Pseudocode for Triangle.java

* Global variables:
  + side1 to hold the triangle’s first side
  + side2 to hold the triangle’s second side
  + side3 to hold the triangle’s third side
* Triangle() blank constructor
  + Side side1, side2, and side3 to 1.0
* Triangle(double, double, double) constructor
  + Set side1 to what was passed through
  + Set side2 to what was passed through
  + Set side3 to what was passed
* Create setters and getters for side1, side2, and side3
* getArea() method
  + Use heron’s formula to get the area:
  + Sqrt(s(s-a)(s-b)(s-c)) = Area and s is (a+b+c)/2, and a, b, and c is the sides of the triangles
  + Return the area
* getPerimeter() method
  + Formula for perimeter is a + b + c where a, b, and c are the lengths of the triangle

Pseudocode for Driver.java

* Variables
  + side1 to hold the length of the first side
  + side2 to hold the length of the second side
  + side3 to hold the length of the third side
  + color to hold the color of the triangle
  + filled to hold if the triangle is filled
  + choice to hold the user’s choice to see if they want the triangle filled
  + triangle to hold the Triangle object created afterwards
* Ask the user to enter the length of side 1 of the triangle
* Record the answer in side1
* Ask the user to enter the length of side 2 of the triangle
* Record the answer in side2
* Ask the user to enter the length of side 3 of the triangle
* Record the answer in side3
* Ask the user to enter the color of the triangle
* Record the answer in color
* Ask the user if the triangle is filled or not with a yes or no
* Use a switch statement to check if the user pressed yes or no and set filled appropriately.
* Create a new triangle object with the sides that user put
* Set the triangle’s filled property by calling it and setting it to what the user wanted
* Set the triangle’s color to what the user put
* Show the user the triangle’s properties

Test Cases:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Cases** | **Input** | **Expected Result** | **Actual Result** | **Did it pass?** |
| Case 1 | Side1: 7  Side2: 8  Side3: 9  Color: blue  Filled: y | Area: 26.83  Perimeter: 24  Color: blue  Filled: true | Area: 26.83  Perimeter: 24.00  Color: blue  Filled: true | Y |
| Case 2 | Side1: 4  Side2: 4  Side3: 4  Color: red  Filled: n | Area: 6.93  Perimeter: 12.00  Color: red  Filled: false | Area: 6.93  Perimeter: 12.00  Color: red  Filled: false | Y |
| Case 3 | Side1: 33  Side2: 39  Side3: 46  Color: purple  Filled: yes | Area: 631.54  Perimeter: 118  Color: purple  Filled: true | Area: 631.54  Perimeter: 118.00  Color: purple  Filled: true | Y |

Screenshots: A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

UML:

A screenshot of a social media post

Description automatically generated

Lessons learned:

I learned how to work with Abstract classes and how powerful they are in code. It is very good to use abstract classes when creating a library or some code for others to use. Abstract classes are basically templates that serve a certain purple and in the case of GeometricObject, its purpose was to make sure every one of its subclasses had a getArea() method and a getPerimeter method.

Check list

|  |  |  |  |
| --- | --- | --- | --- |
| **#** |  | **Y/N** | **Comments** |
|  | **Source java files** | **Y** |  |
|  | **Compressed files:** | **Y** |  |
|  | FirstInitialLastName\_Project10\_Moss.zip | **Y** |  |
|  | FirstInitialLastName\_Project10\_doc.zip | **Y** |  |
|  | **Program compiles** | **Y** |  |
|  | **Program runs** | **Y** |  |
|  | **Checklist is completed and included in the Documentation** | **Y** |  |
|  | **Documentation file:** | **Y** |  |
|  | **Comprehensive Test Plan** | **Y** |  |
|  | **Screenshots based on Test Plan** | **Y** |  |
|  | **UML Diagram** | **Y** |  |
|  | **Algorithms/Pseudocode** | **Y** |  |
|  | **Flowchart** | **Y** |  |
|  | **Lessons Learned** | **Y** |  |