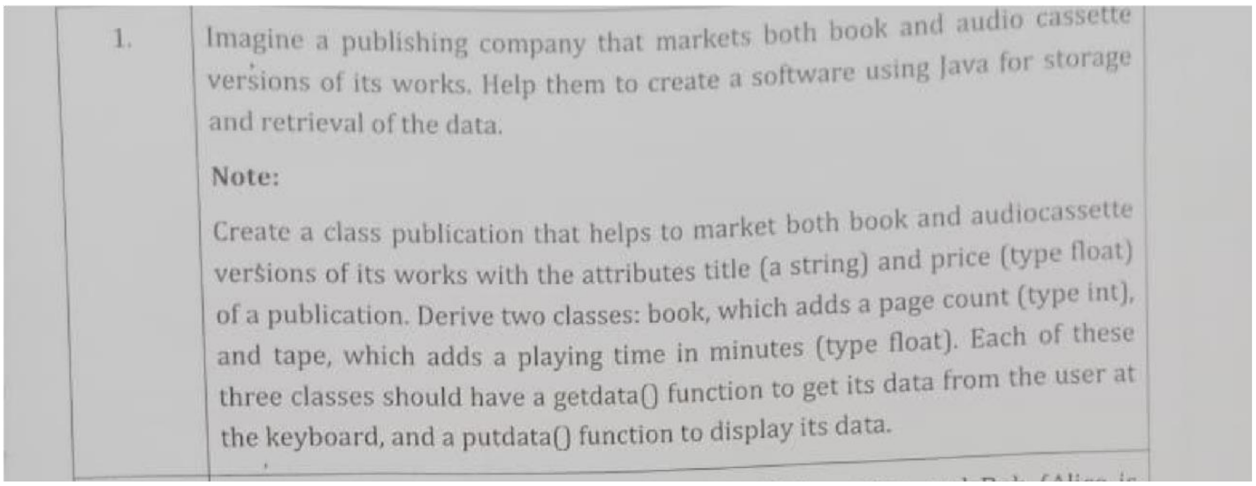
1.

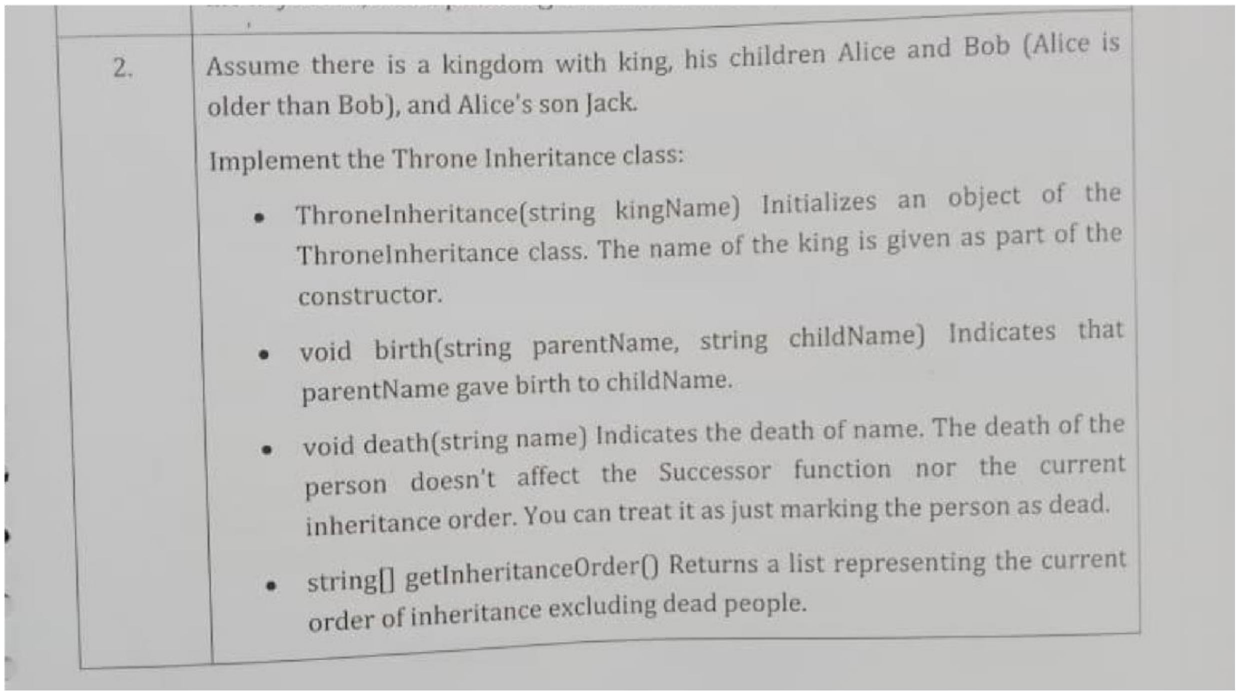


1. Imagine a publishing company that markets both book and audio cassette versions of its works. Help them to create a software using Java for storage and retrieval of the data.

Note:

Create a class publication that helps to market both book and audiocassette versions of its works with the attributes title (a string) and price (type float) of a publication. Derive two classes: book, which adds a page count (type int), and tape, which adds a playing time in minutes (type float). Each of these three classes should have a getdata() function to get its data from the user at the keyboard, and a putdata() function to display its data.

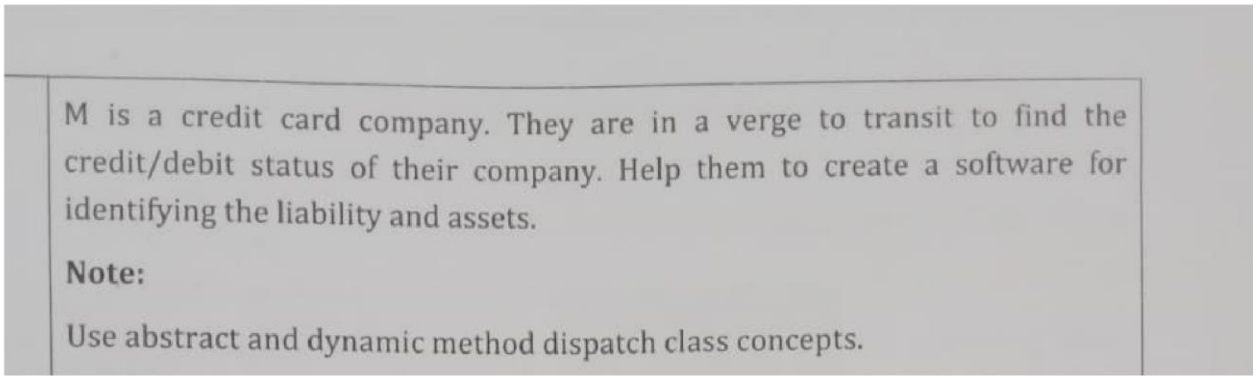
```
import java.util.Scanner;

class Publication
  { String title;
  float price;

  void getdata() {
     Scanner scanner = new Scanner(System.in);
     System.out.print("Enter title: ");
     title = scanner.nextLine();
     System.out.print("Enter price: ");
     price = scanner.nextFloat();
  }

  void putdata() {
     System.out.println("Title: " + title + "\nPrice: " + price);
  }
}

class Book extends Publication {
  int pageCount;

  void getdata() {
     super.getdata();
     System.out.print("Enter page count: ");
     pageCount = new Scanner(System.in).nextInt();
  }

  void putdata() {
        super.putdata();
        System.out.println("Page Count: " + pageCount);
    }
}

class Tape extends Publication {
    float playingTime;

    void getdata() {
        super.getdata();
        System.out.print("Enter playing time (minutes): ");
        playingTime = new Scanner(System.in).nextFloat();
    }

    void putdata() {
        super.putdata();
        System.out.println("Playing Time: " + playingTime);
    }
}

public class Main {
    public static void main(String[] args) {
        Book book1 = new Book();
        book1.getdata();
        book1.putdata();

        Tape tape1 = new Tape();
        tape1.getdata();
        tape1.putdata();
    }
}
```
2.



2.	Assume there is a kingdom with king, his children Alice and Bob (Alice is older than Bob), and Alice's son Jack. Implement the Throne Inheritance class: • ThroneInheritance(string kingName) Initializes an object of the ThroneInheritance class. The name of the king is given as part of the constructor. • void birth(string parentName, string childName) Indicates that parentName gave birth to childName. • void death(string name) Indicates the death of name. The death of the person doesn't affect the Successor function nor the current inheritance order. You can treat it as just marking the person as dead. • string[] getInheritanceOrder() Returns a list representing the current order of inheritance excluding dead people.

```
import java.util.*;

class ThroneInheritance {
   String king;
   Map<String, List<String>> family = new HashMap<>();
   Map<String, Boolean> deathStatus = new HashMap<>();

   ThroneInheritance(String kingName) {
      king = kingName;
   }

   void birth(String parentName, String childName) {
      family.computeIfAbsent(parentName, k -> new ArrayList<>()).add(childName);
   }

   void death(String name) {
      deathStatus.put(name, true);
   }

   List<String> getInheritanceOrder() {
      List<String> result = new ArrayList<>();
      dfs(king, result); return result;
   }
   void dfs(String person, List<String> result) {
      if (!deathStatus.getOrDefault(person, false)) {
          result.add(person);
      }
      family.getOrDefault(person, Collections.empty
```

3.

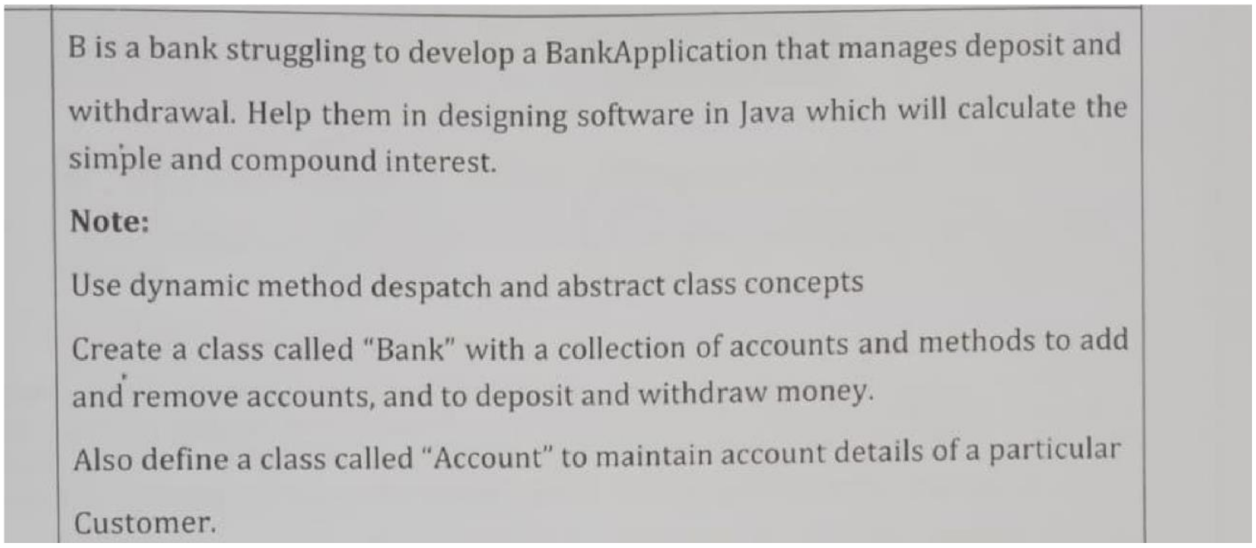

M is a credit card company. They are in a verge to transit to find the credit/debit status of their company. Help them to create a software for identifying the liability and assets.

Note:

Use abstract and dynamic method dispatch class concepts.

```
abstract class Account {
    String accountType;

    public Account(String type) {
        accountType = type;
    }

    abstract void displayStatus();
}

class Liability extends Account {
    public Liability(String type) {
        super(type);
    }

    @Override void
    displayStatus() {
        System.out.println("Liability Account Type: " + accountType);
    }
}

class Asset extends Account {
    public Asset(String type) {
        super(type);
    }

    @Override
    void displayStatus() {
        System.out.println("Asset Account Type: " + accountType);
    }
}

public class CreditCardCompany {
    public static void main(String[] args) {
        Account liabilityAccount = new Liability("Credit Card Debt");
        Account assetAccount = new Asset("Investments");

        liabilityAccount.displayStatus();
        assetAccount.displayStatus();
    }
}
```

4.



B is a bank struggling to develop a BankApplication that manages deposit and withdrawal. Help them in designing software in Java which will calculate the simple and compound interest.

Note:

Use dynamic method despatch and abstract class concepts

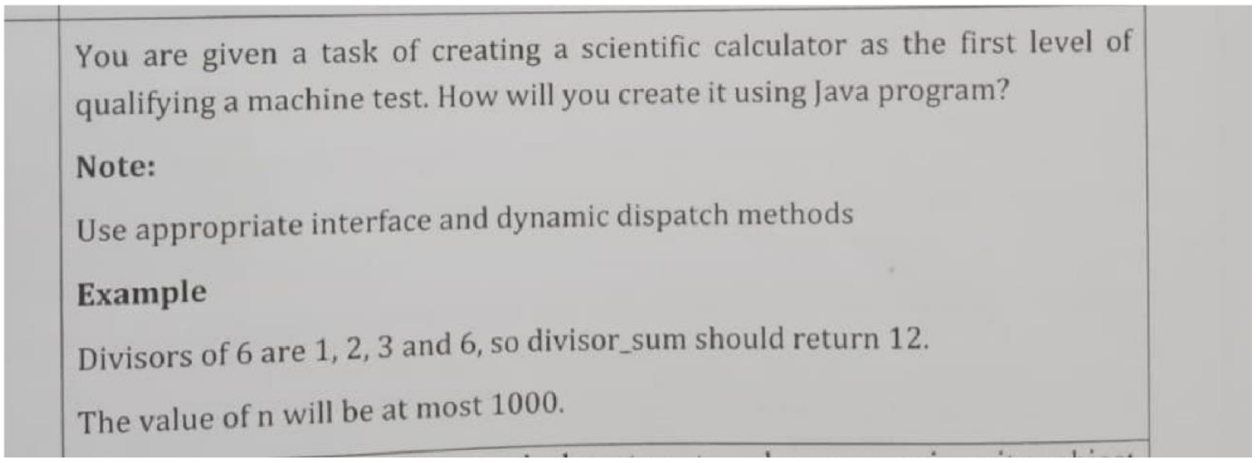
Create a class called "Bank" with a collection of accounts and methods to add and remove accounts, and to deposit and withdraw money.

Also define a class called "Account" to maintain account details of a particular Customer.

```
import java.util.ArrayList;

abstract class Account {
    protected String accountHolder;
    protected double balance;

    public Account(String accountHolder, double initialBalance) {
        this.accountHolder = accountHolder;
        this.balance = initialBalance;
    }
    abstract void deposit(double amount);
    abstract void withdraw(double amount); abstract void

    calculateSimpleInterest(double rate, int years);

    abstract void calculateCompoundInterest(double rate, int years);
}

class SavingsAccount extends Account { public SavingsAccount(String
    accountHolder, double initialBalance) {
        super(accountHolder, initialBalance);
    }

    @Override void
    deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: " + amount);
    }

    @Override void
    withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        } else {
            System.out.println("Insufficient funds");
        }
    }

    @Override void calculateSimpleInterest(double
    rate, int years) { double interest = balance * rate *
    years / 100;
        System.out.println("Simple Interest: " + interest);
    }

    @Override void calculateCompoundInterest(double
    rate, int years) {
        double compoundInterest = balance * Math.pow((1 + rate / 100), years) - balance;
        System.out.println("Compound Interest: " + compoundInterest);
    }
}
```

```
class Bank {
   private ArrayList<Account> accounts = new ArrayList<>();
   public void addAccount(Account account) {
      accounts.add(account);
   }

   public void removeAccount(Account account) {
      accounts.remove(account);
   }
}

public class BankApplication { public
   static void main(String[] args) {
      Bank bank = new Bank();

      Account savingsAccount = new SavingsAccount("John Doe", 1000.0);
      bank.addAccount(savingsAccount);

      savingsAccount.deposit(500.0);
      savingsAccount.withdraw(200.0);
      savingsAccount.calculateSimpleInterest(5.0, 3);
      savingsAccount.calculateCompoundInterest(5.0, 3);
   }
}
```

5.



You are given a task of creating a scientific calculator as the first level of qualifying a machine test. How will you create it using Java program?

Note:

Use appropriate interface and dynamic dispatch methods

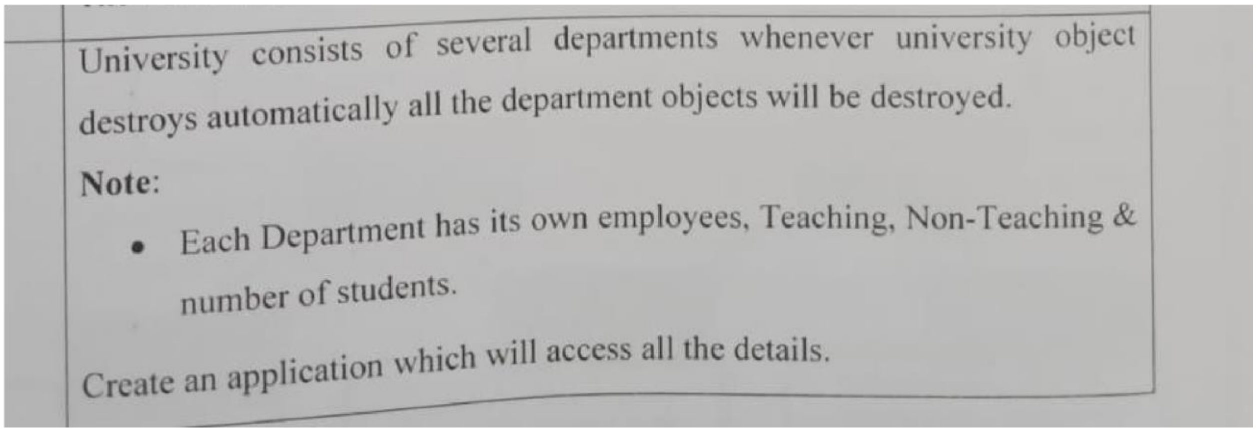
Example

Divisors of 6 are 1, 2, 3 and 6, so divisor_sum should return 12.

The value of n will be at most 1000.

```
interface Calculator {
    double add(double a, double b);
    double subtract(double a, double b);
    double multiply(double a, double b);
    double divide(double a, double b);
    int divisorSum(int n);
}

class BasicCalculator implements Calculator {
    public double add(double a, double b) { return a + b; } public double
    subtract(double a, double b) { return a - b; } public double multiply(double a,
    double b) { return a * b; } public double divide(double a, double b) { return b
    != 0 ? a / b : Double.NaN; } public int divisorSum(int n) {
        int sum = 0; for (int i = 1; i <= n; i++) if (n % i == 0) sum += i; return sum;
    }
}

public class ScientificCalculator {
    public static void main(String[] args) {
        Calculator calculator = new BasicCalculator();
        System.out.println("Add: " + calculator.add(5, 3));
        System.out.println("Divide: " + calculator.divide(5, 3));
        System.out.println("Divisor Sum: " + calculator.divisorSum(6));
    }
}
```

6.


University consists of several departments whenever university object destroys automatically all the department objects will be destroyed.

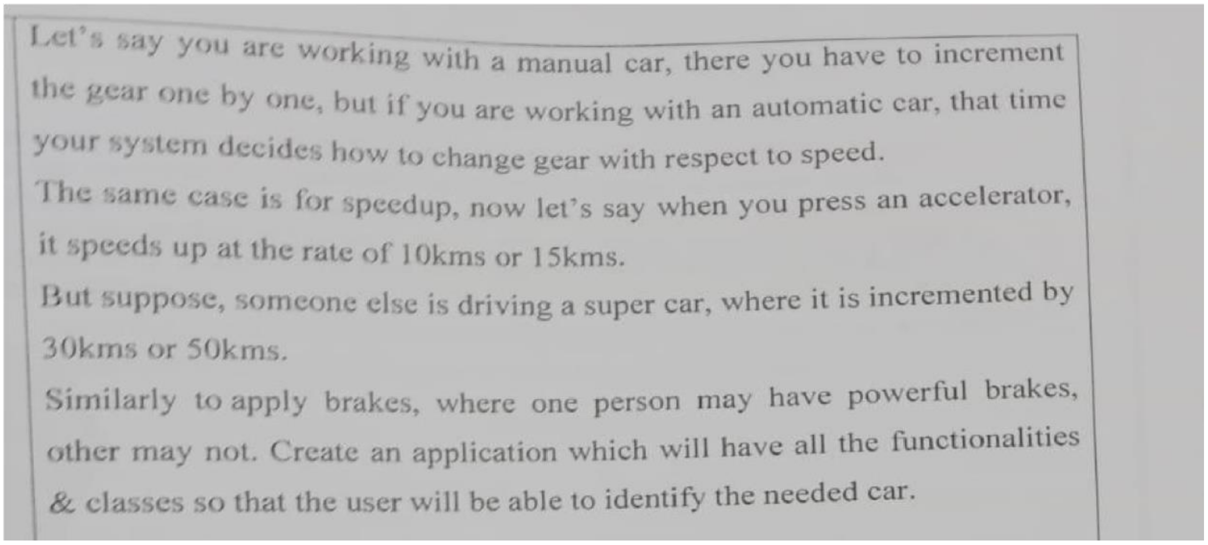
Note:

- Each Department has its own employees, Teaching, Non-Teaching & number of students.

Create an application which will access all the details.

```
class University {
   static class Department {
       int teachingStaff; int
       nonTeachingStaff; int
       students;

       Department(int teaching, int nonTeaching, int students) {
           teachingStaff = teaching;
           nonTeachingStaff = nonTeaching;
           this.students = students;
       }
   }

   public static void main(String[] args) {
       Department computerScience = new Department(20, 10, 500);
       // Add more departments as needed

       // Accessing details
       System.out.println("Teaching Staff in Computer Science: " +
computerScience.teachingStaff);
       System.out.println("Non-Teaching Staff in Computer Science: " +
computerScience.nonTeachingStaff);
       System.out.println("Students in Computer Science: " + computerScience.students);
   }
}
```

7.

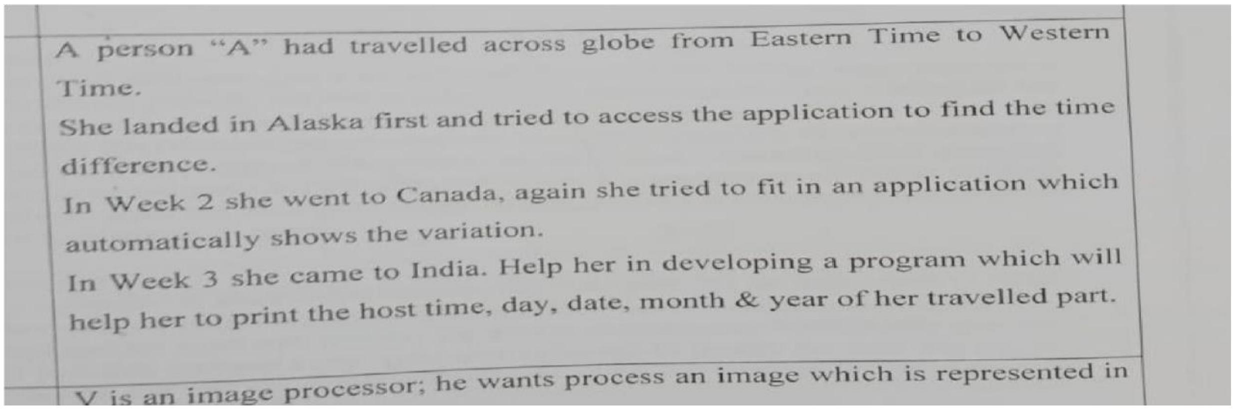


Let's say you are working with a manual car, there you have to increment the gear one by one, but if you are working with an automatic car, that time your system decides how to change gear with respect to speed.
The same case is for speedup, now let's say when you press an accelerator, it speeds up at the rate of 10kms or 15kms.
But suppose, someone else is driving a super car, where it is incremented by 30kms or 50kms.
Similarly to apply brakes, where one person may have powerful brakes, other may not. Create an application which will have all the functionalities & classes so that the user will be able to identify the needed car.

```
class Car {
   int gear; double speed; public

   Car(int gear, double speed) {

   this.gear = gear; this.speed =

   speed;

   }

   public void changeGear(int newGear) {
      this.gear = newGear;
   }

   public void speedUp() {
      this.speed += (gear == 1) ? 10 : 15;
```

```
    }

    public void applyBrakes() {
        this.speed -= (gear == 1) ? 10 : 15;
    }
}

class SuperCar extends Car { public
    SuperCar(int gear, double speed) {
        super(gear, speed);
    }

    @Override public void
    speedUp() {
        this.speed += (gear == 1) ? 30 : 50;
    }
}

public class CarApplication {
    public static void main(String[] args) {
        Car manualCar = new Car(1, 0);
        manualCar.speedUp();
        manualCar.applyBrakes();

        SuperCar superCar = new SuperCar(1, 0);
        superCar.speedUp();
        superCar.applyBrakes();
    }
}
```

8.



A person "A" had travelled across globe from Eastern Time to Western Time.
She landed in Alaska first and tried to access the application to find the time difference.
In Week 2 she went to Canada, again she tried to fit in an application which automatically shows the variation.
In Week 3 she came to India. Help her in developing a program which will help her to print the host time, day, date, month & year of her travelled part.

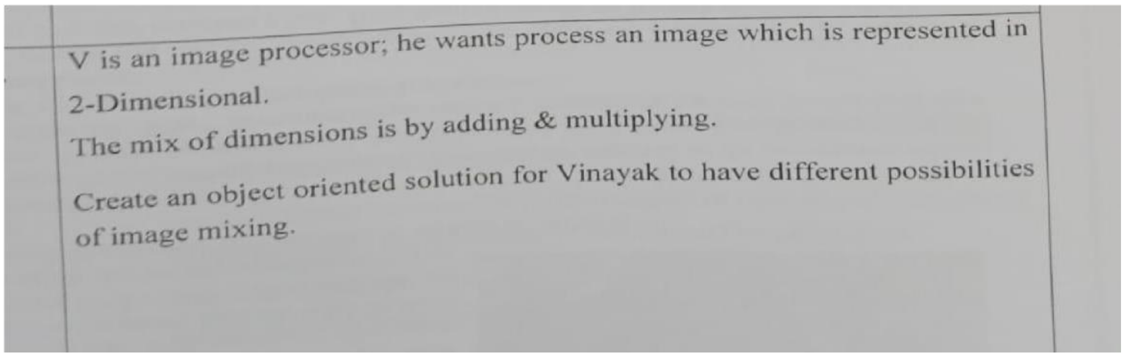
V is an image processor; he wants process an image which is represented in

```
import java.time.ZoneId; import
java.time.ZonedDateTime;

public class TravelApplication {
   public static void main(String[] args) {
      printTimeAndDate("Alaska");
      printTimeAndDate("Canada");
      printTimeAndDate("India");
   }

   public static void printTimeAndDate(String location) {
      ZonedDateTime zonedDateTime = ZonedDateTime.now(ZoneId.of(location));
      System.out.println("Time in " + location + ": " + zonedDateTime);
   }
}
```

OR

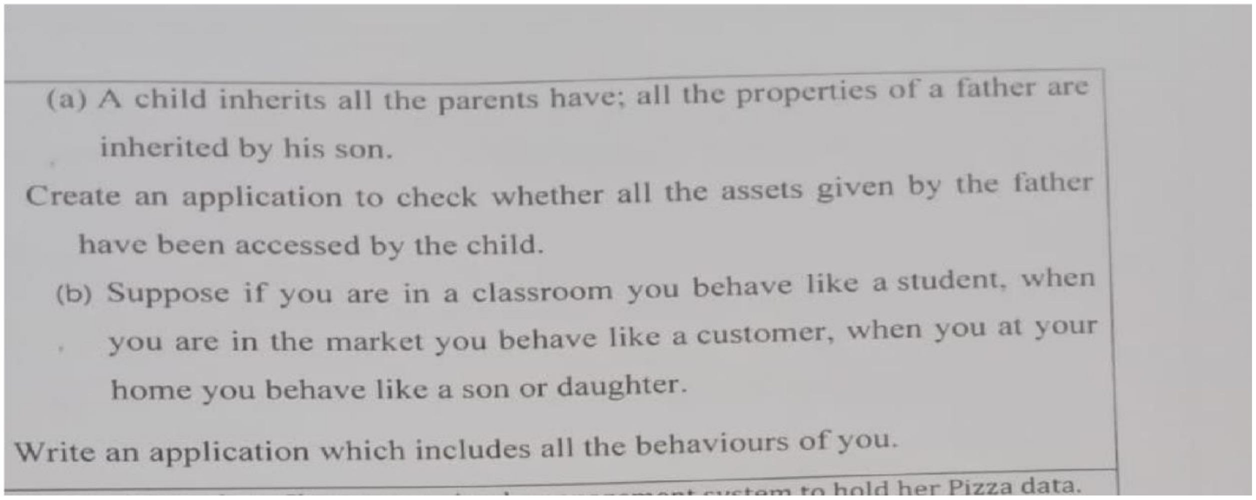
```
import java.time.LocalDateTime; import
java.time.ZoneId; import
java.time.format.DateTimeFormatter;
import java.util.Locale;

public class TravelTimeInfo { public static void
   printTravelInfo(String location) {
      // Get the current time in the specified time zone
      LocalDateTime localDateTime = LocalDateTime.now(ZoneId.of(location));

      // Format the date and time
      DateTimeFormatter formatter = DateTimeFormatter.ofPattern("EEEE, dd MMMM yyyy
HH:mm:ss", Locale.ENGLISH);
      String formattedDateTime = localDateTime.format(formatter);
      // Print the information
      System.out.println("Location: " + location);
      System.out.println("Current Local Time: " + formattedDateTime);
      System.out.println();
   }

   public static void main(String[] args) { //
      Week 1: Alaska (Alaska Time Zone)
      printTravelInfo("America/Anchorage");

      // Week 2: Canada (Eastern Time Zone)
```

```
        printTravelInfo("America/Toronto");

        // Week 3: India (Indian Standard Time)

printTravelInfo("Asia/Kolkata"); } } 9.
```


V is an image processor; he wants process an image which is represented in 2-Dimensional.
The mix of dimensions is by adding & multiplying.
Create an object oriented solution for Vinayak to have different possibilities of image mixing.

```
import java.util.Arrays;

class ImageProcessor {
    int[][] image;

    public ImageProcessor(int[][] image) {
        this.image = image;
    }

    public void addImage(int[][] newImage) {
        for (int i = 0; i < image.length; i++) { for
        (int j = 0; j < image[i].length; j++) {
                image[i][j] += newImage[i][j];
            }
        }
    }

    public void multiplyImage(int factor) {
        for (int[] row : image) {
            Arrays.setAll(row, i -> row[i] * factor);
        }
    }
}

public class ImageProcessingApplication {
    public static void main(String[] args) {
        int[][] initialImage = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        ImageProcessor processor = new ImageProcessor(initialImage);

        int[][] newImage = {{2, 3, 4}, {5, 6, 7}, {8, 9, 10}};
        processor.addImage(newImage);

        processor.multiplyImage(2);
    }
}
```

10.



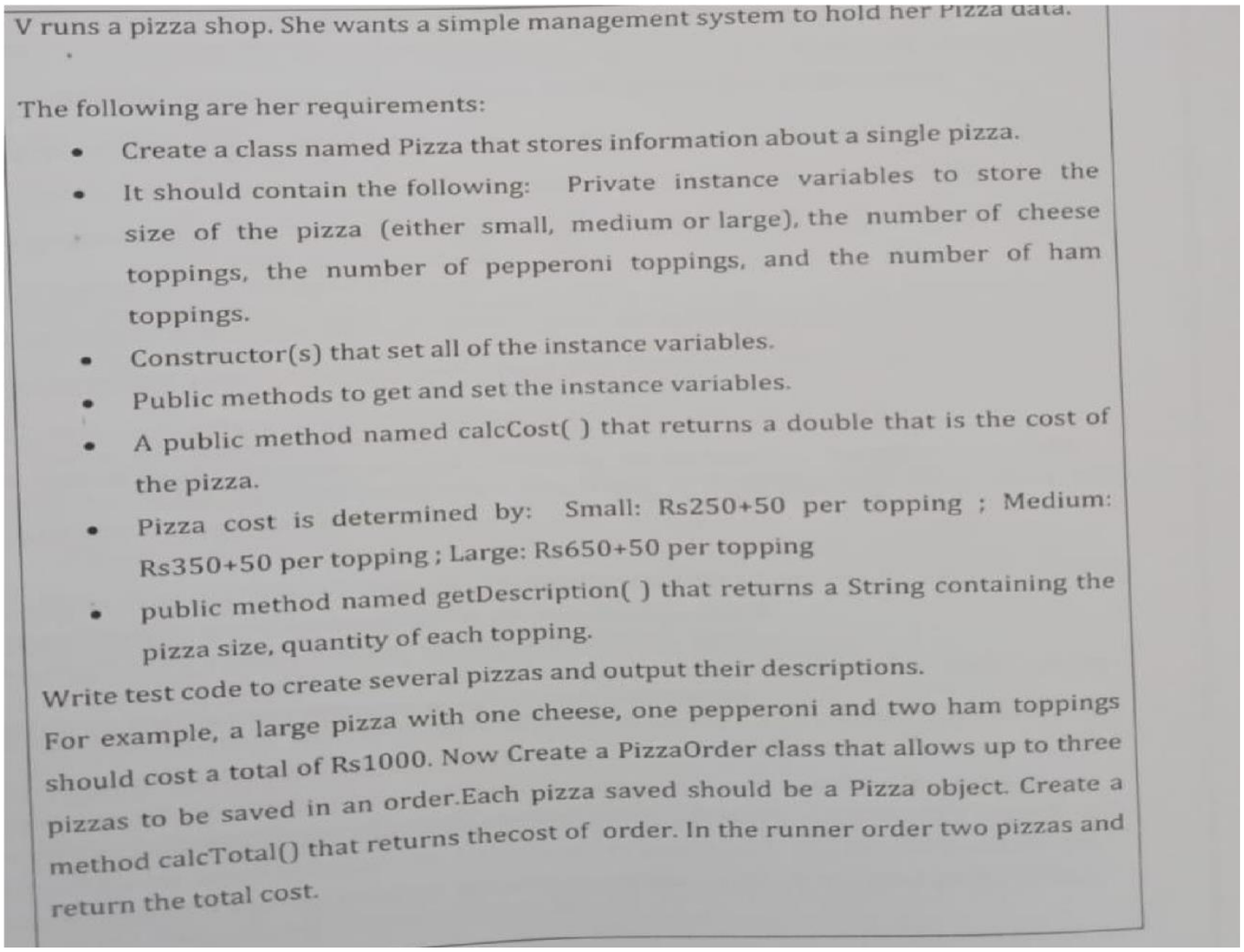
(a) A child inherits all the parents have; all the properties of a father are inherited by his son.

Create an application to check whether all the assets given by the father have been accessed by the child.

(b) Suppose if you are in a classroom you behave like a student, when you are in the market you behave like a customer, when you at your home you behave like a son or daughter.

Write an application which includes all the behaviours of you.

```
class Father {
    String[] assets = {"Car", "House", "Savings"};

    static class Son extends Father {
        void checkInheritedAssets() {
            for (String asset : assets) {
                System.out.println("Inherited Asset: " + asset);
            }
        }
    }

    public static void main(String[] args) {
        Son son = new Son();
        son.checkInheritedAssets();
    }
}

class Person {
    void behave(String location) {
        switch (location) {
            case "classroom":
                System.out.println("Behaving like a student");
                break;
            case "market":
                System.out.println("Behaving like a customer");
                break;
            case "home":
                System.out.println("Behaving like a son or daughter");
                break;
            default:
                System.out.println("Unknown behavior");
        }
    }

    public static void main(String[] args) {
        Person person = new Person();
        person.behave("classroom");
        person.behave("market");
        person.behave("home");
    }
}
```
11.



V runs a pizza shop. She wants a simple management system to hold her Pizza data.

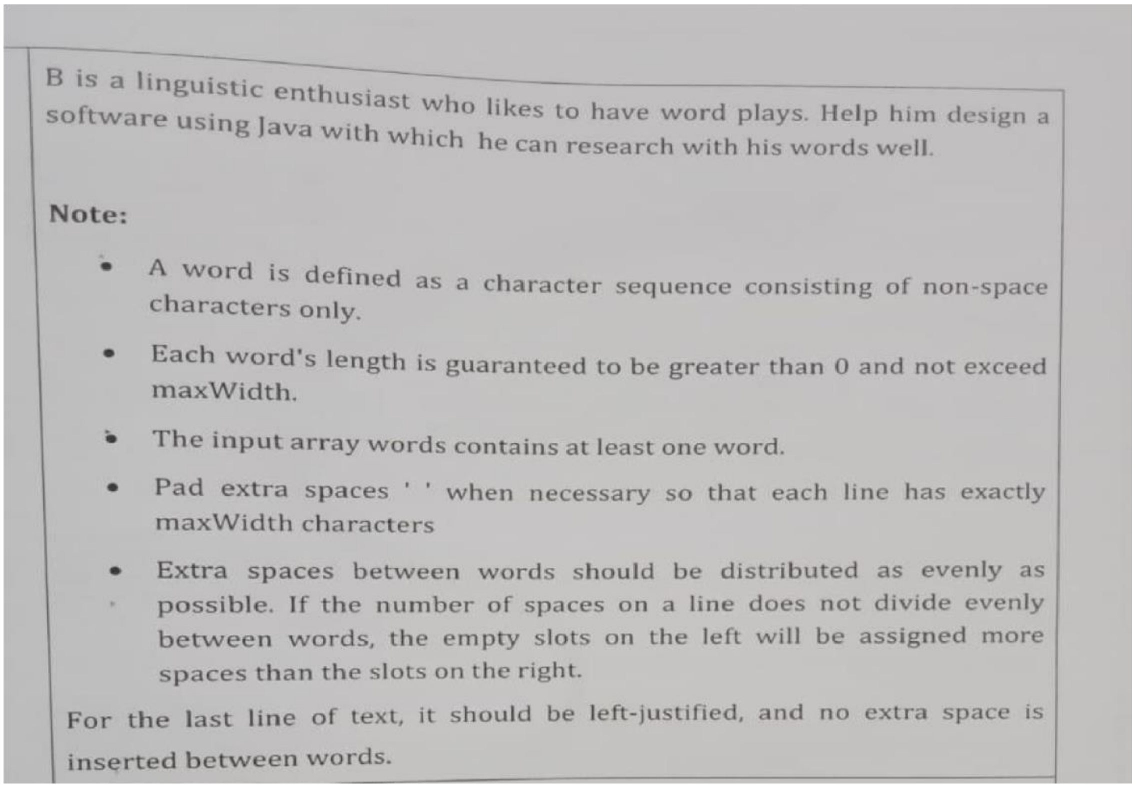
The following are her requirements:

- Create a class named Pizza that stores information about a single pizza.
- It should contain the following: Private instance variables to store the size of the pizza (either small, medium or large), the number of cheese toppings, the number of pepperoni toppings, and the number of ham toppings.
- Constructor(s) that set all of the instance variables.
- Public methods to get and set the instance variables.
- A public method named calcCost( ) that returns a double that is the cost of the pizza.
- Pizza cost is determined by: Small: Rs250+50 per topping ; Medium: Rs350+50 per topping ; Large: Rs650+50 per topping
- public method named getDescription( ) that returns a String containing the pizza size, quantity of each topping.

Write test code to create several pizzas and output their descriptions. For example, a large pizza with one cheese, one pepperoni and two ham toppings should cost a total of Rs1000. Now Create a PizzaOrder class that allows up to three pizzas to be saved in an order.Each pizza saved should be a Pizza object. Create a method calcTotal() that returns thecost of order. In the runner order two pizzas and return the total cost.

```
class Pizza {
   private String size; private int cheeseToppings,
   pepperoniToppings, hamToppings;

   public Pizza(String size, int cheese, int pepperoni, int ham) {
      this.size = size;
      cheeseToppings = cheese;
      pepperoniToppings =
      pepperoni; hamToppings =
      ham;
   }

   public double calcCost() {
      return (size.equals("Small") ? 250 : (size.equals("Medium") ? 350 : 650)) + 50 *
(cheeseToppings + pepperoniToppings + hamToppings);
   }

   public String getDescription() {
        return size + " Pizza with " + cheeseToppings + " cheese, " + pepperoniToppings + "
pepperoni, " + hamToppings + " ham.";
    }
}

class PizzaOrder {
    private Pizza[] pizzas = new Pizza[3];
    private int count = 0;

    public void addPizza(Pizza pizza) {
        if (count < 3) pizzas[count++] = pizza;
    }

    public double calcTotal() {
        double totalCost = 0; for (int i = 0; i < count; i++) totalCost
        += pizzas[i].calcCost(); return totalCost;
    }
}

public class PizzaShop {
    public static void main(String[] args) {
        PizzaOrder order = new PizzaOrder();
        order.addPizza(new Pizza("Large", 1, 1, 2));
        order.addPizza(new Pizza("Medium", 2, 0, 1));

        System.out.println("Total Cost: Rs " + order.calcTotal());
    }
}
```

12.


B is a linguistic enthusiast who likes to have word plays. Help him design a software using Java with which he can research with his words well.

Note:

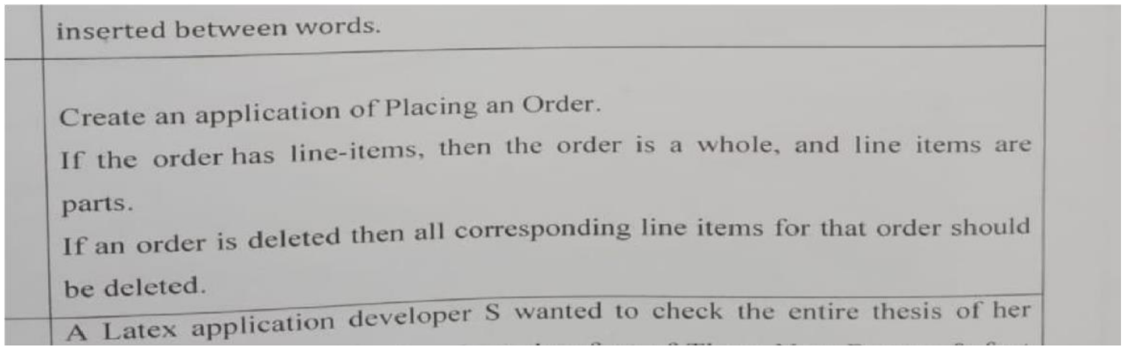
- A word is defined as a character sequence consisting of non-space characters only.
- Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
- The input array words contains at least one word.
- Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters
- Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left-justified, and no extra space is inserted between words.

```
import java.util.ArrayList;
import java.util.List;

class WordFormatter { public static List<String> fullJustify(String[]
   words, int maxWidth) { List<String> result = new ArrayList<>(); int
   start = 0, end;

      while (start < words.length) { int len = words[start].length(); for (end = start + 1; end <
         words.length && len + 1 + words[end].length() <= maxWidth;
end++) len += 1 + words[end].length();

         StringBuilder line = new StringBuilder(words[start]);
         int extraSpaces = maxWidth - len + (end - start - 1);

         for (int i = start + 1; i < end; i++) {
            line.append(" ".repeat(extraSpaces / (end - start - 1) + (i - start <= extraSpaces %
(end - start - 1) ? 1 : 0)))
                  .append(words[i]);
         }

         result.add(line.toString());

         start = end;
```

```
        }

        return result;
    }

    public static void main(String[] args) { String[] words
        = {"This", "is", "a", "sample", "text"}; int maxWidth
        = 10;
        fullJustify(words, maxWidth).forEach(System.out::println);
    }
}
```

13.



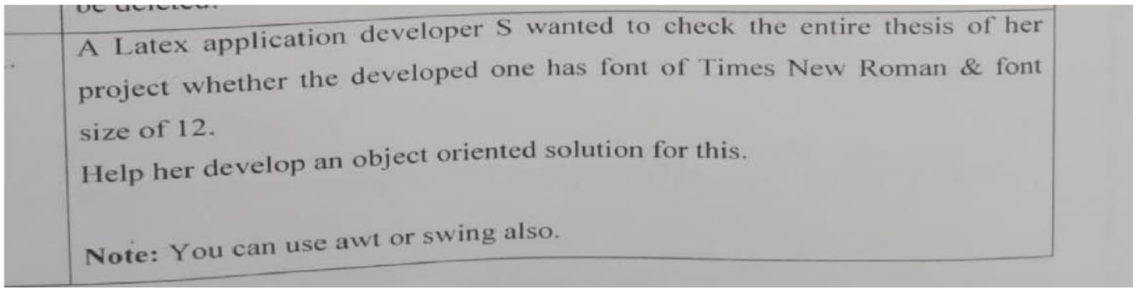
inserted between words.

Create an application of Placing an Order.
If the order has line-items, then the order is a whole, and line items are parts.
If an order is deleted then all corresponding line items for that order should be deleted.

A Latex application developer S wanted to check the entire thesis of her

```
import java.util.ArrayList;
import java.util.List;

class LineItem {
    int itemId;
    String itemName;

    public LineItem(int itemId, String itemName) {
        this.itemId = itemId;
        this.itemName = itemName;
    }
}

class Order {
    int orderId;
    List<LineItem> lineItems = new ArrayList<>();

    public Order(int orderId) {
        this.orderId = orderId;
    }
    public void addLineItem(LineItem lineItem) {
        lineItems.add(lineItem);
    }

    public void deleteOrder() {
        lineItems.clear();
    }
}

public class OrderApplication {
    public static void main(String[] args) {
        Order order1 = new Order(1);
        order1.addLineItem(new LineItem(101, "Item1"));
        order1.addLineItem(new LineItem(102, "Item2"));

        Order order2 = new Order(2);
        order2.addLineItem(new LineItem(201, "Item3"));

        order1.deleteOrder(); // Deleting order1 along with its line items
    }
}
```

14.

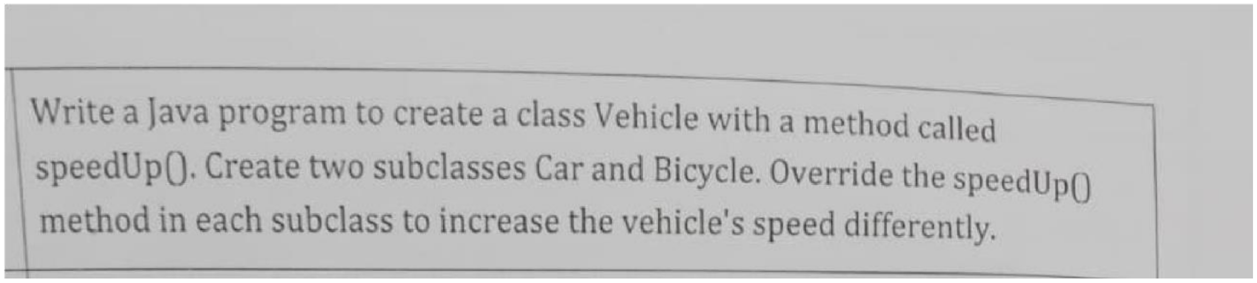


A Latex application developer S wanted to check the entire thesis of her project whether the developed one has font of Times New Roman & font size of 12.
Help her develop an object oriented solution for this.

Note: You can use awt or swing also.

```
import java.awt.Font;

class ThesisChecker {
    Font thesisFont;

    public ThesisChecker(Font thesisFont) {
        this.thesisFont = thesisFont;
    }

    public boolean isFontValid() {
        return thesisFont.getFamily().equalsIgnoreCase("Times New Roman") &&
thesisFont.getSize() == 12;
    }
}

public class LatexApplication {
    public static void main(String[] args) {
        Font thesisFont = new Font("Times New Roman", Font.PLAIN, 12);
        ThesisChecker thesisChecker = new ThesisChecker(thesisFont);

        if (thesisChecker.isFontValid()) {
            System.out.println("Thesis font is Times New Roman with font size 12.");
        } else {
            System.out.println("Thesis font is not as required.");
        }
    }
}
```

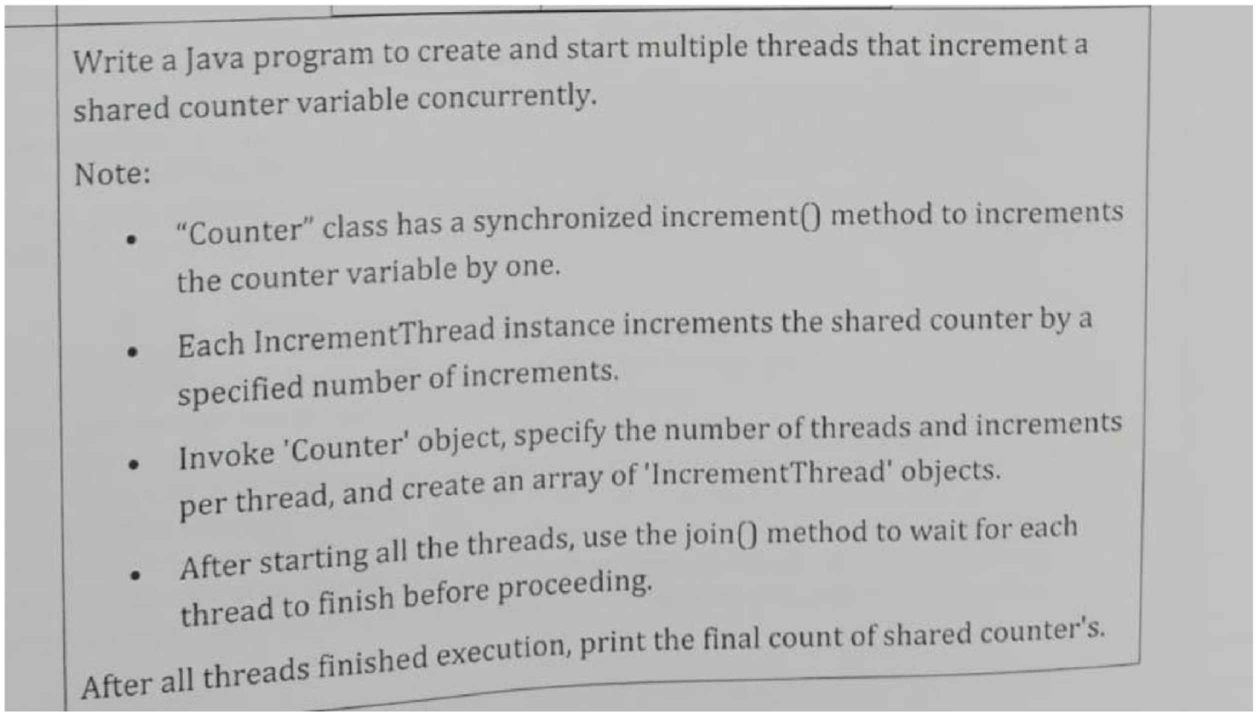
15.



Write a Java program to create a class Vehicle with a method called speedUp(). Create two subclasses Car and Bicycle. Override the speedUp() method in each subclass to increase the vehicle's speed differently.

```
class Vehicle {
    void speedUp() {
        System.out.println("Vehicle is speeding up.");
    }
}

class Car extends Vehicle
    { @Override void
    speedUp() {
        System.out.println("Car is accelerating.");
    }
}

class Bicycle extends Vehicle {
    @Override void
    speedUp() {
        System.out.println("Bicycle is pedaling faster.");
    }
}
public class VehicleApplication { public
    static void main(String[] args) {
    Vehicle vehicle = new Vehicle();
    vehicle.speedUp();

        Car car = new Car();
        car.speedUp();

        Bicycle bicycle = new Bicycle();
        bicycle.speedUp();
    }
}
```

16.



Write a Java program to create and start multiple threads that increment a shared counter variable concurrently.

Note:

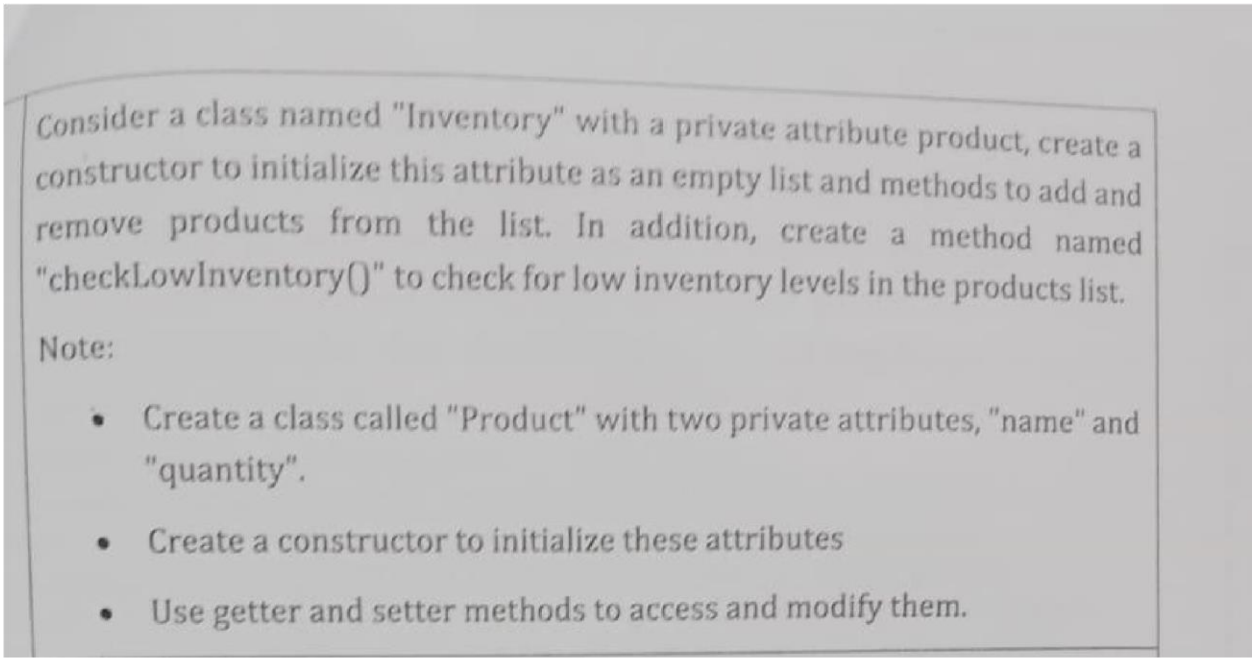
- "Counter" class has a synchronized increment() method to increments the counter variable by one.
- Each IncrementThread instance increments the shared counter by a specified number of increments.
- Invoke 'Counter' object, specify the number of threads and increments per thread, and create an array of 'IncrementThread' objects.
- After starting all the threads, use the join() method to wait for each thread to finish before proceeding.

After all threads finished execution, print the final count of shared counter's.

```
class Counter {
    private int count = 0;

    synchronized void increment() {
        count++;
    }
```

```
   int getCount() {
      return count;
   }
}
class IncrementThread extends Thread {
   private Counter counter;
   private int increments;

   public IncrementThread(Counter counter, int increments) {
      this.counter = counter;
      this.increments = increments;
   }

   public void run() {
      for (int i = 0; i < increments; i++) {
         counter.increment();
      }
   }
}

public class ThreadExample {
   public static void main(String[] args) throws InterruptedException {
      Counter sharedCounter = new Counter(); int numThreads = 3;
      int incrementsPerThread = 1000;

      IncrementThread[] threads = new IncrementThread[numThreads];
      for (int i = 0; i < numThreads; i++) {
         threads[i] = new IncrementThread(sharedCounter, incrementsPerThread);
         threads[i].start();
      }

      for (IncrementThread thread : threads) {
         thread.join();
      }

      System.out.println("Final Count: " + sharedCounter.getCount());
   }
}
```
17.

}

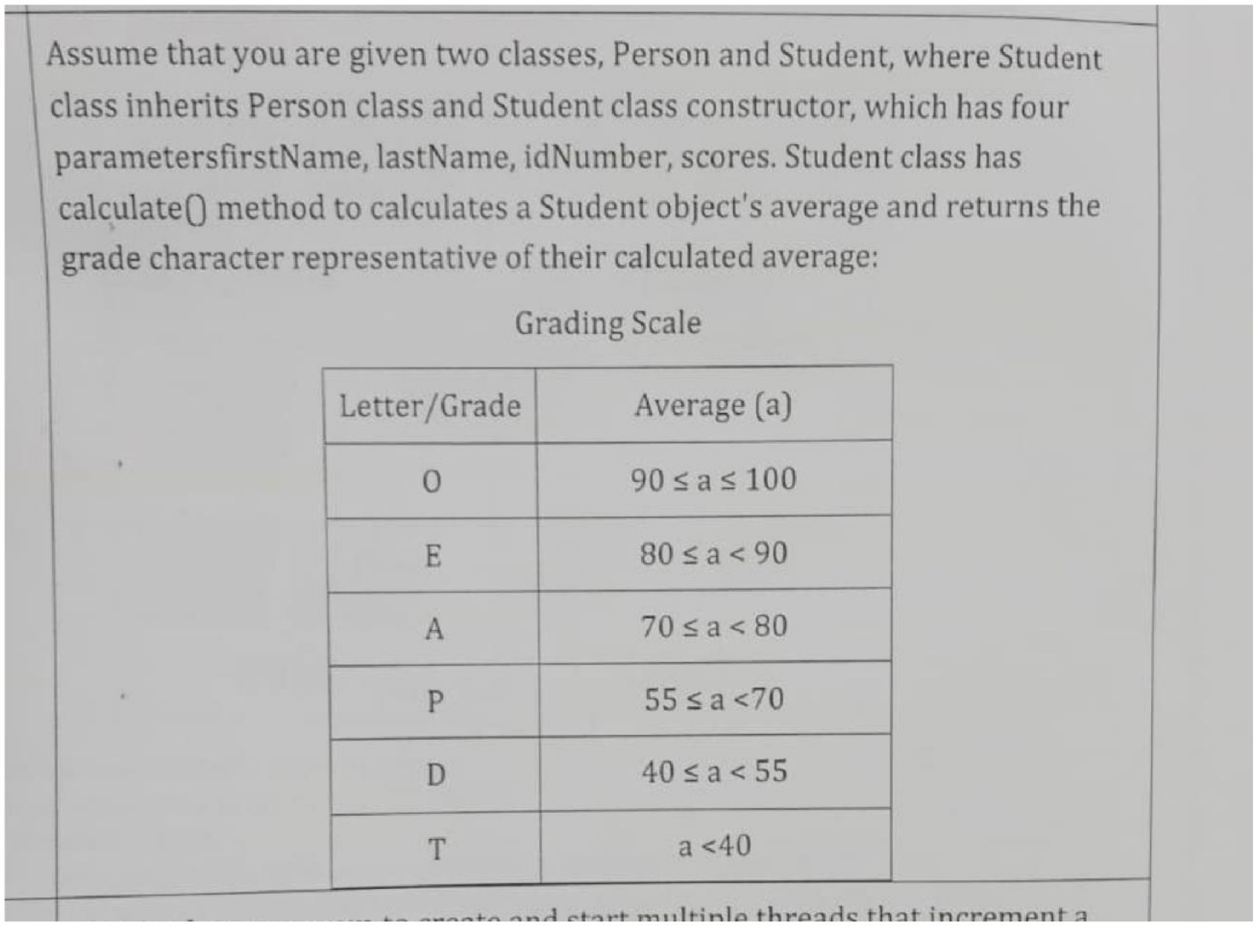

Consider a class named "Inventory" with a private attribute product, create a constructor to initialize this attribute as an empty list and methods to add and remove products from the list. In addition, create a method named "checkLowInventory()" to check for low inventory levels in the products list.

Note:

- Create a class called "Product" with two private attributes, "name" and "quantity".
- Create a constructor to initialize these attributes
- Use getter and setter methods to access and modify them.

```
import java.util.ArrayList;
import java.util.List;

class Product {
  private String name;
  private int quantity;

  public Product(String name, int quantity) {
    this.name = name;
    this.quantity = quantity;
  }

  // Getter and setter methods

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getQuantity() {
    return quantity;
  }
```

```
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}

class Inventory {
    private List<Product> products;

    public Inventory() {
        this.products = new ArrayList<>();
    }

    public void addProduct(Product product) {
        products.add(product);
    }

    public void removeProduct(Product product) {
        products.remove(product);
    }

    public void checkLowInventory() {
        for (Product product : products) {
            if (product.getQuantity() < 5) {
                System.out.println("Low inventory for product: " + product.getName());
            }
        }
    }
}

public class InventoryManagement {
    public static void main(String[] args) {
        Inventory inventory = new Inventory();
        Product product1 = new Product("Item1", 10);
        Product product2 = new Product("Item2", 3);

        inventory.addProduct(product1);
        inventory.addProduct(product2);

        inventory.checkLowInventory();
    }
}
```



Assume that you are given two classes, Person and Student, where Student class inherits Person class and Student class constructor, which has four parametersfirstName, lastName, idNumber, scores. Student class has calculate() method to calculates a Student object's average and returns the grade character representative of their calculated average:

Grading Scale

Letter/Grade	Average (a)
O	90 ≤ a ≤ 100
E	80 ≤ a < 90
A	70 ≤ a < 80
P	55 ≤ a <70
D	40 ≤ a < 55
T	a <40

to create and start multiple threads that increment a

```
class Person {

    protected String firstName;

    protected String lastName;

    protected int idNumber;



    // Person class code

}

class Student extends Person {

    protected int[] scores;
    public char calculate() {

        int sum = 0;

        for (int score : scores) {

            sum += score;

        }

        double average = (double) sum / scores.length;  // Use double for more precision



        if (average >= 90 && average <= 100) return 'O';

        else if (average >= 80) return 'E';

        else if (average >= 70) return 'A';

        else if (average >= 55) return 'P';

        else if (average >= 40) return 'D';

        else return 'T';

    }



    // Student class code

}

public class Main1 {

    public static void main(String[] args) {

        // Creating a student

        Student student = new Student();
        student.firstName = "John";

        student.lastName = "Doe";

        student.idNumber = 12345;



        // Assigning scores

        int[] scores = {80, 92, 75, 88, 95};

        student.scores = scores;



        // Calculating and printing the grade

        char grade = student.calculate();

        System.out.println("Student " + student.firstName + " " + student.lastName + " with ID " +
student.idNumber +

                " has an average grade of " + grade);

    }

}
```