Activity 2

Jesse Dalton
February 12, 2023

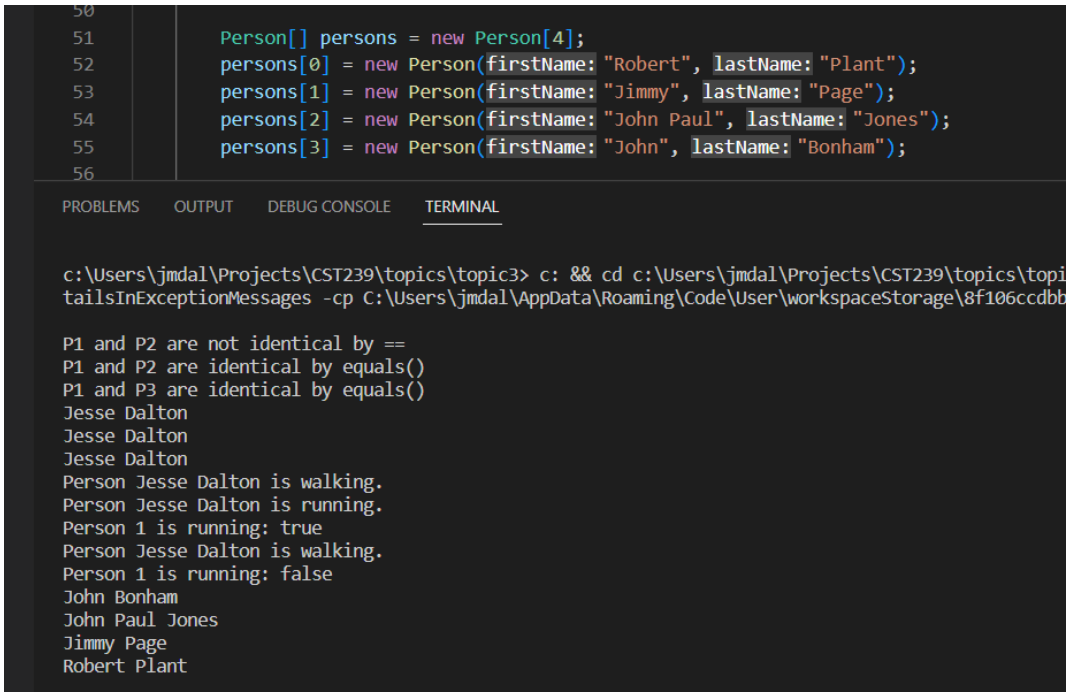
GitHub: https://github.com/jmdalton0/cst239-act3

PART 1: Person Interface

Test class output:

```
P1 and P2 are not identical by ==
P1 and P2 are identical by equals()
P1 and P3 are identical by equals()
Jesse Dalton
Jesse Dalton
Jesse Dalton
Person Jesse Dalton is walking.
Person Jesse Dalton is running.
Person 1 is running: true
Person Jesse Dalton is walking.
Person 1 is running: false
```

Test class output after Person implements Comparable:



Test class output after Person implements Comparable using age:

In the test class, an array of Person objects is created. This array is passed to Arrays.sort(), which is a Java util function that is part of Arrays. Because the Person class implements the Comparable interface, this array can be sorted using Arrays.sort(). The sort method calls the compareTo() method on each Person object to see how they compare to each other, then sorts them. Depending on how the compareTo() method is implemented on Person (age or name), the sort method sorts the Persons.

PART 2: Shapes

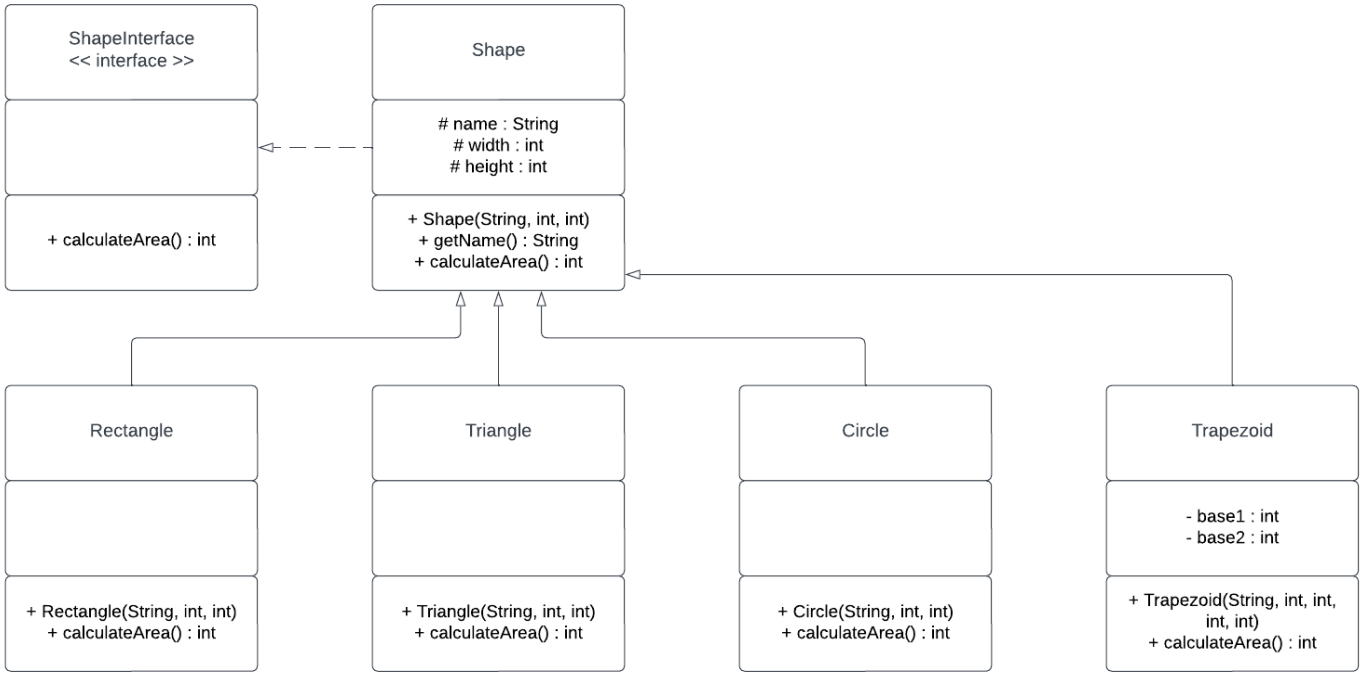
Test Driver Output:

```
This is a shape named Rectanlge with an area of 2000
This is a shape named Triangle with an area of 250
```

Test Driver Output after additional shapes:

```
This is a shape named Rectanlge with an area of 2000
This is a shape named Triangle with an area of 250
This is a shape named Trapezoid with an area of 350
This is a shape named Circle with an area of 78
```

UML Diagram:



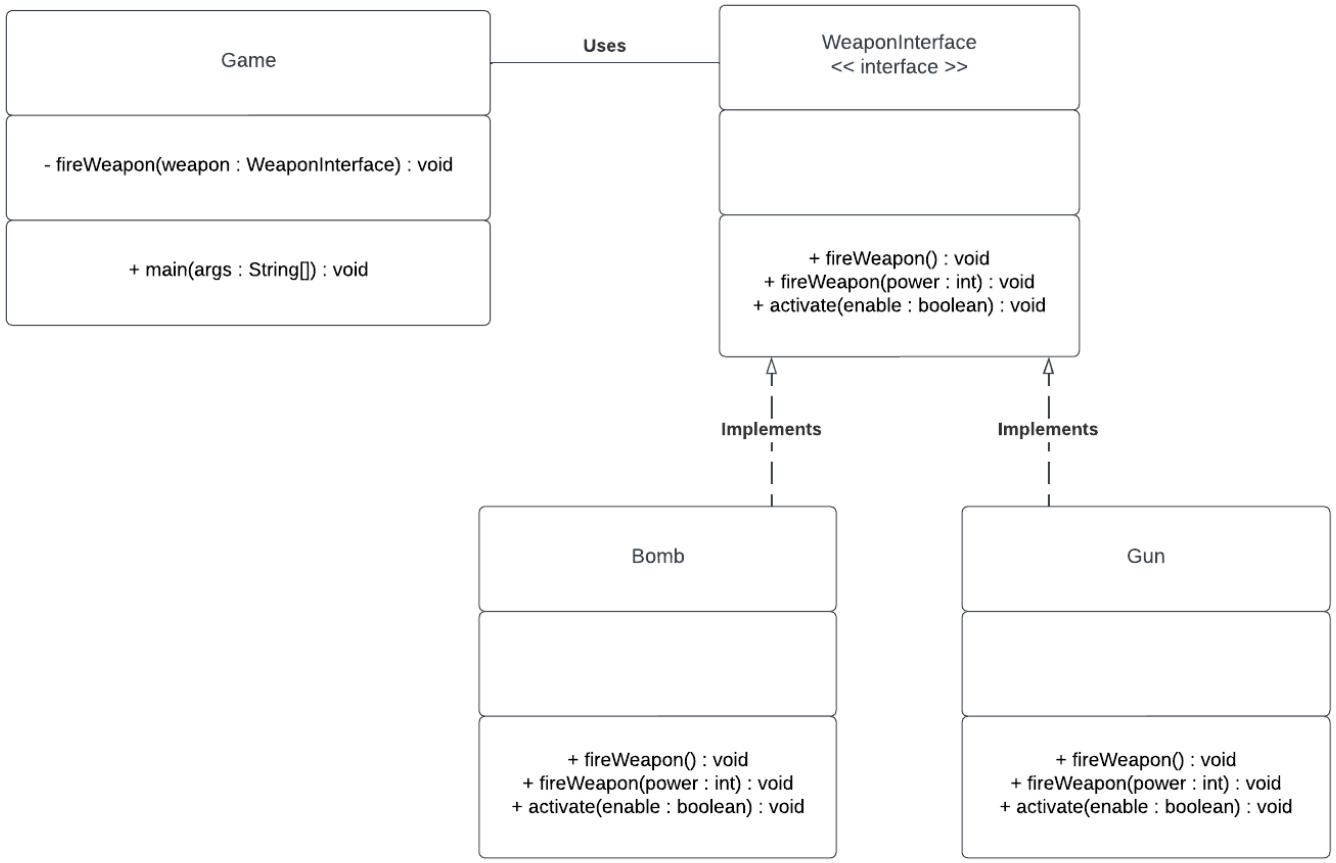
Polymorphism is the ability of classes that inherit from other classes to provide different functionality from the parent class for the same, overridden method, even if the declared type of the object is the parent class. In our example, each Shape in the array found in the test driver is declared as type Shape, which is the parent class. However, the actual type of each individual shape is one of the 4 children subclasses that we created. All of these subclasses implement their own version of calculateArea(). Because of polymorphism, when calculateArea() is called on each of these objects, the correct version is used depending on the actual type of the shape.

PART 3: Weapons

Test Driver Output:

```
In Bomb.activate() with an enable of true
In Bomb.fireWeapon() with a power of 5
In Gun.activate() with an enable of true
In Gun.fireWeapon() with a power of 5
```

UML Diagram:



Polymorphism is the ability of classes that inherit from other classes or implement an interface to provide different functionality from the parent class/interface for the same, overridden method, even if the declared type of the object is the parent class/interface. In our example, each Weapon in the array found in the test driver is declared as type WeaponInterface, which is the parent interface. However, the actual type of each individual weapon is one of the 2 children subclasses that we created. All of these subclasses implement their own version of the three methods found in WeaponInterface. Because of polymorphism, when any of these methods are called on each of these objects, the correct version is used depending on the actual type of the weapon.

PART 4: Debugger