrders*;**

**static void addToPendingOrders(Order order)**

**{**

**System.*out*.println("This will add the order to the queue using queue add method");**

**}**

**static Order getNextOrder()**

**{**

**System.*out*.println("this will dequeue element from queue and returns next order");**

**return new Order(1, new ArrayList<Product>());**

**}**

**}**

**package com.ttn.barista;**

***//This class can further be modified if we give customer chances to customize the order, like an extra topping***

**public class Product {**

**String name;**

**float price;**

**public Product(String name, float price) {**

**this.name = name;**

**this.price = price;**

**}**

**public String getName() {**

**return name;**

**}**

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

**this.name = name;**

**}**

**public float getPrice() {**

**return price;**

**}**

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

**this.price = price;**

**}**

**}**

**11. Convert the following code so that it uses nested while statements instead of for statements:**

**int s = 0;**

**int t = 1;**

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

**{**

**s = s + i;**

**for (int j = i; j > 0; j−−)**

**{**

**t = t \* (j - i);**

**}**

**s = s \* t;**

**System.out.println("T is " + t);**

**}**

**System.out.println("S is " + s);**

**Solution**

**public class WhileNotFor {**

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

**int s=0,t=1;**

**int i=0,j;**

**while(i<10)**

**{**

**s+=i;**

**j=i;**

**while(j>0)**

**{**

**t\*=(j-1);**

**j--;**

**}**

**s\*=t;**

**System.*out*.println("T is"+t);**

**i++;**

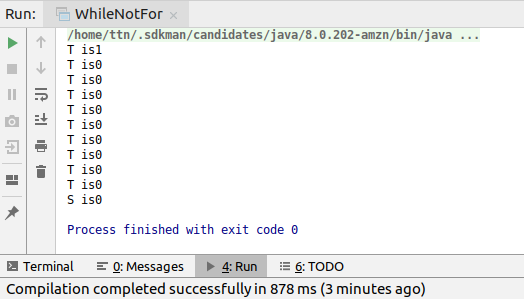
**}**

**System.*out*.println("S is"+s);**

**}**

**}**

**Output**



**12.What will be the output on new Child(); ?**

**class Parent extends Grandparent {**

**{**

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

**}**

**public Parent() {**

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

**}**

**static {**

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

**}**

**}**

**class Grandparent {**

**static {**

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

**}**

**{**

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

**}**

**public Grandparent() {**

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

**}**

**}**

**class Child extends Parent {**

**public Child() {**

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

**}**

**static {**

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

**}**

**{**

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

**}**

**}**

**Output**

**This will be the output:-**

static - grandparent

static - parent

static - child

instance - grandparent

constructor - grandparent

instance - parent

constructor - parent

instance - child

constructor - child

**Q13. Create a custom exception that do not have any stack trace.**

**Solution**

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

**public** CustomException(String message)

{

**super**(message);

}

@Override

**public synchronized** Throwable fillInStackTrace() {

**return this**;

}

}

**public class** Main {

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

**try** {

**throw new** CustomException(**"My Exception"**);

} **catch** (CustomException e) {

e.printStackTrace();

}

}

}

**Output**

