GMU Spring 2023 – CS 211 – Exercise 3

DUE DATE: MONDAY, APRIL 10TH, 11:59PM

Objective

The purpose of this assignment is to practice error handling, exceptions, testing and debugging. You will first implement a few classes that solve a programming task, and then you will write a small tester to test your code. No tester is going to be provided for this assignment.

Overview

1. Define a class named **`Point`** that represents a point in a 3D space.
2. Define a class named **`Face`** that represents an equilateral triangle.
3. Define a class named **`Tetrahedron`** that represents a regular tetrahedron.
4. Define exception classes **`FaceException`** and **`TetrahedronException`**, which shall be used to signal improper values and impossible actions relating to the above three classes.
5. Write a class named **`E3Tester`** that performs unit testing for the above classes.
6. Submit your assignment through Gradescope

Background

We will create a class that represents a regular tetrahedron and is able to calculate certain geometrical properties of it (area, volume, etc.). But to define a tetrahedron, we need to define the faces of the tetrahedron first. And to define a face of a tetrahedron, we need to define points in a 3D space (check the provided wikipedia links for more info about tetrahedrons and faces).

Once we build the code, we need to test its logical correctness with unit testing. Also, sometimes we may accidentally try to feed bad values into a method. The code in these cases should raise an exception and the unit tester should be able to verify that the code is properly signaling the illegal operations via exceptions.

Instructions

1. For this exercise only, you're allowed to **discuss and share ideas publicly on Piazza**. You still can't share any code though, just ideas and thoughts. And nowhere outside Piazza (it's still a violation of the HC if you discuss this assignment in other channels).
2. In the **`E3Tester`** class that is using the JUnit Framework, you're allowed to import and declare anything you want

3. In the other five classes:
    a) You are not allowed to import anything (or use the fully qualified name to bypass this restriction).
    b) You're not allowed to add any public methods other than the ones listed in the specs below.
    c) You're not allowed to add any fields (not even private) other than the ones listed in the specs below.

Grading

Grading has two parts with a 50% weight each:

1. To grade your solution we will use our own automated tester (no manual grading)
2. To grade your tester, we will use a buggy solution and will count the number of bugs your tester can detect. The more errors your tester finds the higher your score will be.

Submission

Submission instructions are as follows:

1. Upload the six **`.java`** files to the following Gradescope folder (do not zip the files)

   https://www.gradescope.com/courses/498183/assignments/2788679

2. Download the files you just uploaded to Gradescope and compile/run them to verify that the upload was correct

3. Make a backup of the files on OneDrive using your GMU account

   If you skip steps 2 and 3 and your submission is missing the proper files, there won't be a way to verify your work and you will get zero points.

Task A

You must implement the following 5 classes for the solution of the task:

`FaceException`

Making an exception class is extremely simple: we would only need to extend our class from the appropriate type of parent exception, and then create constructors which pass on the initialization task to the parent constructors.

The `FaceException` is used to signal impossible operations related to the **`Face`** class. It has only three requirements:

- It must derive from `ArithmeticException`
- It must have a default (non parametric) constructor.
- It must have a constructor which takes an error message as a parameter. It can simply pass this error message up to the parent constructor.

`TetrahedronException`

The `TetrahedronException` will be used to signal impossible operations related to the **`Tetrahedron`** class. Like the earlier `FaceException`, the `TetrahedronException` will have two constructors. However, unlike the other exception, the `TetrahedronException` should derive from `Exception` instead of a `ArithmeticException`.

`Point`

It represents a point in a 3D space and has the following members:

- Fields for three coordinates: **`x`**, **`y`**, **`z`**
- A constructor (parameters are in the same order as above) that initializes the above coordinates. Coordinates cannot be negative numbers. If a coordinate is negative the constructor raises an `ArithmeticException`. For example, if the value passed for **`y`** is -1.4, the exception message will say `Invalid value -1.4 for coordinate y`
- An override of method `toString` that returns a string in the form `{x,y,z}` e.g. **`{1.4,7.91,123.07}`**
- An override of method `equals` that returns true only if all the coordinates of the Point are equal to another Point's coordinates.
- A method `public double distance(Point other)` that calculates the Euclidean distance between two points.

Face

It represents a face of a tetrahedron (i.e. an equilateral triangle) that has the following members:

- Fields for three points: **a**, **b**, **c**
- A constructor that initializes the three points of the face
  Warning: The three points can be in any order!
  The face cannot have a zero area. If this happens, the constructor must raise a `FaceException` with the message `A face can't have a zero area`
  If the three points do not form an equilateral triangle, the constructor raises a `FaceException` with the message `Points must be equidistant`
- An override of method `toString` that returns a string in the form `[a-b-c]` e.g. `[{2.0,13.401,7.88}-{12.1,0.01,0.06}-{0.0,2.0,3.0}]`
- An override of method `equals` that returns true only if all the points of the Face are equal.
- A method `public boolean adjacent(Face other)` that returns true only if the Face has a common edge with another Face.
- A method `public double edge()` that calculates the length of the edge of the Face.
- A method `public double area()` that calculates the area of the Face.

Tetrahedron

It represents a regular tetrahedron that has the following members:

- Fields for four faces: **a**, **b**, **c**, **d**
- A constructor `Tetrahedron(Face a, Face b, Face c, Face d)` that initializes the four faces of the tetrahedron, and a second constructor that creates the tetrahedron based on four points, `Tetrahedron(Point a, Point b, Point c, Point d)`
  Warning: In either version, the four parameters can be in any order!
  The constructor must validate the inputs and, in case the four parameters do not form a valid tetrahedron, it should raise a `TetrahedronException` and pass the message `The four faces do not form a tetrahedron`
  Warning: You must reuse instead of rewriting code.
- A method `public double area()` that calculates the surface area of the tetrahedron
- A method `public double volume()` that calculates the volume of the tetrahedron

Task B

You must implement the following class for testing a solution:

`E3Tester`

You must write your own tester. A template is provided to help you get started. Be reminded that an empty test will naively “pass” because no assert statement has failed. This doesn’t mean that the tester is correct! You must fill in the template with actual (non empty) tests before you start testing your solution.

Assuming that your solution passes all the tests of your own tester, a question that arises is how good your tester is. Did you pass the tests because the solution is correct or because the tester is unable to detect any errors? To verify that your tester is good, make a backup of your solution and then start introducing bugs to your solution. For every bug you create, use your tester to check if it can detect the error. The more bugs it detects the better the tester is.