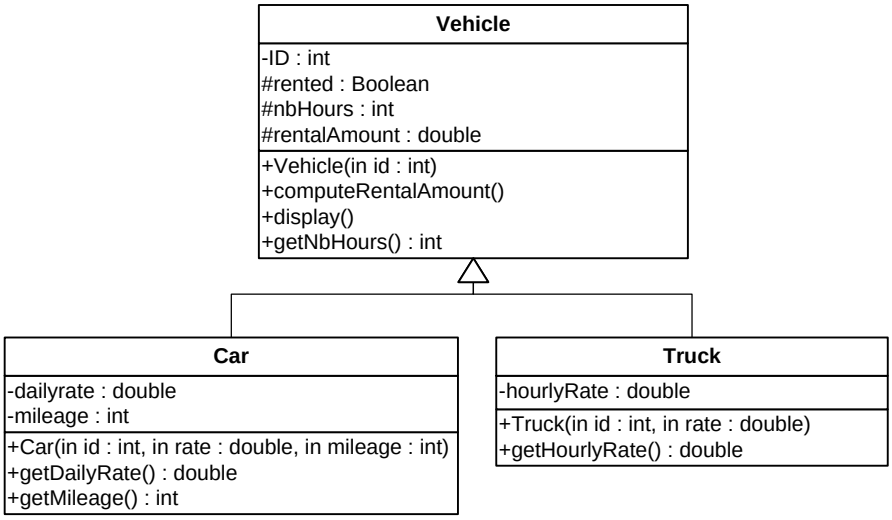
Name:	Section:	CSC113 Final	Time allowed: 3h
ID:	Dr.:	Spring 2013	

Exercise 1

Consider the following UML class diagram:



Class**Vehicle**

- *ATTRIBUTES*:

  -ID: the ID of the vehicle.

  -rented: true if the vehicle is rented (checked out).

  - nbHours: duration of the rental in hours.

  - rentalAmount: rental amount of the vehicle.

- *METHODS*:

  - **Vehicle(id:int)**: constructor
  - **display()**. this method displays all the attributes of the Vehicle object.
  - **computeRentalAmount()**. computes and sets the **rentalAmount** of the vehicle if the vehicle is rented.
    - For the **Truck** class, **rentalAmount** is: (hourlyRate* nbHours)
    - For the **Car** class, **rentalAmount** is: (dailyRate * (nbHours/24)+ mileage * 0.4)

Class **Car**

- *ATTRIBUTES*:
  - **dailyRate:** the daily rental rate.
  - **mileage:** the number of kilometers drove by the last renter.
- *METHODS*:
  - **Car(id:int, rate:double, mileage:int):** Constructor.
  - **getDailyRate().** Returns the daily rate of the Car.
  - **getMileage().** Returns the mileage of the Car.
  - **display().** This method displays all the attributes of the Car.

Class**Truck**

- *ATTRIBUTES*:
  - **hourlyRate:** the hourly rate of the Truck.
- *METHODS*:
  - **Truck(id:int, rate:double):** Constructor.
  - **getHourlyRate().** Returns the hourly rate of the Truck.
  - **display().** This method displays all the attributes of the Truck.

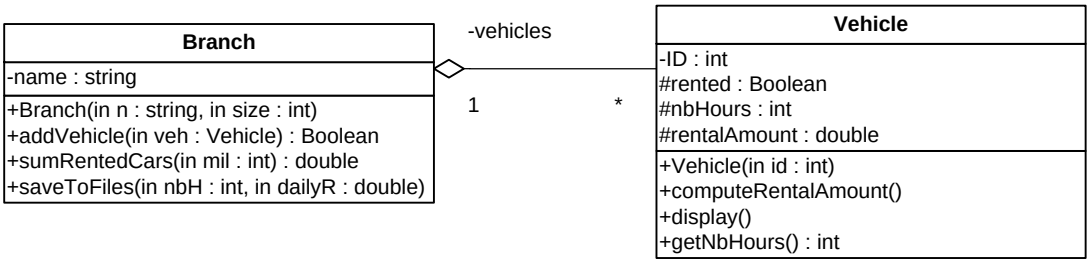
Question: Translate into Java code the classes**Vehicle** and **Car**.

Answer Exercise 1: The Class **Vehicle**

Answer Exercise 1: The Class **Car**

Exercise 2

Consider the following UML class diagram:



Class **Branch**

- *ATTRIBUTES*:

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

- *METHODS*:

  - **Branch(n: string, size: int).** Constructor **.** If the size is less or equal than zero, the constructor throws an exception with the following message "Invalid Size".

  - **addVehicle(veh: Vehicle).** Adds a vehicle to the branch. It returns true if the vehicle is added, false otherwise**.**

  - **sumRentedCars(mil: int).** Computes and returns the sum of the rental amounts of all rented cars having a mileage less than ***mil***.

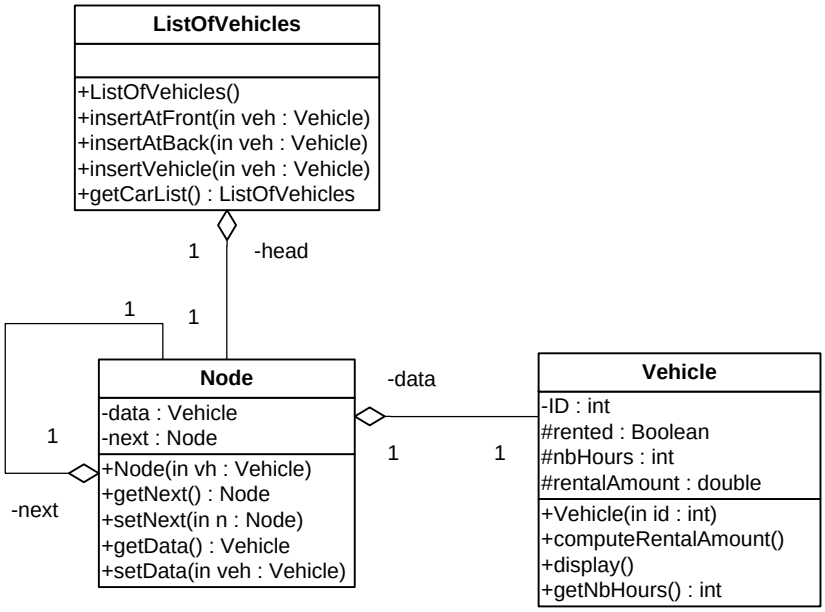
  - **saveToFile(nbH:int, dailyR:double).** This methods saves the vehicles of the branch as follows:
    -The **Car** objects with **dailyRate** equals to ***dailyR*** are saved in the file "cars.data".
    -The **Truck** objects with **nbHours** greater than ***nbH*** are saved in the file "trucks.data".

<u>Question:</u>Translate into Java code the class **Branch**.

Answer Exercise 2: The Class **Branch**

Exercise 3

Consider the following UML class diagram:



Class **ListOfVehicles**

- *ATTRIBUTES*:

  - **head:** head of the list.

- *METHODS*:

  - **ListOfVehicles(). Constructor**
  - **insertAtFront(veh:Vehicle).** Inserts the vehicle ***veh*** at the front of the current list.
  - **insertAtBack(veh:Vehicle).** Inserts the vehicle ***veh*** at the back of the current list.
  - **insertVehicle(veh:Vehicle).** Inserts the vehicle ***veh*** in the list as follows:
    - if ***veh*** is a Car, it is inserted at the front of the current list
    - if ***veh*** is a Truck, it is inserted at the back of the current list.
  - **getCarList():ListOfVehicles:** Returns a list containing all Car objects from the current list.
  - **saveToFile(nbH:int, dailyR:double).** This methods saves the vehicles of the list as follows:

    -The **Car** objects with **dailyRate** equals to ***dailyR*** are saved in the file "cars.data".

    -The **Truck** objects with **nbHours** greater than ***nbH*** are saved in the file "trucks.data".

Question: Translate into Java code the class **ListOfVehicles.**

Answer Exercise 3: The Class **ListOfVehicles**

8 pts

```
public abstract class Vehicle              ----------------   1
{
      private int ID;
      protected boolean rented;
      protected int nbHours;
      protected double rentalAmount;

      public Vehicle(int id)
      {
            ID = id;                 ----------------   1
            rented = false;
            nbHours = 0;
            rentalAmount = 0.00;
      }

      public int getNbHours()
      {
            return nbHours;         ----------------   1
      }

      public abstract void computeRentalAmount();  ----------------   1

      public void display()
      {
            System.out.println("ID: " + ID);     ----------------   1
            System.out.println("Rented: " + rented);     ---------- 1
            System.out.println("Nb Hours: " + nbHours);     ------- 1
            System.out.println("Rental Amount: " + rentalAmount);-- 1
      }
}
```

10 pts

```
public class Car extends Vehicle implements Serializable  ----------- 1
{
      private double dailyRate;
      private int mileage;

      public Car(int id, double rate, String mileage)
      {
            super(id);          ----------- 1
            dailyRate = rate;    ----------- 1
            this.mileage = mileage;     ----------- 1
      }

      public double getDailyRate()
      {
            return dailyRate;     ----------- 1
      }

      public int getMileage()
      {
            return mileage;      ----------- 1
      }
```

```
    public void computeRentalAmount()
    {
        rentalAmount = dailyRate * (nbHours / 24) + mileage * 0.4;  ------- 1
    }

    public void display()
    {
        super.display();                        ----------- 1
        System.out.println("Daily Rate: " + dailyRate);     ----------- 1
        System.out.println("Mileage: " + mileage);     ----------- 1
    }
}
```

30 pts

```
import java.io.*;


public class Branch
{
      private String name;
      private Vehicle vehicles[];   ----------- 1
      private int nb;   ----------- 1

      public Branch(String n, int size) throws Exception   ----------- 1
      {
            if(size <= 0)  ----------- 1
            {
                  throw new Exception("Invalid Size");   ----------- 1
            }
            name = n;    ----------- 1
            nb = 0;               ----------- 1
            vehicles = new Vehicle[size];   ----------- 1
      }

      public boolean addVehicle(Vehicle veh)
      {
            if(nb < vehicles.length)   ----------- 1
            {
                  vehicles[nb] = veh;    ----------- 1
                  nb++;     ----------- 1
                  return true;    ----------- 0.5
            }
            return false;   ----------- 0.5
      }

      public double sumRentedCars(int mil)
      {
            double sum = 0.00;    ----------- 1
            for(int i=0; i<nb; i++)   ----------- 1
            {
                  if(vehicles[i] instanceof Car   ----------- 1
                  && vehicles[i].rented   ----------- 1
                  && ((Car)vehicles[i]).getMileage() < mil ) ---------- 1
                  {
                        vehicles[i].computeRentalAmount();
                        sum += vehicles[i].getRentalAmount();   ----------- 1
                  }
            }

            return sum;       ----------- 1
      }
```

```
public void saveToFile(int nbH, double dailyR)
{
      Try   ----------- 1
      {
            File f1 = new File("cars.data");  ----------- 1
            FileOutputStream fos1 = new FileOutputStream(f1);  ---- 1
            ObjectOutputStream os1 = new ObjectOutputStream(fos1); -- 1

            File f2 = new File("trucks.data");
            FileOutputStream fos2 = new FileOutputStream(f2); ----------- 1
            ObjectOutputStream os2 = new ObjectOutputStream(fos2);


            for(int i=0; i<nb; i++)   ----------- 1
            {
                  if(vehicles[i] instanceof Car  ----------- 1
                  && ((Car)vehicles[i]).getDailyRate() == dailyR) --- 1
                  {
                        os1.writeObject(vehicles[i]);  ----------- 1
                  }
                  else if(vehicles[i].getNbHours() > nbH) ----------- 1
                  {
                        os2.writeObject(vehicles[i]);  ----------- 1
                  }
            }
      }
      catch(Exception e)
      {

      }
}


}
```

22 pts

```
public class ListOfVehicles
{
            private Node head;  ----------- 1

            public ListOfVehicles()
            {
                  head = null; ----------- 1
            }


          public void insertAtFront(Vehicle v)
          {
               Node n = new Node(v);  ----------- 1

               n.setNext(head);        ----------- 1
               head = n;        ----------- 1
          }

          public void insertAtBack(Vehicle v)
          {
             Node n = new Node(v);  ----------- 1
             if (head == null)  ----------- 1
             {
                 head = n;   ----------- 1
             }
             else
             {
               Node c = head;  ----------- 1
               while (c.getNext() != null)   ----------- 1
               {
                     c = c.getNext();   ----------- 1
               }
               c.setNext(n);   ----------- 1
             }
          }

          public void insertVehicle(Vehicle veh)
          {
                if(veh instanceof Car)   ----------- 1
                {
                      insertAtFront(veh);  ----------- 1
                }
                else
                {
                      insertAtBack(veh);  ----------- 1
                }
          }
```

```
public ListOfVehicles getCarList()
   {
         Node d = head;   ----------- 1
         ListOfVehicles list = new ListOfVehicles();  ----------- 1

         while (d != null)   ----------- 1
       {
               if (d.getData() instanceof Car)  ----------- 1
               {
                     list.insertAtBack(d.getData());  ----------- 1
               }
         d = d.getNext();   ----------- 1
       }
         return list;            ----------- 1
   }

   public void saveIntoFiles(int nbH, double dailyR)
   {
            File f1 = new File("cars.data");  ----------- 1
            FileOutputStream fos1 = new FileOutputStream(f1);  ---- 1
            ObjectOutputStream os1 = new ObjectOutputStream(fos1); -- 1

            File f2 = new File("trucks.data");
            FileOutputStream fos2 = new FileOutputStream(f2); ----------- 1
            ObjectOutputStream os2 = new ObjectOutputStream(fos2);

            Node d = head;   ----------- 1

            while (d != null)   ----------- 1
            {
                  if(d.getData() instanceof Car  ----------- 1
                  && ((Car)d.getData()).getDailyRate() == dailyR) --- 1
                  {
                        os1.writeObject(d.getData());  ----------- 1
                  }
                  else if(d.getData().getNbHours() > nbH) ----------- 1
                  {
                        os2.writeObject(d.getData());  ----------- 1
                  }

                  d = d.getNext();   ----------- 1
            }

            os1.close();
            os2.close();

   }


}
```