King Saud University

Course Code:	CSC 113
Course Title:	**Computer Programming II**
Semester:	Fall 2020
Exercises Cover Sheet:	**Midterm 1 Exam**

Student Name:	
Student ID:	
Student Section No.	

Exercise 1

Give the output of the following program.

```java
public class Flight {
	private String flightNum;
	protected int distance;
	public Flight() {
		flightNum = "Initiated";
		distance = 2000;
		System.out.println ("The Flight "+ flightNum + " is Created");
	}
	public Flight (String flightNum, int dist) {
		this.flightNum = flightNum;
		this.distance=dist;
	}
	public void display() {
		System.out.println ("Flight number: " + flightNum + " distance: " +  distance );
	}
	public int cost () {		return 4000; }
}
```

```java
public class LongDistanceFlight extends Flight {
	protected int rate;
	public LongDistanceFlight () {	rate = 5;	}
	public LongDistanceFlight (String flightNum, int dist, int r) {
		super(flightNum, dist);
		rate = r;
	}
	public void display () {
		System.out.println ("Long Distance Flight ");
		super.display();
	}
	public int cost() {
		if (distance < 2000 )
			System.out.println ("Warning: Distance is Less Than 2000 Km");
		return (super.cost()+ distance * rate);
	}
}
```

```java
public class InternationalFlight extends LongDistanceFlight {
	protected int airportFees;
	public InternationalFlight(String flightNum, int distance, int rate, int fees) {
		super(flightNum,distance,rate);
		airportFees = fees;
	}
	public InternationalFlight(int fees) {
		airportFees = fees;
	}
	public void display()  {
		System.out.println ("International Flight ");
		super.display();
		System.out.println(this.cost());
	}
	public int cost() {
		return (super.cost() + airportFees);
	}
}
```

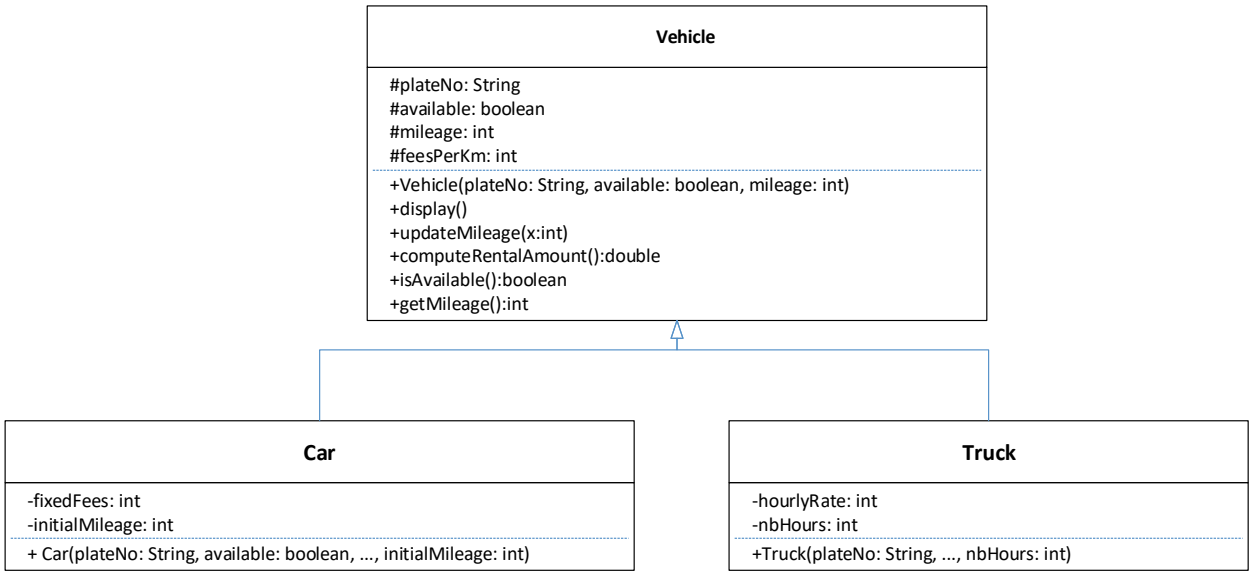
```java
public class Test {

	public static void main(String[] args) {
		Flight flight;

		flight  = new InternationalFlight("SV366", 1000, 3, 400);
		flight.display();

		flight  = new InternationalFlight(500);
	}
}
```

Answer:

```
International Flight
Long Distance Flight
Flight number: SV366 distance: 1000
Warning: Distance is Less Than 2000 Km
7400
The Flight Initiated is Created
```

Exercise 2

Consider the following UML class diagram:



Class **Vehicle**

- *ATTRIBUTES*:

  -plateNo: the plate number of the vehicle.

  -available: true if the vehicle is not rented.

  - mileage: total distance traveled in kms.

  - feesPerKm: rental amount of the vehicle per km.

- *METHODS*:

  - **Vehicle(plateNo: String, available:** boolean**, mileage:** int**)**: constructor
  - **display()**: this method displays all the attributes of the Vehicle object.
  - **updateMileage(x:** int**)**: this method updates the mileage of the Vehicle after each rental. It sets the mileage of the Vehicle to x.
  - **isAvailable():** this method returns the availability of the Vehicle.
  - **computeRentalAmount()** : computes and returns the **rental amount** of the vehicle as follows.
    - For the **Truck** : **rental amount** = (hourlyRate* nbHours)
    - For the **Car: rental amount** = feesPerKm * (mileage–initialMileage) + fixedFees

Class **Car**

- *ATTRIBUTES*:

  - **fixedFees:** the fixed rental fees of the Car.

- **initialMileage:** the mileage of the Car before being rented.

- *METHODS*:
  - **Car(plateNo: String, available:** boolean**, mileage:** int**, feesPerKm:** int**, fixedFees:** double**, initialMileage:** int**):** Constructor.

Class **Truck**

- *ATTRIBUTES*:
  - **hourlyRate:** the hourly rate of the Truck.
  - **nbHours:** the number of rental hours of the Truck.

- *METHODS*:
  - **Truck(plateNo: String, available:** boolean**, mileage:** int**, feesPerKm:** int**, hourlyRate:** double**, nbHours:** int**):** Constructor.

Question: Write using Java the class **Vehicle** and the class **Car**.

Answer Exercise 2: The Class **Vehicle**

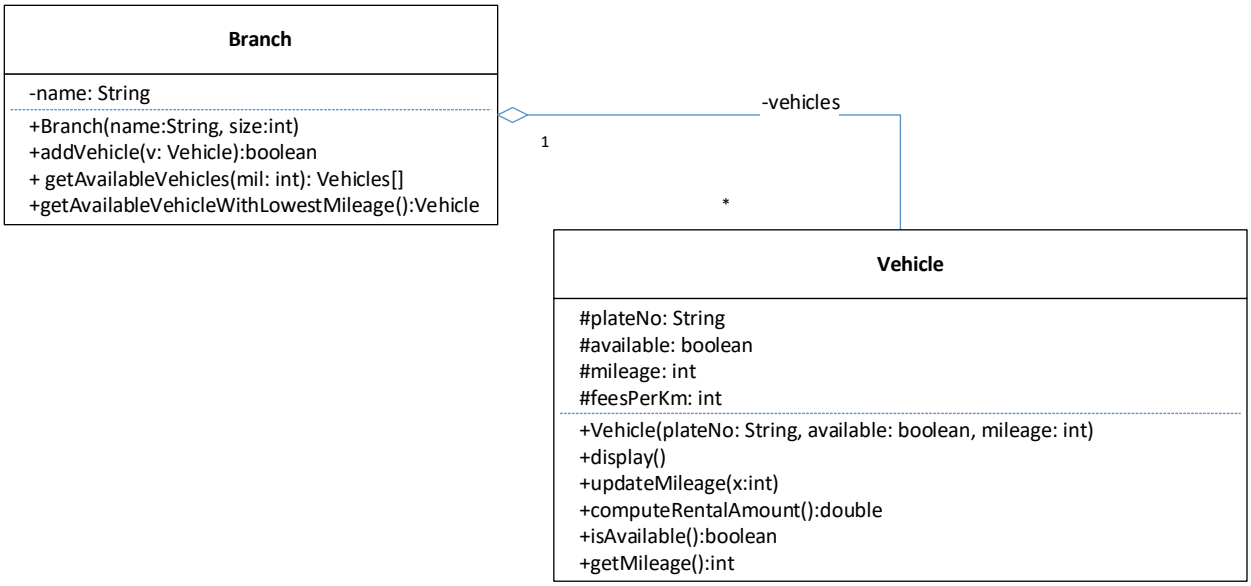
```java
package mid1Fall2020;

public abstract class Vehicle {
	protected String plateNumber;
	protected int mileage;
	protected int feesPerKm;
	protected boolean available ;

	public Vehicle(String plateNumber, boolean available, int mileage) {
		this.plateNumber = plateNumber;
		this.mileage = mileage;
		this.mileage = mileage;
	}

	public void display() {
		System.out.println("plateNumber: "+plateNumber);
		System.out.println("mileage: "+mileage);
		System.out.println("feesPerKm: "+feesPerKm);
		System.out.println("available: "+available);
	}

	public void updateMileage(int x) {
		if (! available) {
		mileage = x ;
		available = true ;
		}
	}

	public boolean isAvailable() {
		return available;
	}

	public abstract double computeRentalAmount();

}
```

Answer Exercise 2: The Class **Car**

```
package mid1Fall2020;

public class Car extends Vehicle{
      private int      fixedFees ;
      private int      initialMileage ;

      public Car(String plateNumber, boolean available, int mileage, int
fixedFees, int initialMileage) {
            super(plateNumber, available, mileage);
            this.fixedFees = fixedFees;
            this.initialMileage = initialMileage;
      }

      public double computeRentalAmount() {
            return   feesPerKm * (mileage - initialMileage) + fixedFees ;
      }

}
```

Exercise 3

Consider the following UML class diagram:



Class **Branch**

- *ATTRIBUTES*:

  - **name:** the name of the vehicle rental company branch.

- *METHODS*:

  - **Branch(name:** String**, size:** int**).** Constructor**.**

  - **addVehicle(v:** Vehicle**).** This method adds the Vehicle **v** to the Branch. If successfully added, the method returns true. It returns false otherwise.

  - **getAvailableVehicles(mil:** int**). r**eturns an array that contains ***all available*** (not rented) Vehicles having a mileage less than ***mil***.

  - **getAvailableVehicleWithLowestMileage().** This method returns the ***available*** Vehicle having the ***lowest mileage***.

<u>Question:</u> Write using Java the class **Branch**.

Answer Exercise 3: The Class **Branch**

```
public class Branch {
      private String  name;
      private Vehicle[] vehicles;
      private int    nbV;

      public Branch(String name, int size) {
            this.name = name;
            this.nbV = 0;
            vehicles = new Vehicle[size] ;
      }

      public void addVehicle(Vehicle v) {
            if (nbV<vehicles.length) {
                  vehicles[nbV++]= v ;
                  return;
            }
            System.out.println("no insert possible the branch is full");
      }

      public Vehicle[] getAvailableVehicles(int mil) {
            Vehicle[] availableVehicles = new Vehicle[nbV];
            int j = 0 ;
            for(int i =0 ; i<nbV;i++) {
                  if (vehicles[i].isAvailable() && vehicles[i].mileage<mil)
                        availableVehicles[j++]=vehicles[i];
            }
            return availableVehicles;
      }

      public Vehicle getAvailableVehiclesWithLowestMileag(int mil) {
            Vehicle availableVehicle = null;
            double minMileage = Double.MAX_VALUE;
            for(int i =0 ; i<nbV;i++) {
                  if (vehicles[i].isAvailable() &&
vehicles[i].mileage<minMileage) {
                        availableVehicle=vehicles[i];
                        minMileage = vehicles[i].mileage ;
                  }
            }
            return availableVehicle;
      }

}
```