**Maiza Falcon Rojas**

**February 06, 2024**

**Assignment: Activity 2: Inheritance**

**Class: CST-239**

Link to Github: <https://github.com/maiza02/Activity-2-.git>

Link to Video: <https://www.loom.com/share/f19cf2ac07d74d22994f2d59070e6be5?sid=31e6de6d-f685-4a81-988e-1fe22b2b6907>

**Part 1.**

**Figure 1. Screenshot of game results**

A screenshot of a computer

Description automatically generated

**Theory of operation for Figure 1.**

The SuperHero class contains methods for attacking opponents (attack), determining if the superhero is dead (isDead), and updating the health based on the damage received (determineHealth). The Game class creates instances of Superman and Batman, initiates the game loop, and prints messages indicating the outcome when one of the superheroes is defeated. The output of the program will display the creation of superheroes, the start of the game, and the result of each attack until one of the superheroes is defeated. The output will indicate the damage dealt in each attack and the remaining health of each superhero. If Superman is defeated, it will print "Batman defeated Superman," and if Batman is defeated, it will print "Superman defeated Batman." The loop continues until one of the superheroes is defeated.

**Part 2.**

**Figure 2: Screenshot of Game Output**

A screenshot of a computer

Description automatically generated

**Theory of operation for Figure 2.**

The output of the program will display the messages from the fireWeapon method indicating the power values passed to each weapon. Specifically, it will print "In Weapon.fireWeapon() with a power of 10" for the Bomb instance and "In Weapon.fireWeapon() with a power of 5" for the Gun instance. The output reflects the behavior of each weapon firing with its corresponding power.

**Figure 3:** **Screenshot of Game Output** A screenshot of a computer

Description automatically generated

**Theory of operation for Figure 3.**

In the Game class's main method, instances of Bomb and Gun are created, and the fireWeapon method is called on each instance with specified power values (10 for Bomb and 5 for Gun). The output of the program will display messages corresponding to the overridden fireWeapon methods in the Bomb and Gun classes, reflecting the specific behavior of each weapon when firing. Specifically, it will print "In Bomb.fireWeapon() with a power of 10" for Bomb and "In Gun.fireWeapon() with a power of 5" for Gun.

**Figure 4:** **Screenshot of Game Output** A screenshot of a computer

Description automatically generated

**Theory of operation for Figure 4.**

In the Game class's main method, instances of Bomb and Gun are created, and the fireWeapon method is called on each instance with specified power values (10 for Bomb and 5 for Gun). Additionally, the overloaded fireWeapon methods without parameters are called for both Bomb and Gun. The output of the program will display messages reflecting the specific behavior of each weapon when firing, taking into account the overridden and overloaded methods. Specifically, it will print messages like "In Bomb.fireWeapon() with a power of 10" and "In overloaded Gun.fireWeapon()" based on the method calls in the Game class.

**Figure 5:** **Screenshot of Game Output** A screenshot of a computer

Description automatically generated

**Figure 6: Gun implementation code**

A screenshot of a computer program

Description automatically generated

**Figure 7: Bomb implementation code**

A screen shot of a computer program

Description automatically generated

**Figure 8: Weapon implementation code**

A screen shot of a computer program

Description automatically generated

**Figure 9: Error** A screenshot of a computer program

Description automatically generated

**Why the error occurred in Figure 9 Explanation.**

The error in the code occurs because the Weapon class is declared as final, and the code is trying to extend it with the Bomb and Gun classes. In Java, a final class cannot be extended, meaning that the code cannot create subclasses for it.

**Figure 10: Error** A computer screen with text on it

Description automatically generated

**Why the error occurred in Figure 10 Explanation.**

The error in the code is related to the method fireWeapon() in the Bomb and Gun classes. The issue is that the code is trying to override the fireWeapon() method in the subclasses (Bomb and Gun) without adhering to the method signature defined in the superclass (Weapon). It takes an int parameter named power. However, in the Bomb and Gun classes, the code is trying to override this method without the required int power parameter.

**Figure 11: Error** A computer screen shot of a program

Description automatically generated

**Why the error occurred in Figure 11 Explanation.**

The error in the code occurs because the fireWeapon method in the abstract class Weapon is declared as abstract, but it has a default implementation. Abstract methods in Java should not have a body or default implementation.

**Part Three.**

**Figure 12: Output of Test Class**

A screenshot of a computer

Description automatically generated

**Theory of operation for Figure 12.**

The code, the Test class creates three Person objects and tests their equality using the == operator and the equals() method. The first two Person objects, person1 and person2, are explicitly created with different values for the first names. Therefore, the output for the == operator comparison will be "These persons are not identical using ==". However, the equals() method is overridden in the Person class to compare the content of the objects, so the output for the equals() method comparison will be "These persons are not identical using equals()".

The third Person object, person3, is created using a copy constructor that takes another Person object (person1) as an argument. Since the copy constructor copies the content of the person1 object, the equals() method comparison between person1 and person3 will result in "These copied person is identical using equals()". Finally, the code prints the string representations of the Person objects using the toString() method. The output will display the first and last names of each person.

**Figure 13: Output of Test Class**

A screenshot of a computer

Description automatically generated

**Theory of operation for Figure 13.**

The code creates three Person objects (person1, person2, and person3) with different names. It tests equality using both the == operator and the equals() method. The equals() method compares the values of the firstName and lastName fields.A copy of person1 is created using the copy constructor, resulting in person3. The code then prints the objects using the toString() method, which has been overridden in the Person class to provide a custom string representation.

**How could Overridden be used in the Milestone project?**

In the Milestone project, overriding the equals() method facilitates efficient management of the Salable Product inventory by enabling comparison of key properties, such as name and description, to prevent duplicate entries. This ensures the integrity of your inventory data. Simultaneously, the overridden toString() method plays a crucial role in presenting a user-friendly interface. It provides a human-readable representation of Salable Products, aiding in the clear display of product details during inventory management and user interactions. By implementing and documenting these methods according to JavaDoc standards, you enhance the project's functionality, usability, and maintainability.

**Description of what the @Override annotation**

The @Override annotation is used to indicate that a method in a subclass is intended to override a method with the same signature in its superclass.It helps catch errors at compile-time by notifying the developer if the annotated method doesn't actually override a method in the superclass. This prevents accidental method name typos or changes in the signature that might lead to unintended behavior.It is considered good practice to use @Override to make the code more readable, maintainable, and less error-prone, especially when dealing with complex class hierarchies and inheritance.

**Part 4.**

**Figure 14: Setting Breakpoints task**

A screenshot of a computer

Description automatically generated

**Figure 15: Inspecting Variables task**

A screenshot of a computer

Description automatically generated

**Figure 16: Stepping task**

A black rectangular object with white text

Description automatically generated

**Figure 17: Inspecting Call Stack task**

A screenshot of a computer

Description automatically generated

**Follow Up Questions:**

**What was challenging?**

I had a difficult time with debugging, this is the first class that has introduced it to me. I am not sure if I even did that part of the assignment correctly.

**What did you learn?**

I learned what debugging is and how to add breakpoint,Inspecting Variables, Stepping Into a Function, Step Over Function, and Step Return from a function, Inspecting the Call Stack.

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

I would like to improve my knowledge of debugging, I am definitely struggling with it and need to improve on it.

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

The ability to design and implement efficient class structures is fundamental to building robust and scalable software systems.