Objectives and instructions

You will learn about classes, objects, and methods in object-oriented programming using Java. In this project, we have designed the classes and we ask you to implement the classes. You must follow our specifications. As you read about the classes that you will implement, also make sure that you understand the reasoning behind the design, as described on the first pages. Read this entire document and the provided skeleton code before starting to write the program! Complete the steps in order to avoid adding complications atop unfinished or buggy code.

All about ZigZags

Dr. Fan is working on a complicated research project that involves a lot of 3D modeling. Specifically, she is working with zigzags, which are sequences of connected points. To make her life easier, you need to write some 3D modeling classes that she will then use to create models and display them using her renderer (graphics). There are three classes to be implemented:

- ZigZagSystem: a collection of ZigZags
- ZigZag: an individual ZigZag, or a collection of points (in a ZigZag)
- ZigZagPoint: an individual point in a ZigZag

First, let’s get a visual idea of the system. A point in 3-d space is represented by three coordinates, x, y, and z. When we deal with a point in a ZigZag, we also need to know where the next point is. Therefore, a ZigZagPoint object, drawn below, needs to have four pieces of data.

![ZigZagPoint](x, y, z
next)

Now, let’s draw a collection of points that form a ZigZag. Notice that the points are linked. In programming, we call such a collection of linked data a linked list.

![ZigZag](x, y, z
next)

When you look at the above collection of points as one whole unit, we call it a ZigZag. What is an important piece of data in the ZigZag object? The first point in the ZigZag! Since the points are linked (see diagram above), once you have access to the first point in the ZigZag, you can access the information in any other point in the same ZigZag. We need to deal with many ZigZags, so we want to look at many ZigZags as one collection. We can use the idea of linked list again. That is, we’ll put one other piece of data in the ZigZag object for
linking to the next ZigZag object. Below, we draw a ZigZag to denote a ZigZag object. The two important pieces of data for the ZigZag object are firstpoint and nextzigzag.

Now we can have a collection of ZigZags that are linked, drawn below. The collection is our ZigZagSystem with just one important piece of data, the first ZigZag in the collection.

Step A

First, we need to write the ZigZagPoint class, which defines the most basic object in this program. This class will need to store the xyz coordinates of the point it represents as instance variables. All coordinates in this project will be integers. A ZigZagPoint object also acts as a node in a linked list, so each ZigZagPoint also has an instance variable that refers to the next ZigZagPoint in the list. The basic functionality required by ZigZagPoint at this stage is:

1. Accessor (getter) methods for the instance variables (x, y, z, and next)
2. Modifier (setter) method for variable next: Sets variable next to refer to a passed in ZigZagPoint.
3. Constructor: Takes in three coordinates as parameters and sets up the ZigZagPoint object.
4. Method toString() that returns the xyz coordinates in string form. For instance, a ZigZagPoint where x is 10, y is 10, and z is 20 will be represented in string form as “(10, 10, 20)”.

Testing:
Write a static method main in class ZigZagSystem to test your implementation of class ZigZagPoint. You should test each method immediately after implementing it. How? In method main, write code to instantiate a ZigZagPoint object (you can do this even before you write a constructor). Then write code to call the method that you have just written and output the result. Now compile and run the program. Is the output correct? If not, analyze your code and fix the error(s). Test one method at a time! That way you know exactly where the errors occur. Leave the test code in method main—it is a documentation of your testing. As you add more test code, you can suppress the previous tests simply by “commenting out” the old tests—don’t delete the tests.

Step B

Write the ZigZag class. A ZigZag object represents a linked list of ZigZagPoints and acts as a node in a linked list of ZigZags represented by ZigZagSystem. Hence it needs two instance variables, one referring to the first ZigZagPoint in the ZigZag, and the second to the next ZigZag object in the ZigZagSystem. The basic functionality required is:
1. Accessor (getter) methods for the instance variables (firstPoint and nextZigZag)
2. Modifier (setter) method for variable nextZigzag
3. add(ZigZagPoint p): Adds the passed in ZigZagPoint to this ZigZag.
4. remove(ZigZagPoint p): Removes the passed in ZigZagPoint from this ZigZag.
5. toString(): If the ZigZag contains three points (3,3,3), (4,4,2), (5,5,3), then the string representation of the ZigZag will be “(3, 3, 3) - (4, 4, 2) - (5, 5, 3)”.

Have you been testing your code? Keep testing by writing more code in method main in class ZigZagSystem. Specifically, you need to instantiate a ZigZag object and call the methods in class ZigZag one at a time.

Step C

Implement the ZigZagSystem class. This represents a linked list of ZigZag objects and hence has an instance variable referring to the first ZigZag object in the system. Basic functionality required:

1. Accessor (getter) method for the instance variable (first)
2. add(ZigZag z): Adds the passed in ZigZag to the ZigZagSystem.
3. remove(ZigZag z): Removes the passed in ZigZag from the ZigZagSystem.
4. toString() that returns a string representation of the system. This will simply print out the string representation of each ZigZag in the system on a separate line.

More tests... In method main in class ZigZagSystem, create a ZigZagSystem, add ZigZags to it, and check if the functionality you have built works correctly.

Step D

We’ll add some advanced functionality to the classes.

1. Overload the ZigZagPoint constructor such that there are only two passed parameters, representing the x and y coordinates. In the created ZigZagPoint object, the z coordinate is set to zero. The method header for this overloaded constructor is not in the skeleton code. You need to write this entire constructor yourself.
2. Add a numberOfZigZags() method to class ZigZagSystem that returns the number of ZigZag objects currently in the system.
3. Add a getDistance(ZigZagPoint p) method to class ZigZagPoint that takes in another ZigZagPoint and returns the distance between them.
4. Add four new methods to class ZigZag:
   a. ZigZagLength(): Returns the total length of the ZigZag, where length is defined as the sum of the distances between connected points.
   b. avgLegDistance(): Returns the average distance between two connected points in the ZigZag.
   c. join(ZigZag z): Appends the passed in ZigZag to the end of this ZigZag.
   d. reverse(): Reverses the ZigZag.

At this point you have a working ZigZag modeler!
Notes, before you start programming!

1. Skeleton methods have been provided for all the methods that you need to code. These do not do anything right now, and return 0 or null if a return value is required. Remember to change these to return meaningful values.

2. You can add extra methods, but all the methods described above for which skeleton code is provided have to be implemented, and their names, parameters and return types must match exactly.

3. Read the comments given above each skeleton method for more information on what the method does, and how it’s expected behave.