Jason Hatfield

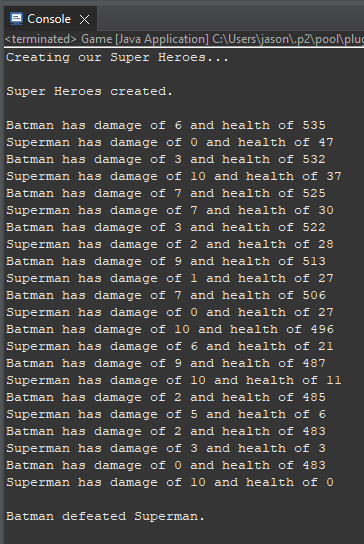
CST-239

Activity 2

11/1/2022

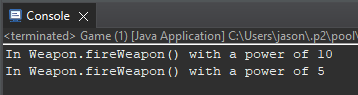
Professor Mark Smithers

### Part 1

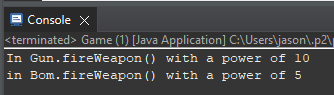
****

When the program is started, the Game driver class creates a Superman and Batman object. Each object randomly generates a health integer from 1 - 1000, which is passed into the attack method of the SuperHero parent class. This class has two children classes, Batman and Superman. Each of those classes takes the health from the parent class and passes that health individually back to output onto the game console.

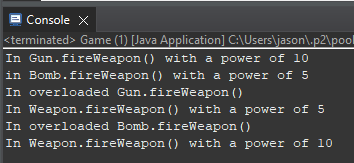
### Part 2



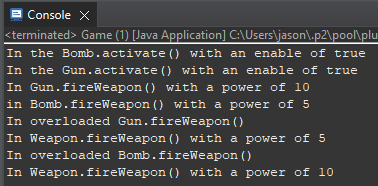
The game driver creates two objects, a bomb, and a gun, with the firepower of 10 and 5, respectively. Although there are no methods in the Gun and Bomb children classes, the output is from the Weapon parent class. This occurs because the children's class inherits the println function from the Weapon parent class. The *extends* keyword in the class signature creates this inheritance bond.

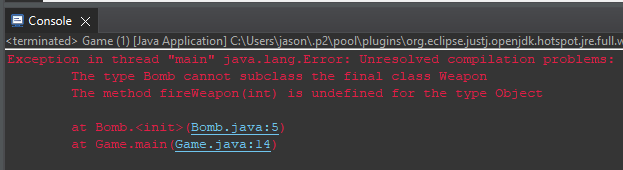
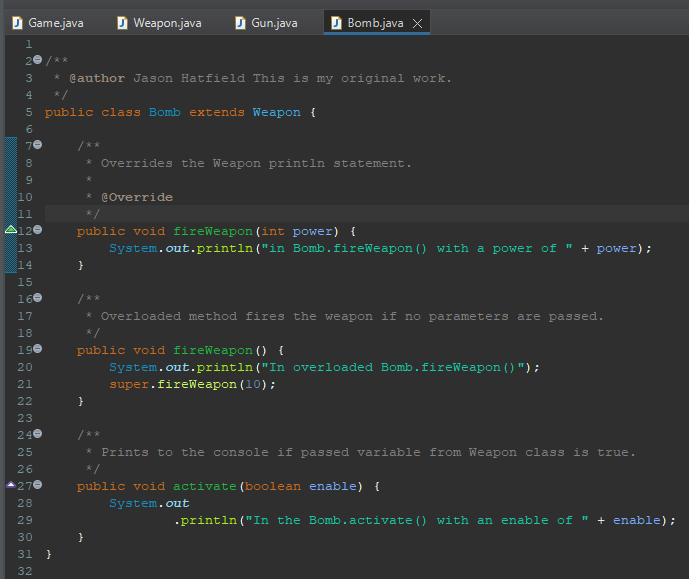
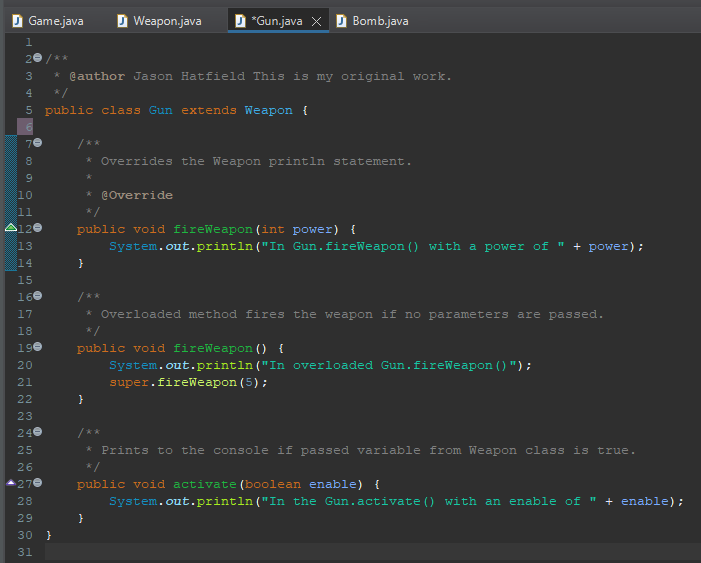
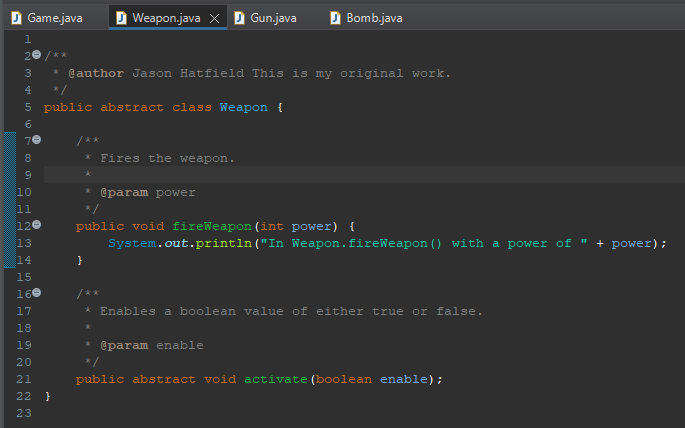


Each child class, Gun and Bomb, has a public method that matches the method in the parent class Weapon. When a child class has the same method signature as the parent class, the contents of this method override that of the parent class. In this instance, both of the children's classes have overridden the Weapon parent class by invoking a different output to the console.

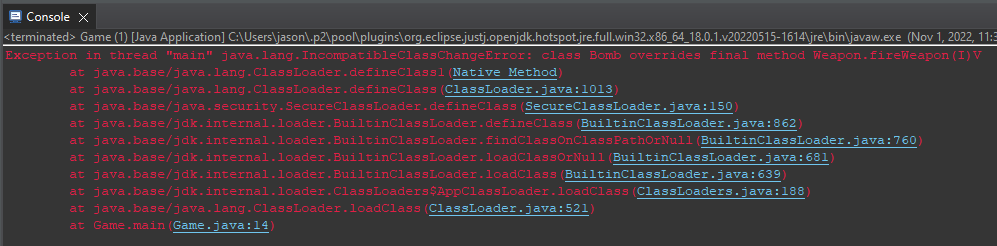


The *overload* method provides an alternative output based on the input in the driver class. The program changes based on the input by adding a new method in the child class that takes a different parameter, or in this case, no. So, when an integer is passed through the parameter, we get one outcome, and when no parameter is passed, we get another.

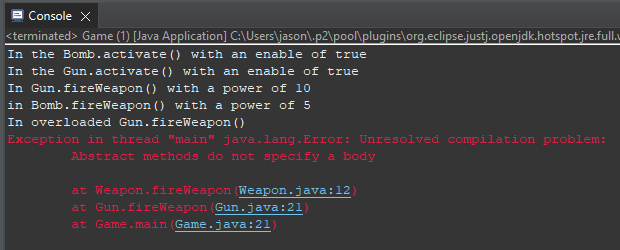




The activate(boolean enable) method must be abstract; therefore, the class must also be abstract. The error occurred because we removed the abstract and replaced this with the final keyword. I experimented and found that Java does not allow a final abstract class, and only one or the other can be present.

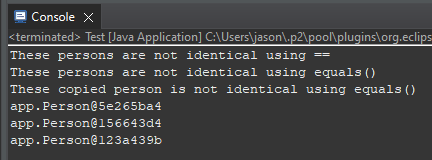


When changing the method signature of *fireWeapon* to final, we receive errors in both the Gun and Bomb subclasses. This error results from the restriction placed on the *fireWeapon* method in the superclass that cannot be overridden by the subclasses.

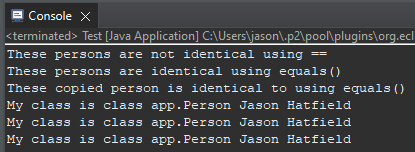


As the compiler went through line 21 of the Gun and Bomb classes, we received an unresolved compilation error. The error states that abstract methods do not specify a body. This error means that a declaration must only be present, not a body. Because there is a body present when the abstract method compiles, the error is thrown.

### Part 3



Although person1 and person2 have the same name; these are not identical because the objects are located at different addresses. Each person is created as a new object which is stored in the Person class, so when they are related in the if statement, the return is not true. To access the same person, we would need to invoke an if statement that compared person1 to person1.



The boolean method in the Person class contains multiple if statements that, if false, will change this.firstName to person.firstName. If the boolean value is this, null, or does not equal other.getClass(), the output will be the println. For this scenario, all three if statements were not true, so a new person object was created, and the name was returned.

As my milestone project grows and becomes more complex, I can see the value in

using the equals() and toString() methods. The toString() method will override the default output and input another line of code. Similarly, the equals() boolean method will allow my milestone project to use if statements once my end user selects various products. Ultimately, I will implement these methods into my program to help with adding products to the user's cart.

There are two reasons for the @Override annotation, but most importantly, this

annotation is a best practice for programmers. The first reason for using this annotation is to ensure the method following the statement is actually overriding the superclass. If the method is not completing this action, a compile-time error will be presented to the programmer. Secondly, the annotation improves readability in the code, which gives either the original or programmers a better idea of the original intentions of the code.

### Part 4

