**CST-239 Activity 3: Interfaces and Polymorphism**

Batossa Bakouma

May 2025

Grand Canyon University

CST-239: Object-Oriented Programming

Instructor: Komal Chhibber

**Part 1: Person Interface and Comparable**

**a. Theory of Operation:**

In this activity, the Person class was updated to implement both a custom PersonInterface and the Comparable interface. The walk and run methods demonstrated interface behavior, while the compareTo method allowed sorting by either last name or age. Arrays.sort() used the compareTo logic to order objects, showing polymorphism through interface and natural ordering support.

**b. Screenshots:**

* walk-run-output.png
* sorted-by-age.png
* equals-test-output.png

**c. Note:**

JavaDocs for Part 1 are located in topic3-1/doc/

**🔹 Part 2: Polymorphic Shapes**

**a. Theory of Operation:**

Polymorphism was demonstrated by having multiple shape classes (Rectangle and Triangle) inherit from a common ShapeBase class, which implements ShapeInterface. Each shape class overrode the calculateArea() method. The Test class used an array of ShapeBase and was able to call calculateArea() polymorphically on different shapes.

**b. Screenshot:**

* shapes-output.png

**c. UML Diagram:**

* Paste or insert image shapes-uml-diagram.png

**d. Note:**

JavaDocs for Part 2 are located in topic3-2/doc/

**🔹 Part 3: Polymorphic Weapons**

**a. Theory of Operation:**

In this activity, the abstract Weapon class from Activity 2 was replaced with a WeaponInterface to better demonstrate polymorphism. Two classes, Bomb and Gun, implemented this interface and provided their own versions of activate() and fireWeapon() methods. The Game class used an array of the interface type to call shared behavior on both weapon types, proving that polymorphism allows the same method calls to operate differently depending on the object instance.

**b. Screenshot:**

* weapons-output.png

**c. UML Diagram:**

Under UML for part 3

**d. Note:**

JavaDocs for Part 3 are located in topic3-3/doc/

**🔹 Part 4: Using the Debugger**

**a. Screenshots:**

* breakpoint.png
* inspect-variables.png
* step-into.png
* **b. Debugging Summary:**

In this section, I demonstrated how to use the Eclipse debugger. I set a breakpoint at bomb.activate(true); in Game.java, then inspected the bomb and gun variables. I used the Step Into feature to trace into the activate() method and observed the call stack. These tools helped me understand how control flows through method calls at runtime.