# COMP 111

# Lab 1 – Winter 2019

# Scenario

As you pace nervously outside the office of CIO Sam Snake, the poster displayed nearby catches your eye. The slogan jumps out at you, “SOUP up your engine with **S**nake **O**il **U**ltraBio **P**etrol.” During the hiring process you learned that Snake Oil aims to be a market leader in “biofuels” and is rapidly opening up stations across the Midwest to provide consumers with alternatives to traditional gasoline.

The door opens and a thin, tall man introduces himself and ushers you to a chair next to his broad, clean desk. “You like snakes?” the CIO laughs as he tosses a stuffed toy snake in your lap. “Others got a Gecko – we’ve got a snake!” he laughs. Sam is one of three Snake brothers leading the highly successful family fuel business.

“We are glad to have you aboard,” he continues, noting the sudden departure of a lead programmer has put Snake Oil in a bit of a fix. “Grace was very talented, and we were happy she was able to find a position she had been seeking as a Senior Software Engineer.”

He explains that Snake Oil customers need help in locating the nearest “SOUP” station while they travel. “Grace had been working on the software that will locate stations – both on our Snake Oil website and on smart phones. Now we need you to pick up the development of her prototype.” Mr. Snake describes Grace as a very disciplined programmer. “You know she got her degree from Franklin!” There was no need to explain he meant Franklin University. “She developed in small increments, used an object-oriented paradigm and a ‘Test Driven Development’ approach.”

Mr. Snake pulled a folder from his drawer and slid it across the desk. “Here is where things stand,” motioning to the folder. “She was developing a simple Station class to model one of our ‘SOUP’ stations.” You glance through the folder to see that the Station class has four instance fields – a description, price per gallon, and a latitude and longitude. The comments indicate the latitude and longitude together provide the location of the station.

As you scan the Java code, your eyes widen. The CIO notices your discomfort and confirms, “Grace left the company before completely implementing the Station class. It is mostly the normal object-oriented, Java stuff – a couple of constructors and the normal getters and setters.”

You see a constructor without parameters and another constructor with parameters. There are several methods – getDescription, setDescription, getPricePerGallon and others – that confirm the CIO’s assessment.

“What is the price per gallon?” you ask. “Is the biofuel sold in gallons and this is the price of the biofuel per gallon – or is this the price of an energy-equivalent gallon of gas?”

Mr. Snake beams at you, “Great question!”

You think to yourself how glad you are that you did homework on alternate fuels, as the CIO continues, “The price per gallon is that of the alternate fuel – which is liquid and sold in gallons, but did you see there is a method for obtaining price per gallon of a GGE?”

Indeed, there is a method called calcPricePerGGE in the Station class. “GGE?” you ask.

Snake answers, “A gallon of our alternate fuel may have more or less energy than a gallon of gasoline. The price per GGE is how much you would have to pay for a gallon of alternate fuel equivalent in energy to that of a gallon of gasoline.”

“I see,” you respond, “and the calcDistance method obtains the distance between the Station and a target location whose latitude and longitude are given as parameters.”

Mr. Snake is clearly pleased with your questions. “Yes, but computing distances between locations given in terms of latitude and longitude is a tricky problem, so Grace warned me that she was going to use a rough approximation in this version of Station.”

As you leaf through the folder you find these notes:

**With latitude and longitude values in degrees, approx. distance in miles:**

**sqrt(x \* x + y \* y)**

**where**

* **first location is [lat1, lon1]**
* **second location is at [lat2, lon2]**
* **x = 69.1 \* (lat2 - lat1)**
* **y = 53.0 \* (lon2 - lon1)**

**Note: distance errors may be 10 percent or more!**

You scratch your head. The CIO notices, and offers, “Grace was pretty far along with this. The day before she left, we reviewed her work and agreed to the instance fields of the Station class, as well as the names, parameters, return types, and function of each of the methods. Don’t change any of those - you just need to implement the methods and use the formula for distance she has in her notes. Grace was meticulous in developing a test class using the JUnit framework for every class she works on. She completed the StationTest class, and we reviewed that, too. She also showed me a StationDemo class she wrote to demonstrate Station. StationDemo contains only one method – a ‘main’ that creates a couple instances of Station and calls on their methods.”

Snake looks at his watch, “I suggest you go study what Grace has done and then get started on finishing out the Station class. I need it in two weeks.”

A bit stunned, you smile weakly. Snake stands and offers you his hand, “Again, welcome – we are pleased to have you at Snake Oil.”

You gather the folder and turn to go, but as you reach the door, Mr. Snake remembers something, “Oh, and one more thing. Before she left, Grace told me there were some syntax errors in the last version she compiled, but she did not have time to fix them. Also, she was aware there might be logic errors in code she had written.”

“Logic errors,” you inquire?

“Yes. She stressed that the JUnit test method she wrote in StationTest was fine, but methods in Station she tested did not produce the desired result. I wish I could remember which methods – just be sure to look for them.”

You nod and smile again, tightly gripping the toy snake in your hand as you leave.

# Details

When you return to your desk, you find some additional details in the notes Grace has left behind. The first answers a question you were pondering as you walked back to your office. You know the implementation of a constructor should assign initial values to all of the instance variables of a class. But what should you do if there are no parameter values for a constructor?

**“Default” values for an instance of Station:**

**- latitude and longitude should be 0 0.0**

**- price per gallon set to ~~0~~ 0.0**

**- description set to “NO DESCRIPTION PROVIDED”**

Mr. Snake had not made clear what type of alternative fuel was available at a station and how to convert its price per gallon into price per “Gasoline Gallon Equivalent.” You find the following in Grace’s notes:

**To convert the price per gallon of the alternative fuel to the price per GGE**

**Multiply the alt fuel price per gallon by 1.41**

# Assignment

1. A BlueJ *starter* project will be provided to you. It will contain three classes – Station, StationTest, and StationDemo. The version of the Station class provided has most of the methods completed to help get you started and provide the syntax and approach to use for the methods you will code. You will need to find and fix any syntax errors and complete and compile the Station class – following any guiding comments within the code and meeting requirements described in this write up.
2. After fixing the syntax errors, run the code in BlueJ (see item 5 below). This will reveal logic errors that need to be corrected in the Station class. Note that in StationTest, all the tests (except one) are valid tests (i.e., if a test fails, there is likely a logic error in Station class).

Read through StationTest to better understand Junit tests. You will notice there are a couple of test methods in which you are to add a test case or fix the expected value in the test case. There are Java comments in the code indicating where you need to add or fix.

1. You must use the provided BlueJ starter project and develop your solution using the BlueJ IDE.
2. Do NOT change the name, return type, or parameter order/type of any of the class or test class methods. These are used by Web-CAT to grade your submission, and you will end up losing points if they are changed.

Each class (in this assignment, Station) constructor and method should be tested by a unit test method in the test class (in this assignment, StationTest). Besides serving to verify that your code is accomplishing what you intended, examining the unit test methods will help you understand how the objects should behave (how the class methods create, access, and mutate the object). That is, writing the test methods first helps guide your class coding efforts.   
  
Follow these steps to complete this assignment:

* 1. Find and correct any syntax errors so that the classes in the project compile cleanly.
  2. Review the first JUnit test method, completing the code if necessary. Note that proper testing includes making manual calculations to verify that expected values for a method action are equivalent to actual results.
  3. Once you feel the unit test method is a valid and robust test of the method’s expected behavior, review the corresponding class constructor or method and complete the coding if necessary.
  4. Run the JUnit test for the method. If it does not pass, review and correct the code for this method.
  5. When the test passes, continue in like manner for the next and subsequent methods, iteratively completing tests and developing your class or classes. All class methods must be tested.

1. All classes must compile cleanly.
2. Check your programming style using the “Checkstyle” tool provided within BlueJ (Tools 🡪 Checkstyle).
3. Document the overall project in the README file as directed in the Action Items for the Lab.
4. Submit your completed lab to Web-CAT using BlueJ (Tools 🡪 Submit) and review the Web-CAT results for errors. Repeat the above steps as needed to resolve any errors.
5. The class StationDemo included in the project file demonstrates the use of the Station class, calling on its constructors and methods and displaying results. While it supports your understanding of the lab, it is not part of the actual solution. You can modify it if that is helpful to you, but the driver class will not be examined by Web-CAT nor graded by your instructor.