CS F213: OOP Lab 4
Static, Inheritance, Nested classes, Access modifiers

Tanya Khera, Vishal Sharma

16th March 2021

The objective of this lab is to test your understanding of the static keyword, inheritance, nested classes, and access modifiers. You will create a model of multiple vehicle showrooms in a city, each of which buys, sells, and does some servicing for two kinds of vehicles - cars and bikes.

General Instructions -

Read the following instructions very carefully before starting your work -

1. Make sure that the source code is written in the default package.
2. Ensure that the name of each file is the same as the corresponding class name.
3. Ensure that the name of methods is the same as those given in Javadoc. Failure to comply may cause deduction in credit even if the implementation is correct.
4. You won't be provided with a template file (helper code). You have to implement all the classes by yourself (including their respective constructors).
5. Do not copy directly the names of methods from Javadoc. Write them out yourselves.
6. You are required to upload all five classes at once. It is not possible to test individual classes on Codepost.
7. A brief description of the classes and methods to be used is given further in the problem statement but you **must refer to the Javadoc for the complete description.**

8. Be sure to check your code on the test cases provided locally (as jar file), before submitting on Codepost. **Remember that Codepost takes only the latest submission and not the best submission into consideration.**
9. The advisable order of writing the code:
   a) Vehicle and Tyre
   b) Car
   c) Bike
   d) City
   e) Showroom

General assumptions -

These are a set of assumptions that will be useful to you while solving the lab:

1. Each showroom is a showroom+servicing center, where tyres can also be changed.
2. Assume all vehicles are always available for sale.
3. Each showroom will have only one vehicle of a particular model and company (e.g. it would have only one Honda Civic, if any). It will not try to buy another car with the same model and company, if it has one.
4. maxPrice will be reasonable (it will be less than or equal to currentAccountBalance).
5. A car can be bought from **any** showroom that has it and is selling it at a price <= maxPrice that the buying showroom is willing to pay.
6. A showroom will not try to buy more cars than the maximum it can have.
7. The price of a vehicle will remain fixed throughout its life.
8. The indices provided to removeCar(), removeBike(), sellCar(), sellBike() will always be valid.

Tips -

1. The Car and Bike classes are quite similar. Once you code one of them, coding the other should be easy and quick.
2. Some methods make use of other methods.

Note: The below description of classes is extremely brief. Only the names of fields and methods have been written (along with some additional information/description, if required). Students MUST refer to the javadoc for the complete details (use, return types, arguments, etc.) of all methods and fields.

Class Descriptions -

1. **Vehicle**
    a. Fields -
        - **companyName**
        - **modelName**
        - **registrationNum**
        - **price**

    b. **Nested classes** -
        - **Tyre**
          In this fictional world, a tyre is going to exist only in the context of a vehicle. So, creating a new tyre will also have to be in the context of some vehicle.
            a. Fields -
                - **tyreCompany**
                - **vehicleType**

            b. Methods -

            - **getVehicleType()**
            - **getTyreCompany()**

   c. Methods -
      - **getCompanyName()**
      - **getModelName()**
      - **getRegistrationNum()**
      - **getPrice()**

2. **Car**
   A car is a type of vehicle.
   a. Fields -
      - **tyres** - An array consisting of tyre objects, which represents the tyres of the car. Every car will have four tyres, and each of those tyres will be of the same company (though the tyres of a car can be removed and new ones can be put in, of the same/a different company. Changing of tyres will happen in entirety - i.e. all four tyres will be changed at a time).

   b. Methods -
      - **getTyres()** - Method to return the array containing the car's tyres
      - **changeTyres()** - Method to change the car's tyres. Changing of tyres will happen in entirety - i.e. all four tyres will be changed at a time. This method expects a valid array of four car tyre objects to be passed to it.

3. **Bike**
   A bike is a type of vehicle.
   a. Fields -

  - **tyres** - An array consisting of tyre objects, which represents the tyres of the bike. Every bike will have two tyres, and each of those tyres will be of the same company (though the tyres of a car can be removed and new ones can be put in, of the same/a different company. Changing of tyres will happen in entirety - i.e. both tyres will be changed at a time).

b. Methods -
  - **getTyres()** - Method to return the array containing the car's tyres
  - **changeTyres()** - Method to change the bike's tyres. Changing of tyres will happen in entirety - i.e. both tyres will be changed at a time. This method expects a valid array of two bike tyre objects to be passed to it.

4. **City**
A city has five showrooms in it, represented by an array of showrooms. We want to be able to access these showrooms without instantiating the City class.
   a. Fields -
      - **showrooms** - The array containing all the showrooms in the city.

5. **Showroom**
A showroom has different vehicles, and the ability to sell its own vehicles, buy vehicles from other showrooms, and change vehicles' tyres.
   a. Fields -
      - **showroomId**
      - **currentAccountBalance** - The showroom has one account using which it makes any purchases and sales, i.e. money from sales gets added and money for purchases gets subtracted from this account. This field represents the current balance of that account. Initialized to 10000.

- **numCars** - Number of cars the showroom currently has. Initialized to 0.
- **numBikes** - Number of bikes the showroom currently has. Initialized to 0.
- **cars** - An array containing all the cars in the showroom.
- **bikes** - An array containing all the bikes in the showroom.
- **MAX_CARS** - A constant integer having the maximum number of cars that a showroom can have **(must be initialized to 10)**.
- **MAX_BIKES** - A constant integer having the maximum number of bikes that a showroom can have **(must be initialized to 15)**.

b. Methods -
- **getCars()**
- **getBikes()**
- **getNumBikes()**
- **getNumCars()**
- **getCurrentAccountBalance()**
- **getShowroomId()**
- **addCar()** - This method is used to add a given car to the cars of the showroom (this method does not modify the account balance).
- **addBike()** - This method is used to add a given bike to the bikes of the showroom (this method does not modify the account balance).
- **removeCar()** - This method is used to remove a car (at the given index in the cars array) from the showroom, for example when it is being sold (removeCar() does not modify the account balance).
- **removeBike()** - This method is used to remove a bike (at the given index in the bikes array) from the showroom, for example when it is being sold (removeBike() does not modify the account balance).
- **sellCar()** - This method is used to sell a car (at the given index in the cars array) from the showroom.

- **sellBike()** - Very similar to sellCar(). This method is used to sell a bike (at the given index in the bikes array) from the showroom.
- **buyCar()** - Method to buy a car (of a particular company and model) from **another** showroom, with the given price requirement, if such a car is found.
- **buyBike()** - Very similar to buyCar(). Method to buy a bike (of a particular company and model) from **another** showroom, with the given price requirement, if such a bike is found.
- **changeCarTyres()** - Method to change the tyres of a given car, with new tyres of a given company.
- **changeBikeTyres()** - Method to change the tyres of a given car, with new tyres of a given company.
- **<u>HINT</u> for changeCarTyres() and changeBikeTyres():**
  i. Use the car object (bike object) passed to changeCarTyres (changeBikeTyres), for making new tyre objects.
  ii. The syntax for making an array of objects of a nested class is:
  OuterClass.InnerClass[] arr = new OuterClass.InnerClass[length];

(Test cases marks distribution on next page)

Test cases

Methods	Marks allotted
getShowroomId(), getCurrentAccountBalance(), getNumCars(), getNumBikes(), getCars(), getBikes()	0.5
<u>Bike and Car methods</u>: getCompanyName(), getModelName(), getRegistrationNum(), getPrice(), getTyres <u>Tyre methods</u>: getVehicleType(), getTyreCompany(),	0.5
addCar(), buyCar(), getCurrentAccountBalance(), getNumCars()	2
addBike(), sellBike(), getCurrentAccountBalance(), getNumBikes()	2
changeCarTyres(), getCars(), getTyres(), getTyreCompany()	1
changeBikeTyres(), getBikes(), getTyres(), getTyreCompany()	1
Total	7