The University of the West Indies, St. Augustine
COMP 2603 Object Oriented Programming 1
Week 2, Lab 2

Learning Objectives

- Define custom classes with state (primitive variables) and behaviour (accessors, mutators, methods)
- Create no-argument and simple constructors
- Implement a toString( ) method that makes use of accessor methods to return an appropriate description of an object.
- Instantiate classes by creating objects in a main class
- Import and use Java API classes (File reading, String tokenisation)

Create a new BlueJ Project called Lab2. Add the vehicles.txt file from myElearning to your project.

FuelStation Class

1. Write the code for a **FuelStation** class with:
    a. the following attributes:
        - **fuelType** (String) - default is gasoline
        - **fuelVolume** (double) - default is 75000
        - **fuelRate** (double) - default is 2
        - **fuelSales** (double) - default is 0

    Note: Ensure that information hiding principles are not violated. Use private access modifiers for all attributes.

    b. a **no-argument constructor** that initialises state of all attributes to their default values.
    c. **accessor** methods for all attributes. Follow the camelCase convention.
    d. a t**oString( ) method** that produces and returns a string as follows:
    FUEL: gasoline VOL: 75000.0L PRICE PER L: $2.00 SALES: $0.00

    Note: use all accessor methods to produce this string.

2. Add the following methods (error checking is required in all cases) to the FuelStation class:

Method Signature	**Return Type**	**Purpose**
sellFuel(double volume)	boolean	Decreases the fuelVolume of the FuelStation by the supplied volume, if possible, updates the sales values, returns true if successful, false otherwise. This method is private.
canDispenseFuelType (String fuelType)	boolean	Returns true if the FuelStation dispenses fuel of the supplied type, false otherwise. This method is public.
dispense(String fuelType, double volume)	boolean	Returns true if the FuelStation can dispense the supplied type and volume of fuel, false otherwise. This method is public.

Main Class

3. Create a main class, **StationSimulation**, that creates an instance of the **FuelStation** class and prints the result of invoking the **toString( )** method of the instance. Observe the relationship created in the BlueJ editor.

4. Test the three methods created in Step 2 using randomly generated values.

File Reading using Scanner

5. Import the **Scanner** and **File** classes from the Java API into the **StationSimulation** class.

6. Add code to read the data from the **vehicles.txt** file and print out each line of data to the screen. Remember to use try/catch blocks.

Vehicle Class

7. Write the code for a **Vehicle** class with:
    a. the following attributes:
        - **tankCapacity** (int)
        - **fuelType** (String)

    b. a **constructor** that accepts **3 int values** representing the **length**, **width**, **breadth** of the Vehicle's fuel tank. These variables are used to calculate and initialise the **tankCapacity** variable. The **fuelType** should be assigned to either gasoline (if the tankCapacity is even) or diesel (if the tankCapacity is odd).
    c. **accessor** methods for all attributes. Follow the camelCase convention.
    d. a **toString( )** method that produces and returns a string as follows:
    VEHICLE TANK CAPACITY: 60 FUEL TYPE: gasoline

Putting it all together

8. Modify your code block from Step 6 in the **StationSimulation** class to create **Vehicle** objects for each line of data. The three values on each line represent the length, width, breadth of a Vehicle's fuel tank. You will need to parse and extract the values from the line (Tip: Use StringTokenizer or a String's split(..) method) .Test your code works by printing out the details of each Vehicle object (using the toString( )).

9. Write code in the StationSimulation class to fill up the tanks of each Vehicle and print the outcomes. Diesel vehicles should not get any fuel. Sample output below:

```
FUEL: gasoline VOL: 75000.0L PRICE PER L: $2.00 SALES: $0.00

VEHICLE TANK CAPACITY: 60 FUEL TYPE: gasoline
Filled up: true
FUEL: gasoline VOL: 74940.0L PRICE PER L: $2.00 SALES: $120.00

VEHICLE TANK CAPACITY: 42875 FUEL TYPE: diesel
Filled up: false
FUEL: gasoline VOL: 74940.0L PRICE PER L: $2.00 SALES: $120.00

VEHICLE TANK CAPACITY: 400 FUEL TYPE: gasoline
Filled up: true
FUEL: gasoline VOL: 74540.0L PRICE PER L: $2.00 SALES: $920.00
```

Extra Exercises (Requires some research)

- Modify your code to use an ArrayList to store the Vehicle objects. Traverse the ArrayList and then fill up the tanks of each Vehicle.
- Modify the FuelStation class so that a second FuelStation object that dispenses diesel can be created. Traverse the ArrayList again and fill up the tanks of all vehicles at the appropriate FuelStation so that all (both gasoline and diesel) are successfully filled.