Antonio Scalfaro

CMSC 335

Project 1 – Shapes

**Lessons Learned:**

The Shapes program proved to be a fairly straightforward and generally simple program to design and code. I got a chance to sharpen my abilities with data structures and composing classes using inheritance. I was able to use the HashMap data structure implementing the Runnable interface to help select the correct method to create the desired shape selected by the user. This was new for me and took a few tries to get right, mainly because I wanted to pass in my Scanner. I found that Runnable wouldn’t allow that without first making the function input a lambda function that would then call the desired function with the Scanner as an argument. This was out of my comfort zone in terms of data structures and interfaces, but that proved to be worth it in the grand scheme of the project. I decided to use this to avoid using a large chain of if/else or switch statements and was ultimately happy with the end result and cleanliness of the code.

This program contains 13 classes (shown in UML diagram below, Figure 1.1, 1.2, & 1.3) that share the same package, allowing them to be compiled all together. The Main class contains the main method, and the program can be run from this class. The program behaves in a similar fashion to the demo in the Project 1 pdf, it will prompt the user with a menu to pick a shape to create or exit the program. It will continue to prompt the user until a valid response is input, then will ask the user for the dimensions required for the shape. The area (for 2D shapes) or volume (for 3D shapes) will automatically be output. The program asks the user if they would like to continue, if yes, the user menu is displayed again, if not the exit message is displayed, and the program terminates. All inputs are validated for appropriate input and the Triangle and Torus shapes require extra checking to ensure that the inputs can also create a valid Triangle or Torus. The program passed all test cases found below in Table 1. As a last note, I do apologize for the UML diagram format in Figures 1.1, 1.2, & 1.3. I usually insert the UML diagram using the add-in from draw.io, however I was receiving an error on Word that would not allow the add-in to work so I had to take screenshots. I will also include the pdf in the zipped project folder to see the diagram more easily.

**Figure 1.1:**

A screenshot of a computer

Description automatically generated

**Figure 1.2:**

A screenshot of a computer

Description automatically generated

**Figure 1.3:**

A screenshot of a computer

Description automatically generated

**Table 1:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test #** | **Description** | **Screenshot** | **Pass/Fail** |
| **1** | Initiate program, select 10 from user menu to terminate program with no shape creation | Figure 2 | Pass |
| **2** | Initiate program, test user menu catches invalid input. Select 1 to create Circle, check invalid data is caught for radius. Select ‘y’ to continue, select 2 to create Rectangle, check invalid data is caught for both fields. Select ‘n’ to terminate. | Figures 3.1, 3.2, 3.3 | Pass |
| **3** | Initiate program, test Square (check invalid data), when prompted for continue submit invalid data (anything other than “N”, “n”, “y”, “Y”). Continue to Triangle (test invalid data, test invalid triangle). Terminate program | Figures 4.1, 4.2, 4.3 | Pass |
| **4** | Initiate program, test Sphere (check invalid data) and Cube (check invalid data). Terminate program | Figure 5.1, 5.2 | Pass |
| **5** | Initiate program, test Cone (check invalid data) and Cylinder (test invalid data). Terminate program. | Figures 6.1, 6.2 | Pass |
| **6** | Initiate program, test Torus (check invalid data, check invalid Torus). Terminate program. | Figure 7 | Pass |

**Figure 2:**

A computer screen shot of a computer program

Description automatically generated

**Figure 3.1:**

A screenshot of a computer program

Description automatically generated

**Figure 3.2:**

A screenshot of a computer program

Description automatically generated

**Figure 3.3:**

A computer screen shot of a program

Description automatically generated

**Figure 4.1:**

A screenshot of a computer program

Description automatically generated

**Figure 4.2:**

A computer screen shot of a program

Description automatically generated

**Figure 4.3:**

A screenshot of a computer program

Description automatically generated

**Figure 5.1:**

A computer screen shot of a program

Description automatically generated

**Figure 5.2:**

A screenshot of a computer program

Description automatically generated

**Figure 6.1:**

A computer screen shot of a program

Description automatically generated

**Figure 6.2:**

A screenshot of a computer program

Description automatically generated

**Figure 7:**

A screenshot of a computer program

Description automatically generated