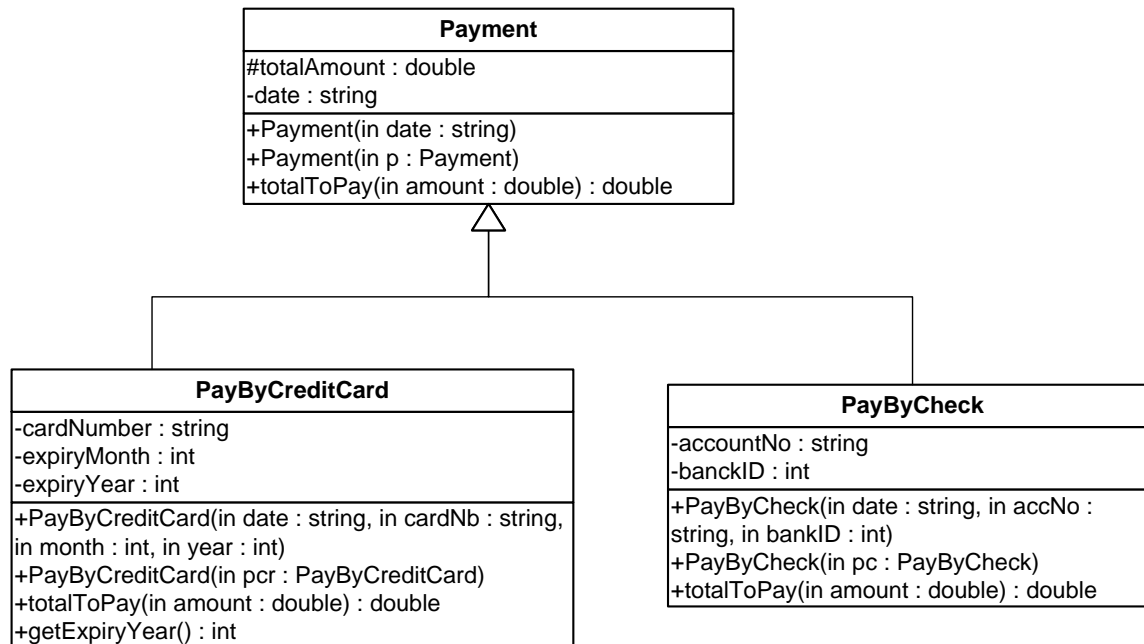


Exercise 1



Payment class:

- Attributes:
 - ***totalAmount***: the *total amount of the payment*. It is initially equal to 0.
 - ***date***: the date of the payment.
- METHODS:
 - ***Payment(date: String)***: constructor.
 - ***Payment(p: Payment)***: *copy* constructor.
 - ***totalToPay(amount: double): double***: this method receives an amount, calculates and returns the *total amount of the payment*. The *total amount of the payment* is computed as follows:
 - *for the payment by credit card*:
 - $total\ amount\ of\ the\ payment = amount - (amount * 0.05)$
 - *for the payment by check*:
 - $total\ amount\ of\ the\ payment = amount + (amount * 0.10)$

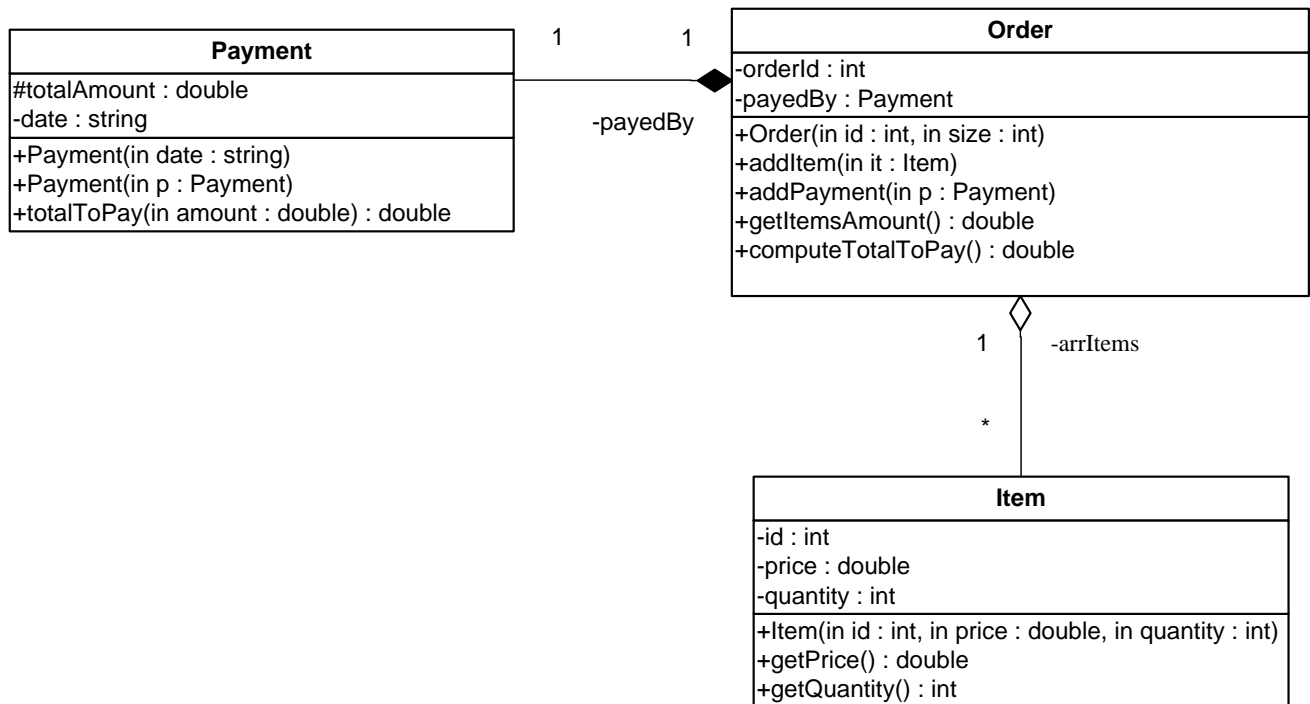
PayByCreditCard class

- Attributes:
 - ***cardNumber***: the credit card number.
 - ***expiryMonth***: the month of the expiry date of the credit card.
 - ***expiryYear***: the year of the expiry date of the credit card.
- METHODS:

- *PayByCreditCard* (*date: String, cardNb: String, month: int, year: int*): constructor.
- *PayByCreditCard* (*pcr: PayByCreditCard*): copy constructor.
- *totalToPay(amount: double): double*: this method receives an amount, calculates and returns the *total amount of the payment* as mentioned above.

QUESTION: Translate into Java code the class *Payment* and the class *PayByCreditCard*.

Exercise 2



Payment class: described in exercise 1.

Order class

- Attributes:
 - **orderId**: the order number.
 - **payedBy**: the payment mode of the order.
- METHODS:
 - **Order (id: int, size: int)**: constructor. It receives the order number and the number of items.
 - **addItem (it: Item)**: adds a new item in the array **arrItems**.
 - **addPayment (p: Payment)**: this method assigns the *payment mode p* to the attribute **payedBy**.

Notice: The attribute **payedBy** is not an array.

The relationship with Payment is a **composition**.

- **getItemsAmount(): double**: this method returns an exception “No Items” if there are no items in the order. Otherwise, it calculates and returns the **total amount of the items** of the order as follows:

$$\text{total amount of the items} = \sum_{i=1}^{nb \text{ of items}} (\text{price of the item} * \text{quantity})$$

- **computeTotalToPay(): double**: this method returns the **total amount to pay** of the order.

The **total amount to pay** of the order is calculated by the object *payedBy* based on the **total amount of the items** of the order using the method *totalToPay()*.

QUESTION: Translate into Java code the class *Order*.

1.1. Solution Final Exam

Exercise 1

```
public abstract class Payment {

    protected double totalAmount;
    private String date;

    public Payment(String d) {
        date = d;
        totalAmount = 0.0;
    }
    public Payment(Payment p) {
        date = p.date;
        totalAmount = p.totalAmount;
    }

    public abstract double totalToPay(double d);
}

public class PayByCreditCard extends Payment {
    private String cardNumber;
    private int expiryMonth;
    private int expiryYear;

    public PayByCreditCard (String d, String cardNumber, int month, int
year) {
        super(d);
        this.cardNumber = cardNumber;
        expiryMonth = month;
        expiryYear= year;
    }

    public PayByCreditCard (PayByCreditCard pcr) {
        super(pcr);
        this.cardNumber = pcr.cardNumber;
        expiryMonth = pcr.expiryMonth;
        expiryYear= pcr.expiryYear;
    }

    public double totalToPay(double amount) {
        totalAmount = amount - amount * 0.05;
        return totalAmount;
    }

    public int getExpiryYear() {
        return expiryYear;
    }
}
```

Exercise 2

```

public class Order {
    private int orderId;
    private Payment payedBy;
    private Item[] arrItems;
    private int nbItems;

    public Order(int id, int size) {
        orderId = id;
        arrItems = new Item[size];
        nbItems = 0;
        payedBy = null;
    }
    public void addItem(Item it) {
        if (nbItems < arrItems.length) {
            arrItems[nbItems] = it;
            nbItems ++;
        }
    }
    public void addPayment(Payment p) {
        if (p instanceof PayByCreditCard) {
            payedBy = new PayByCreditCard((PayByCreditCard)p);
        }
        else {
            payedBy = new PayByCheck((PayByCheck) p);
        }
    }

    public double getItemsAmount() throws Exception {
        double res = 0.0;
        int i;

        if (nbItems == 0) throw new Exception("No Items");

        for (i =0; i<nbItems; i++) {
            res += arrItems[i].getPrice() * arrItems[i].getQuantity();
        }
        return res;
    }

    public double computeTotalToPay() {
        double total = 0.0;

        try {
            total = payedBy.totalToPay(getItemsAmount());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return total;
    }
}

```

1.2. Midterm 2

Exercise 1.

1. What is the output of the following? 4 Points

```
public class ExceptTest{  
  
    public static void main(String args[]){  
        int a[] = new int[2];  
        try{  
            System.out.println("Access element three :" + a[3]);  
        }catch(ArrayIndexOutOfBoundsException e){  
            System.out.println("Exception thrown");  
        }  
        finally{  
            a[0] = 6;  
            System.out.println("First element value: " +a[0]);  
            System.out.println("The finally statement is executed");  
        }  
    }  
}
```

Answer :

Exception thrown**2**
First element value: 6**1**
The finally statement is executed**1**

2. What will happen when you attempt to compile and run the following class? **4 Points**

```
public class Base{
    public Base(int i){
        System.out.println("Base");
    }
}

public class Second extends Base {
    public static void main(String args[]){
        Second s = new Second ();
    }
    public Second (){
        System.out.println("Second");
    }
}
```

Choose the right answer

- 1) Compilation and output of the string "Second" at runtime
- 2) Compilation and no output at runtime
- 3) Compilation and output of the string "Base"
- 4) Compile time error: An error occurs at the constructor of the class Second because this constructor calls implicitly the default constructor (without parameter constructor) of the class Base which does not exist.

Answer:

Option 4.

EXERCISE 2

Give the output of the following program. **10 Points**

```
public class Flight {
    private String flightNum;
    protected int dist;

    public Flight() {
        flightNum = "Unknown";
        dist = 500;
        System.out.println ("The Flight is Created");
    }

    public Flight (String flightNum, int dist) {
        this.flightNum = flightNum;
        this.dist=dist;
    }

    public void display() {
        System.out.println ("Flight number: " + flightNum + " distance: " + dist );
    }

    public int cost () throws Exception {
        return 200;
    }
}
```

```
public class LongDistanceFlight extends Flight {
    protected int rate;

    public LongDistanceFlight () {
        rate = 2;
    }

    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() throws Exception {
        if (dist < 1000 ) throw new Exception (
            "Exception: Distance Less Than 1000 Km");
        return (super.cost()+ dist*rate);
    }
}
```

```
public class InternationalFlight extends LongDistanceFlight{
```

```

protected int airportFee;

public InternationalFlight(String s, int d, int r, int f) {
    super(s,d,r);
    airportFee = f;
}

public InternationalFlight(int f) {
    airportFee = f;
}

public void display() {
    System.out.println ("International Flight ");
    super.display();
    try {
        System.out.println(this.cost());
    }
    catch (Exception e) {System.out.println(e.getMessage());
    }
}

public int cost() throws Exception {
    return (super.cost()+airportFee);
}
}

```

```

public class TestFlights {
    public static void main(String[] args) {
        int i;
        Flight [] flightList = new Flight[2];

        flightList[0] = new InternationalFlight("SV3875", 1000, 3, 100);
        flightList[1] = new InternationalFlight(500);

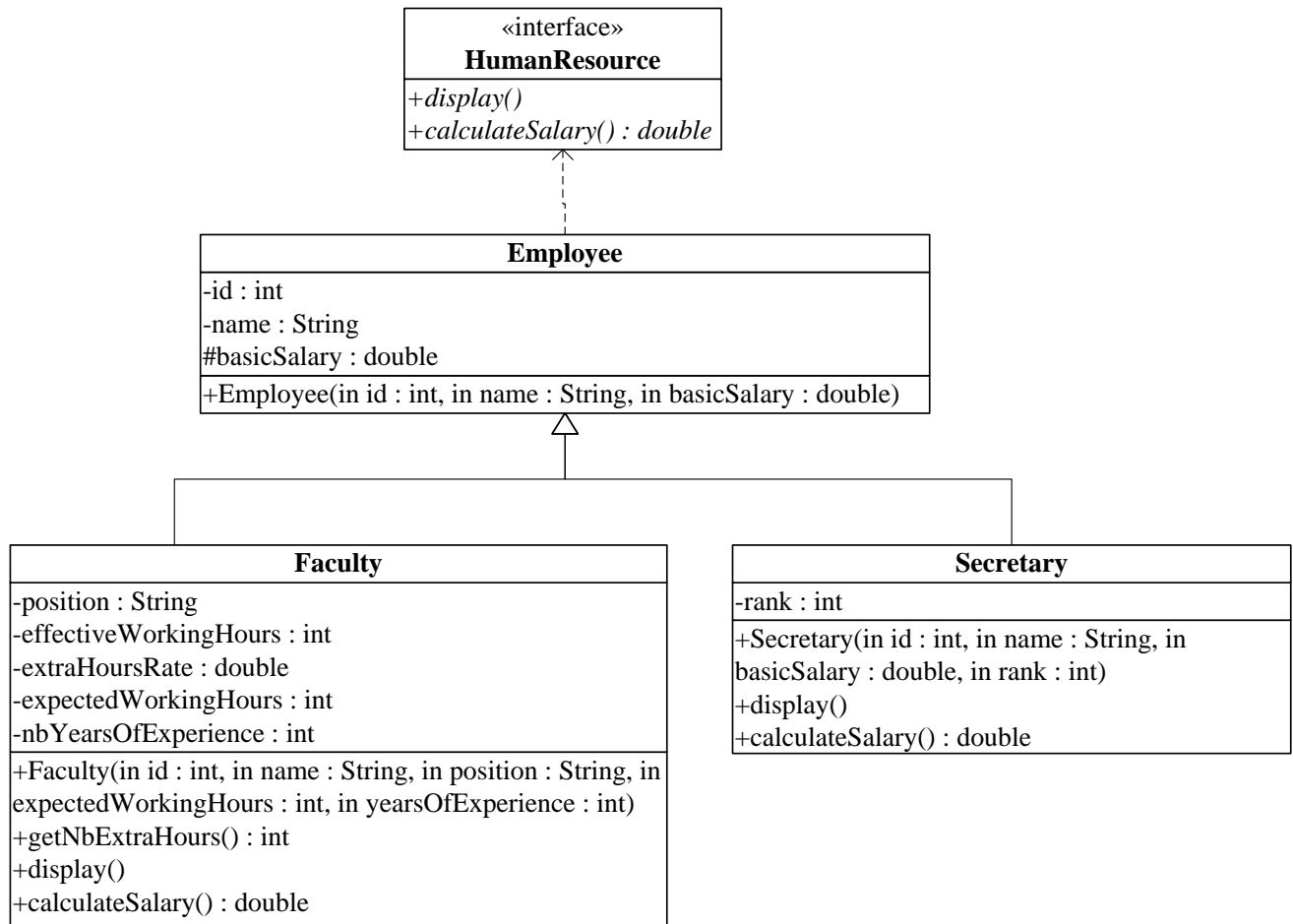
        for (i=0; i< 3; i++) {
            System.out.println("Iteration " + (i+1));
            try {
                flightList[i].display();
            } catch (Exception e) { System.out.println(
                "Exception in Iteration " + (i+1));
            }
        }
    }
}

```

Answer:

The Flight is Created1
Iteration 1
International Flight1
Long Distance Flight1
Flight number: SV3875 distance: 10001
33001
Iteration 2
International Flight1
Long Distance Flight1
Flight number: Unknown distance: 5001
Exception: Distance Less Than 1000 Km1
Iteration 3
Exception in Iteration 31

EXERCISE 3



HumanResource interface:

- METHODS:
 - ***display()***: this method displays all the attributes.
 - ***calculateSalary***: calculates and returns the salary.

Employee class

- METHODS:
 - ***Employee(id: int, name: string, basicSalary: double)***: constructor.
 - ***display()***: this method displays all the attributes of the employee.
 - ***calculateSalary***: returns the salary of the employee which is calculated as following:
 - ***For Faculty***:
 $\text{salary} = \text{basic salary} + (\text{number of extra hours} * \text{extra hours rate} * 4) + (\text{number of years of experience} * 500).$
 - ***For Secretary***:

salary = basic salary + (rank * 2000).

Faculty class

- Attributes:
 - **position**: the position of the Faculty.
 - **effectiveWorkingHours**: the number of working hours.
 - **extraHoursRate**: the rate of an extra hour.
 - **expectedWorkingHours**: The minimum load of the faculty.
 - **nbYearsOfExperience**: The number of years of experience.
- METHODS:
 - **Faculty(id: int, name: string, basicSalary: double, expectedWorkingHours: int)**: constructor.
 - **display()**: this method displays the salary and all the attributes of the Faculty.
 - **getNbExtraHours**: this method returns the number of extra hours which is calculated as following:
 - number of extra hours = effective working hours - expected working hours.*
 - N.B:**
 - If the number of extra hours is less than 0 this method throws an exception.
 - the number of extra hours is obtained **only** using the method getNbExtraHours.

Secretary class

- Attributes:
 - **rank**: the rank of the Secretary.
- METHODS:
 - **Secretary(id: int, name: string, basicSalary: double, rank: int)**: constructor.
 - **display()**: this method displays the salary and all the attributes of the secretary.

QUESTION: Translate into Java code the interface HumanResource, the class **Employee** and the class **Faculty**.

Answer:

```
public interface HumanResource { .....1
    public void display(); .....1
    public double calculateSalary(); .....1
}

public abstract class Employee implements HumanResource { .....1 + .....1
    private int id; .....0.5
    private String name; .....0.5
    protected double basicSalary; .....0.5

    public Employee(int i, String s, double d) {
        id = i; .....1
        name = s; .....1
        basicSalary = d; .....1
    }

    public void display() {
        System.out.println(id + name + basicSalary); .....3
    }
}

public class Faculty extends Employee { .....1
    private String position; .....0.5
    private int effectiveWorkingHours; .....0.5
    private double extraHoursRate; .....0.5
    private int expectedWorkingHours; .....0.5
    private int nbYearsOfExperience; .....0.5

    public Faculty(int i, String s, double bs, String pos, int exwh, int
years) {
        super(i, s, bs); .....2
        position = pos; .....1
        expectedWorkingHours = exwh; .....1
        nbYearsOfExperience = years; .....1
    }

    public void display() {
        super.display(); .....2

        System.out.println(position + effectiveWorkingHours+
            extraHoursRate + expectedWorkingHours +
            nbYearsOfExperience); .....2.5
        System.out.println(calculateSalary()); .....2.5
    }

    public int getNbExtraHours() throws Exception{ .....1
        int nbExtra = effectiveWorkingHours - expectedWorkingHours;
.....2
    }
}
```

```

    if (nbExtra < 0) .....1
        throw new Exception("Number of Extra Hours Less than 0");
        .....2
    return nbExtra; .....1
}

public double calculateSalary () {
    double salary = 0.0 ;
    try { .....1
        salary = basicSalary +
            (getNbExtraHours() * extraHoursRate *4) +
            (nbYearsOfExperience * 500.0); .....3
    } catch (Exception e) { System.out.println(e.getMessage()); }
.....1 + .....1
    return salary; .....1
}
}

```

1.3. Midterm 1

EXERCISE 1

Give the output of the following program.

```
public class Flight {
    protected String flightNum;
    protected int dist;
    public Flight () {
        System.out.println ("Created Flights:"); }

    public Flight (String flightNum, int dist) {
        this.flightNum = flightNum;
        this.dist=dist; }

    public void display() {
        System.out.println ("Flight number:" + flightNum );
        System.out.println ("distance:" + dist); }

    public int cost () { return 200; }
}
```

```
public class InternationalFlight extends Flight{

    public InternationalFlight (String flightNum, int dist) {
        super(flightNum, dist); }
    public void display () {
        System.out.println ("International Flight ");
        super.display()); }
    public int cost () { return (super.cost()+dist*2); }
}
```

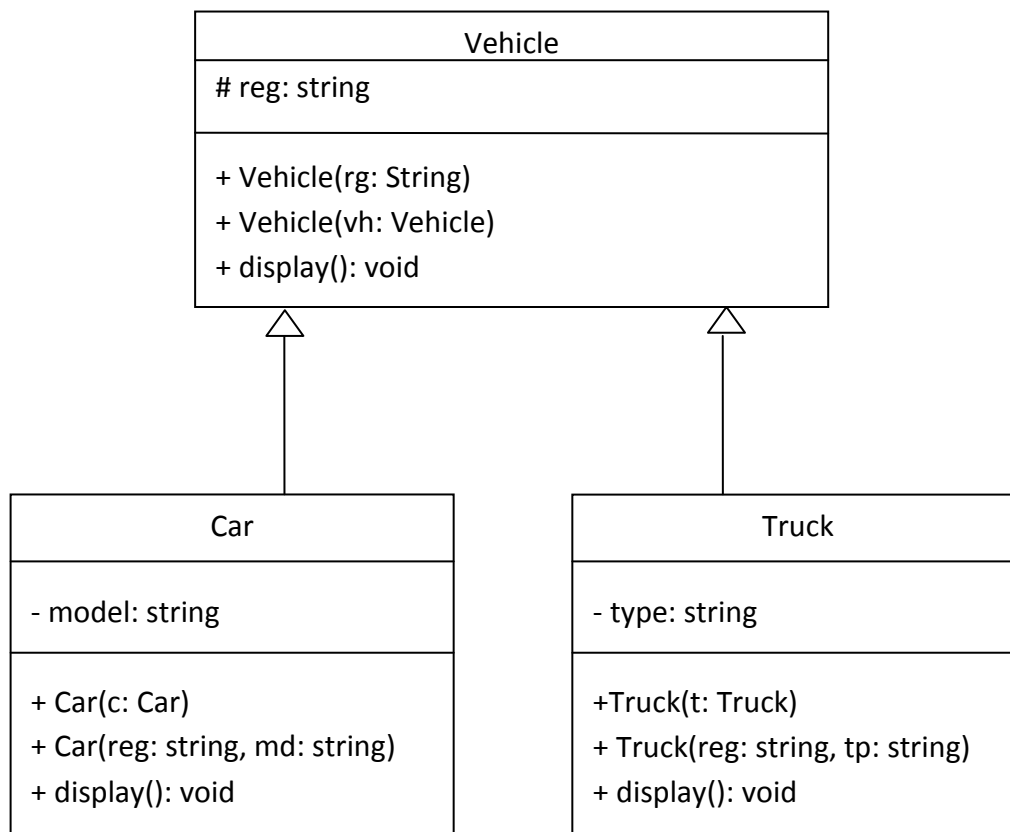
```
public class test {
    public static void main(String[] args) {
        int i;

        Flight x = new Flight ();
        Flight [] flightList = new Flight [2];
        flightList [0]= new InternationalFlight ("FRA334", 4000);
        flightList [1]= new InternationalFlight ("TUN654",3000);

        for (i=0; i< flightList.length; i++) {
            flightList[i].display() ;
            System.out.println("Cost: "+ flightList[i].cost() + "SR");
        }
    }
}
```

EXERCISE 2

Consider the following UML class diagrams:



And the following specification:

- **Vehicle** class
 - o ATTRIBUTES:
 - **reg**: vehicle's registration ID, like: ABC434.
 - o METHODS:
 - **Vehicle(rg: string)**: constructor.
 - **display()**: this method writes on the screen the attributes of **Vehicle**.

- **Car** class
 - o ATTRIBUTES:
 - **model**: car's model, like: Toyota Tercel.
 - o METHODS:
 - **Car(c: Car)**: constructor.
 - **Car(reg: string, md: string)**: constructor.
 - **display()**: this method writes on the screen the attributes of **Car**.

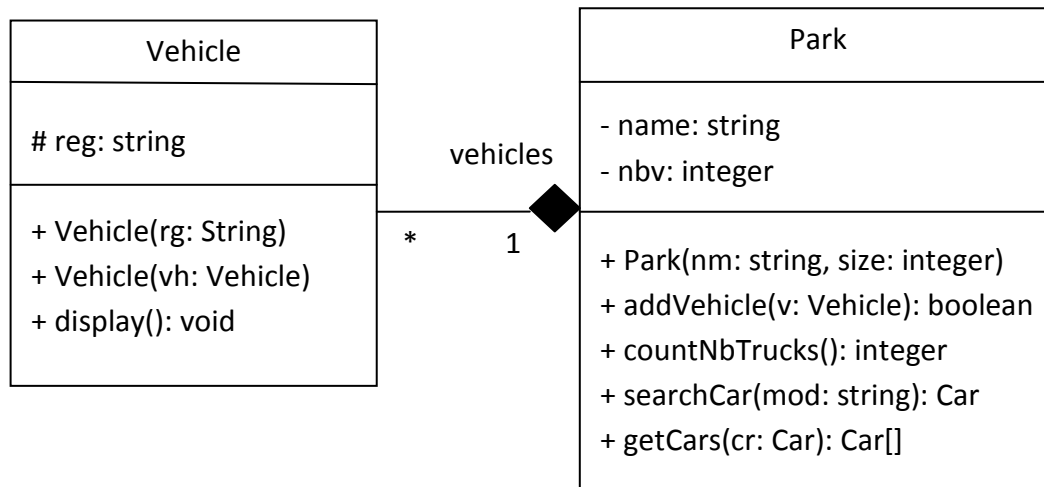
- **Truck** class
 - ATTRIBUTES:
 - **type**: truck's type, like: dump truck.
 - METHODS:
 - **Truck(t: Truck)**: constructor.
 - **Truck(reg: string, tp: string)**: constructor.
 - **display()**:this method writes on the screen the attributes of **Truck**.

QUESTION: Translate into Java code **Vehicle** and **Car** classes only, taking into consideration the above specification.

EXERCISE 3

In order to represent parks of vehicles that may contain cars and trucks, we add a new class,

Park:



Park class

○ METHODS:

- **Park(nm: string, size: integer):** constructor.
- **addVehicle(v: Vehicle):** this method adds a vehicle to the park.
- **countNbTrucks():** returns the number of trucks in the park.
- **searchCar(mod: string):** returns a car that has the model mod.
- **getCars (c: Car):** returns all the cars that have the same model as the model of **cr**.

QUESTION: Translate into Java code the class **Park** class only. Assume that all getters and setters are available.

1.4. Solution Midterm 1

Answer Exercise 2:

```
public class Vehicle
{
    protected String reg;

    public Vehicle(String rg)
    {
        reg = rg;
    }

    public Vehicle(Vehicle v)
    {
        this.reg = v.reg;
    }

    public void display()
    {
        System.out.println("Vehicle's reg: " + reg);
    }
}

public class Car extends Vehicle
{
    private String model;

    public Car(Car c)
    {
        super(c);
        model = c.model;
    }

    public Car(String reg, String md)
    {
        super(reg);
        model = md;
    }

    public void display()
    {
        super.display();
        System.out.println("Car's model: " + model);
    }
}
```

Answer Exercise 3:

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

    public Park(String nm, int size)
    {
        name = nm;
        vehicles = new Vehicle[size];
    }

    public boolean addVehicle(Vehicle v)
    {
        if (nbV < vehicles.length)
        {
            if(v instanceof Truck)
            {
                vehicles[nbV] = new Truck((Truck)v);
            }
            else
            {
                vehicles[nbV] = new Car((Car)v);
            }
            nbV++;
            return true;
        }
        return false;
    }

    public int countNbTrucks()
    {
        int nbT = 0;
        for(int i=0; i<nbV; i++)
        {
            if(vehicles[i] instanceof Truck)
            {
                nbT++;
            }
        }
        return nbT;
    }

    public Car searchCar(String mod)
    {
        for(int i=0; i<nbV; i++)
        {
            if((vehicles[i] instanceof Car))
            {
                if( ((Car)vehicles[i]).getModel().equals(mod))
                {
                    return (Car)vehicles[i];
                }
            }
        }
    }
}
```

```

    }
    return null;
}

public Car[] getCars(Car cr)
{
    Car[] toRet = new Car[vehicles.length];
    int j = -1;

    for(int i=0; i<nbV; i++)
    {
        if((vehicles[i] instanceof Car))
        {
            if( ((Car)vehicles[i]).getModel().
equals(cr.getModel()))
            {
                j++;
                toRet[j] = new Car((Car)vehicles[i]);
            }
        }
    }
    if(j >= 0 )
    {
        return toRet;
    }
    else
    {
        return null;
    }
}

}

public class Application
{
    public static void main(String[] args)
    {
        Park pk = new Park("Park1", 50);
        Car c1 = new Car("CRA122" , "Corolla");
        Car c2 = new Car("CRA656", "Civic");
        Truck t = new Truck("TTR999", "Dump");

        pk.addVehicle(c1);
        pk.addVehicle(c2);
        pk.addVehicle(t);

        System.out.println("Number of trucks: " + pk.countNbTrucks());
        Car c = pk.searhCar("Civic");
        if (c != null)
        {
            c.display();
        }
    }
}

```