Object-Oriented Programming Fundamentals

Laboratory 7 (Week 7) Handout

Objectives

- To practise writing methods that take object parameters
- To practise defining classes that have class type attributes
- To practise writing tests

Reference:

- Lectures 13, 14 and 15

☺ **Required Questions** *Complete all questions* (applying concepts to solve problems)

In this lab we are going to be writing some classes for a vintage car club membership program. To implement this program we will need a class to represent a *vintage car* and a class to represent a *car club member*.

Vintage cars are classified according to the year they were made. There are four categories: Historic, Vintage, Post Vintage and Other.

Question 1 - VintageCar class

The **`VintageCar`** class represents a car that is registered as a member's car in the club. The UML class diagram for the **`VintageCar`** below shows the attributes and methods that you will write for it.

VintageCar
- String make - String model - int year - boolean originalBody - String category
+ VintageCar(String, String, int, boolean) - void setCategory() + String getMake() + String getModel() + int getYear() + boolean getOriginalBody() + String getCategory() + void setOriginalBody(boolean) + boolean isEligibleCar() + int compareAge(VintageCar) + boolean sameType(VintageCar) + String toString()

Be careful to implement the UML diagram **exactly** as it is on the left.

There is a driver class already written, **`CarTester.java`**, that uses the names of methods as written in the UML diagram.

If you do not follow the UML diagram exactly, the driver class will not compile.

a) Create a file called *VintageCar.java* and declare the class and define the attributes for the **`VintageCar`** class as shown in the Class diagram above.

b) Create a **private** method called **`setCategory()`** that will be called by the constructor to initialise the **`category`** attribute. This method checks the year attribute and then sets **`category`** to **`"Historic"`**, **`"Vintage"`**, **`"Postvintage"`** or **`"Other"`** according to the following table:

Year	Category
Pre 1919	Historic
1919 to 1930	Vintage
1931 to 1939	Postvintage
1940 and later	Other

c) Create the constructor. It should set **`make`** to the first string argument**,** **`model`** to the second string argument, **`year`** to the integer argument and **`originalBody`** to the boolean argument and then call **`setCategory()`** to set the **`category`** attribute.

d) Create the 5 accessor methods (get methods) to return the values of the attributes.

e) The only attribute that can be altered is the status of the car body. Create a mutator method **`setOriginalBody()`** that takes a boolean argument and sets the attribute **`originalBody`** to that value.

f) In the car club, only members who own an eligible car are entitled to vote on certain matters. An eligible car is any car manufactured before 1940. So a **`Historic`**, **`Vintage`** or **`Postvintage`** car is eligible, while an **`Other`** car is not. Write a method **`isEligibleCar()`** that returns **`true`** if the car is eligible and **`false`** otherwise.

g) Write a method called **`compareAge()`** that compares the age of the object with the age of another **`VintageCar`** object which is passed into the method as an argument. This method must return 0 if the age of the two cars is the same. It should return a negative number if the car was made before the car argument and a positive number if the car was made after the car argument. Hint: you can do this by returning the result of a simple subtraction statement.

h) Write a method called **`sameType()`** that checks whether the **`make`** and **`model`** of the car are the same as the **`make`** and **`model`** of another **`VintageCar`** object which is passed in to the method as an argument. The method should return **`true`** if both the **`make`** and the **`model`** are the same and **`false`** if either or both the **`make`** and **`model`** are different.

i) Write a **`toString()`** method that returns a string representing the state of the object. The string returned must be formatted *exactly* as follows.

For an object created as follows:

```
VintageCar car14 = new VintageCar("Buick",
                                  "Roadster", 1939, true);
```

the method returns the string:

```
VintageCar[make: Buick, model: Roadster, year: 1939,
originalBody: true, category: Postvintage]
```

Note: there is no newline character in the string – the representation above has wrapped because the string was too long to fit on a single line.

Question 2 – Testing the VintageCar class
Copy the file *CarTester.java* from the library area with the command:

```
cp /home/1st/csilib/cse1oof/lab07/CarTester.java .
```

`CarTester.java` tests the methods of the **`VintageCar`** class. It uses category and boundary value tests where appropriate. The actual outcome of each test is checked against the expected outcome and the success or failure of each test is reported.

Compile and run the tester program and check that all tests are reported as successful. If any tests fail, correct your **`VintageCar`** class and rerun the tester program until all tests are successful.

Extension Questions *OPTIONAL FOR ALL STUDENTS* (more challenging problems!)

Optional Question 3 – CarClubMember class (Challenging)
The **`CarClubMember`** class represents a member of the vintage car club. The UML class diagram for the **`CarClubMember`** below shows the attributes and methods that you will write for it.

CarClubMember
- String name - String phone - VintageCar car1 - VintageCar car2
+ CarClubMember(String, String) + String getName() + String getPhone() + String getCar1Details() + String getCar2Details() + void setPhoneNumber(String) + boolean addCar(VintageCar) + boolean changeOriginalBody(int, boolean) + boolean canVote() + String toString()

a) Create a file called *CarClubMember.java* and declare the class and define the attributes for the **`CarClubMember`** class as shown in the Class diagram above.

b) Create the constructor. It should set **`name`** to the first string argument, and **`phone`** to the second string argument. The attributes **`car1`** and **`car2`** are initially **`null`**.

c) Create the 4 accessor methods (get methods) to return the values of the attributes. **`getCar1Details()`** should return the **`toString()`** representation of the **`car1`** attribute if it is not **`null`**. It should return the string "none" if **`car1`** is **`null`**. Likewise, **`getCar2Details()`** should return "none" if car2 is **`null`** and the **`toString()`** representation of **`car2`** otherwise.

d) Create a mutator method **`setPhoneNumber()`** that takes a **`String`** argument and sets the attribute **`phone`** to that value.

e) Create the method **`addCar().`** The **`addCar()`** method take a **`VintageCar`** object as argument and makes it a registered car of the **`CarClubMember`** if it can. If the **`CarClubMember`** has no registered cars (**`car1`** is **`null`**) then **`addCar()`** makes **`car1`** refer to the argument object. If the **`CarClubMember`** has a single registered car already (**`car1`** is not **`null`** but **`car2`** is **`null`**) then **`addCar()`** makes **`car2`** refer to the argument object. In either of these cases the method returns **`true`** to indicate the successful addition of a car. If the member already has 2 registered cars (neither **`car1`** nor **`car2`** is **`null`**) the car cannot be added and the method must return **`false`**.

f) Create a method called **`changeOriginalBody().`** This method takes an integer representing which car to change and a boolean that is the required body status. If the integer argument is 1 (and **`car1`** is not **`null`**) then **`car1`** has the body status changed to the value of the second argument. If the integer argument is 2 (and **`car2`** is not **`null`**) then **`car2`** has the body status changed to the value of the second argument. In both these cases the method should return **`true`** to indicate the successful updating of the body status. If the integer passed in is not 1 or 2 or if the car referred to is **`null`** then the method must return **`false`** to indicate that the body status was not able to be updated.

g) Create a method called **`canVote()`**. This method returns **`true`** if the **`CarClubMember`** is allowed to vote and **`false`** otherwise. A member is allowed to vote if they have at least one eligible car.

h) Write a **`toString()`** method that returns a string representing the state of the object.

Optional Question 2 – Testing the CarClubMember class (Challenging!!)
Write a **`CarClubMemberTester`** program that tests each of the methods of the **`CarClubMember`** class. Use a similar strategy to the **`CarTester`** program that was used in Question 3.