CSC111 Lab

Object Oriented Programming – Lab IIII

---- Lab 9 ---

Create a project **Lab09**

Q1) Write a Java class **Car** that represent a car in a car dealership. The car has the following attributes:

- **model**: String.
- **year**: int. The year the car was made.
- **mileage**: int representing how many kilometers the car has drove.
- **soom**: double. Best offer for the car so far.
- **hadd**: double. Asking sale price. Must be greater than soom.

Car
- model: String - year: int - mileage: int - soom: double - hadd: double
+ Car () + Car(String model, int year, int mileage, double soom, double hadd) + setModel(String model):void + getModel():String + setYear(Int year):void + getYear() :Int + setMileage(int mileage):void + getMileage() :int + setSoom(double soom):void + getSoom() :double + setHadd(double had):void + getHadd():double + similar(Car c): boolean + difference():double + printInfo(): void

The class has the following methods:

- **Car**()
- **Car**(String model, int year, int mileage, double soom, double hadd)
- **setters and getters** for **model, year, mileage, soom** and **hadd.**
- **similar(Car c):boolean** Compares two objects, the calling object and the argument c and returns true if they are similar, false otherwise. Two cars are similar if they have same model, year and within 10k of eachother mileage. For example, 2014 ford with mileage 120K is similar to a 2014 ford with mileage 125K.
- **difference**():double. the difference between soom and hadd. Must be positive.
- **printCar**(): void

Write a class **testCar** that will do the following:

1. Declare objects **car1** and **car2** of class Car.
2. Read car attributes from the user using the both constructors.
3. Print both cars information.
4. Ask the user repeatedly to enter soom for car1 until **soom** is within 2% of **hadd**.
5. If mileage of car2 is higher than 100000 KM, decrease it by 50000 KM, otherwise leave it as it is.
6. Print both cars information.
7. Feel free to try creating additional objects, read their data and manipulate them.
8. exit.

Sample Run

```
Please enter car model, year, mileage, hadd and soom
Ford 2015 120000 50000 30000
Please enter car model, year, mileage, hadd and soom
Toyota 2014 150000 70000 50000
This car is a Ford and was made in 2015.
It has 120000 KM and soom is 30000.0 Hadd is 50000.0
This car is a Toyota and was made in 2014.
It has 150000 KM and soom is 50000.0 Hadd is 70000.0
Cars are not similar
```

Q2) Design a class named **Ball** that models a moving ball in two-dimensional space. The class can be represented as:

Ball
- x: double - y: double - distTraveledX: double - distTraveledY: double - totDistXAllBalls: double - totDistYAllBalls: double - lastX: double - lastY: double
+ Ball() + Ball(newX: double, newY: double) + getX(): double + getY(): double + move(xDisp: double, yDisp: double): void + getDistTraveledX (): double + getDistTraveledY(): double + getTotDistXAllBalls (): double + getTotDistYAllBalls (): double + toString(): String

Data Members:

- double **x**: position of the ball on X-Axis in a two-dimensional space.
- double **y**: position of the ball on Y-Axis in a two-dimensional space.
- double **distTraveledX** total distance traveled by the ball throughout all its moves on X-Axis.
- double **distTraveledY** total distance traveled by the ball throughout all its moves on Y-Axis.
- static double **totDistXAllBalls** keeps the total distance traveled by all balls throughout all the moves on X-Axis.
- static double **totDistYAllBalls** keeps the total distance traveled by all balls throughout all the moves on Y-Axis.
- static double **lastX** store the last X-Axis position of the most recent ball that has moved.
- static double **lastY** store the last Y-Axis position of the most recent ball that has moved.

Methods:

- A default constructor that sets initial position of the ball to (0, 0).
- A constructor that receives two arguments newX and newY and sets initial position of the ball to them.
- **getX**() a getter that returns the value of x.
- **getY**() a getter that returns the value of y.
- **move**(double **xDisp**, double **yDisp**):void , which changes x and y by the given xDisp and yDisp, respectively. A ball must not move if it is going to finish its movement in a position occupied by another ball (Hint: use lastX and lastY to determine this. If movement is allowed, then change the values of lastX and lastY).
- **getDistTraveledX**():double. Returns the distance traveled by the current ball throughout all of its moves on X-Axis.
- **getDistTraveledY**():double. Returns the distance traveled by the current ball throughout all its moves on Y-Axis.

- public static double **getTotDistXAllBalls**() that return the distance traveled by all balls throughout all of the moves on X-Axis.
- public static double **getTotDistYAllBalls**() that return the distance traveled by all balls throughout all of the moves on Y-Axis.
- **toString**(), which returns the string "**Ball @ (x,y)**".

Create a class **testBall** to test our Ball class. This class will do the following:

- It creates a new ball with a position (2, 2).
- Then it moves the ball by (3, -2).
- Then it moves the ball by (2, -7).
- Then it creates a new ball with a position (0,0).
- Then it moves the ball by (7, -7). Notice that the second ball must not move since it will otherwise occupy the same place occupied by the first ball.
- Then it moves the ball by (2, 4).

Finally, it prints the last position of the two balls using toString method and the total distance traveled on both X-Axis and Y-Axis by each ball and by all balls.

Sample Run

```
B1 Ball @ (2.0,2.0)
B1 Ball @ (5.0,0.0)
B1 Ball @ (7.0,-7.0)
Distance travelled on X for b1 5.0
Distance travelled on y for b1 9.0
B2 Ball @ (0.0,0.0)
The ball in position (0.0,0.0) cannot move to position (7.0,-7.0)
B2 Ball @ (0.0,0.0)
B2 Ball @ (2.0,4.0)
Distance travelled on X for b2 2.0
Distance travelled on y for b2 4.0
Distance travelled on X for all balls 7.0
Distance travelled on Y for all balls 13.0
```

Q3) You have been asked by Saudi Wildlife Authority to design a class named **Species** to manage endangered species. Class details are as following:

- Three data fields (properties) which are:
    - **name** of type **String** which stores name of species,
    - **population** of type **int** which stores the current population of the species and it can not be negative, and
    - **growthRate** of type **double** which stores the growth rate of the current population.
- A method **readInput()** that reads the values of data members. The method must not allow the user to enter incorrect population value. If this happens, it should keep asking for correct value until the user enters it.
- A method **writeOutput()** that prints the data of the species.
- A method **predictPopulation(int years)** that returns the projected population of the species after the specified number of years (which must be a non-negative number).
- A method **setSpecies(String newName, int newPopulation, double newGrowthRate)** that sets values of receiving object to new data sent through parameters. The method should print an error message if **newPopulation** is a negative number.
- Getter methods for **name, population** and **growthRate**.

Draw the UML diagram for the class and then implement the class. Write a test program that :

1. Declare object s of the class Species
2. Reads the information of some species X.
3. Stores the information in s only after making sure population is not negative.
4. Ask the user to enter number of years to predict population
5. Print the species population after the years given above.
6. Repeat step 4 and 5 until the user enters "Stop" as name of the species.

Name your classes **Species** and **TestSpecies**. Use two separate files for each of the two classes.

Sample Run

```
What is the species' name? Arabian Tiger
What is the population of the species? 200
Enter growth rate (% increase per year): 3
Enter how many years? 100
The population of the Arabian Tiger after 100 years is 3843
```