**Maiza Falcon Rojas**

**February 13, 2024**

**Assignment: Activity 3 - Interfaces and Polymorphism**

**Class: CST-239**

**Loom Link:** [**https://www.loom.com/share/ff1ac6484ca04299a717d91aee2da51f?sid=71162326-0273-4b8b-a0d0-214c3dd68f10**](https://www.loom.com/share/ff1ac6484ca04299a717d91aee2da51f?sid=71162326-0273-4b8b-a0d0-214c3dd68f10)

**Github Link: https://github.com/maiza02/Activity-3.git**

**Part 1**

**Figure 1: Screenshot of test class output**

A screenshot of a computer program

Description automatically generated

**Figure 2: Screenshot of test class output**

A screenshot of a computer program

Description automatically generated

**Figure 3: Screenshot of test class output**

A screenshot of a computer program

Description automatically generated

**Theory of Operation:**

The output is displayed based on the comparison of ages between pairs of individuals. The compareTo method in the Person class compares the ages of two persons, and the compareAges helper method in the Test class prints whether one person is younger, older, or the same age as the other. The sequence of comparisons in the main method generates output indicating the relative ages of the individuals involved, providing insights into their age relationships.

**Part 2**

**Figure 4: Test class output for Part 2**

**A screenshot of a computer

Description automatically generated**

**Figure 5: Test class output**

**A screenshot of a computer

Description automatically generated**

**.**

**Figure 6: UMl Diagram**

**A screenshot of a computer screen

Description automatically generated**

**Theory of operation:**

Polymorphism is demonstrated in the code through the use of the ShapeInterface and the calculateArea() method. The calculateArea() method is implemented differently in each shape class (Rectangle, Triangle, Circle, and Oval) based on their specific geometric fo.0rmulas. Despite being called through a common interface, the method exhibits polymorphic behavior, executing the appropriate implementation depending on the actual type of the object. This allows for a unified approach to handling different shapes within the array of ShapeBase instances in the Test class.

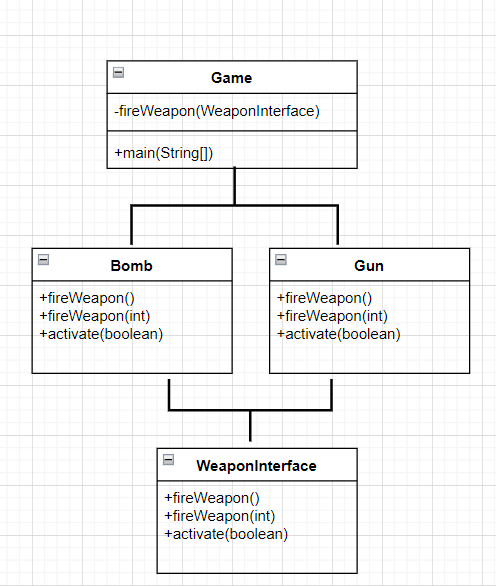
**Part 3:**

**Figure 7: Output for Game Class**

**A screenshot of a computer program

Description automatically generated**

**Figure 8: UML Diagram**

****

**Theory of operation:**

Polymorphism is demonstrated in the code through the use of the WeaponInterface. The fireWeapon(WeaponInterface) method in the Game class accepts any object that implements the WeaponInterface. This allows instances of both Bomb and Gun to be treated uniformly within the for loop in the main method, showcasing polymorphism as different objects, each with their unique implementations of fireWeapon and activate, are processed using a common interface.

**Part 4:**

**A screenshot of a computer program

Description automatically generatedFigure 9: Screenshot of first breakpoint**

**Figure 10: Screenshot of second breakpoint**

**A screenshot of a computer

Description automatically generated**

**Figure 11: Screenshot Variable**

**A screenshot of a computer

Description automatically generated**

**Figure 12: Screenshot Stepping   
A screenshot of a computer program

Description automatically generated**

**Figure 13: Screenshot Inspecting Call Stack**

**A screenshot of a computer program

Description automatically generated**

**Follow up Questions:**

**What was challenging?**

The challenging aspect of the assignment for me was Part 1, where the task “Add an age property of type int to the Person class. Reimplement your compareTo() method to compare a person’s age”. My difficulty arose from my initial attempt to integrate the new code seamlessly into the existing codebase without making necessary adjustments. Upon reflection, I recognized that my approach was suboptimal, as I failed to appropriately modify the code to align with the specified requirements.

**What did you learn?**

I learned how to implement Interface and Polymorphic classes in my code. By implementing Interfaces, I discovered a versatile way to define contracts for classes, ensuring a consistent structure across various implementations.

**How would you improve on the project?**

One way I feel like the program could improve is state if they want us to use the names they provided or make up our own. That is one thing I think can improve because I do not know if I should just make the code exactly like they show or use it as a reference.

**How can you use what you learned on the job?**

The skills acquired in this activity can be directly applied to improve code quality, promote best practices, and contribute to the development of robust and adaptable software solutions in a professional work setting.