

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:

```
50
51     Person[] persons = new Person[4];
52     persons[0] = new Person(firstName: "Robert", lastName: "Plant");
53     persons[1] = new Person(firstName: "Jimmy", lastName: "Page");
54     persons[2] = new Person(firstName: "John Paul", lastName: "Jones");
55     persons[3] = new Person(firstName: "John", lastName: "Bonham");
56
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
c:\Users\jmdal\Projects\CST239\topics\topic3> c: && cd c:\Users\jmdal\Projects\CST239\topics\topi
tailsInExceptionMessages -cp C:\Users\jmdal\AppData\Roaming\Code\User\workspaceStorage\8f106ccdbb

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
John Bonham
John Paul Jones
Jimmy Page
Robert Plant
```

Test class output after Person implements Comparable using age:

```
52
53     Person[] persons = new Person[4];
54     persons[0] = new Person(firstName: "Robert", lastName: "Plant", age: 20);
55     persons[1] = new Person(firstName: "Jimmy", lastName: "Page", age: 80);
56     persons[2] = new Person(firstName: "John Paul", lastName: "Jones", age: 7);
57     persons[3] = new Person(firstName: "John", lastName: "Bonham", age: 24);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
c:\Users\jmdal\Projects\CST239\topics\topic3> c: && cd c:\Users\jmdal\Projects\CST239\topics\topi
tailsInExceptionMessages -cp C:\Users\jmdal\AppData\Roaming\Code\User\workspaceStorage\8f106ccdbb
John Paul Jones, age 7
Robert Plant, age 20
John Bonham, age 24
Jimmy Page, age 80
```

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

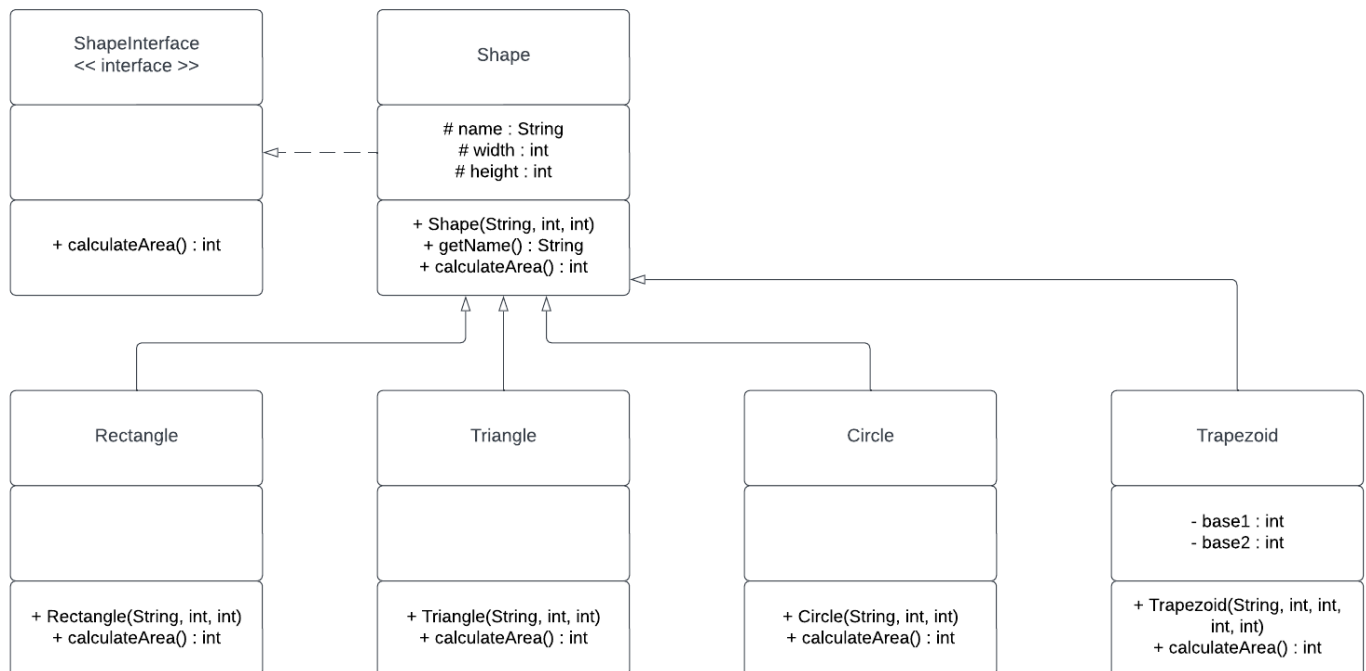
Test Driver Output:

```
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:



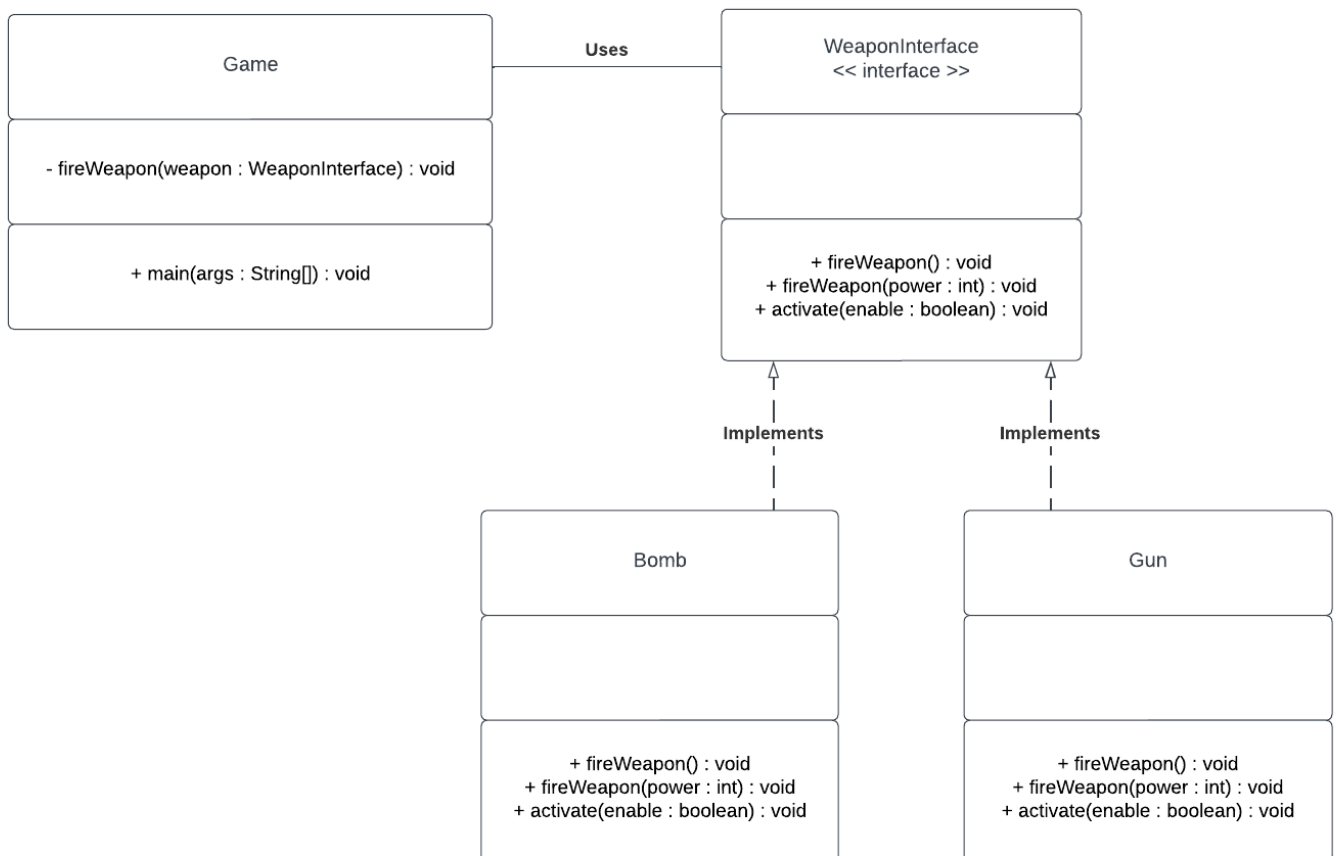
Polymorphism is the ability of classes that inherit from other classes to provide different functionality from the parent class for the same, overridden method, even if the declared type of the object is the parent class. In our example, each Shape in the array found in the test driver is declared as 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

```
Run | Debug
8  public static void main(String[] args) { args = String[0]@9
9  System.out.println();
10
11  WeaponInterface[] weapons = new WeaponInterface[2];
12  weapons[0] = new Bomb();
13  weapons[1] = new Gun();
```

```
8  */
9  @Override
10 public void fireWeapon(int power) { power = 5
11 System.out.println("In Bomb.fireWeapon() with a power of " + power); power = 5
12 }
13
```

VARIABLES

Local

power: 5	1
> this: Bomb@17	2
	3
	4

```
23  /
24 private static void fireWeapon(WeaponInterface weapon) { weapon = Gun@21
25 weapon.activate(enable: true); weapon = Gun@21
26 weapon.fireWeapon(power: 5);
27 }
28 }
29
```

CALL STACK

Thread [main] PAUSED ON STEP

- Game.fireWeapon(WeaponInte
- Game.main(String[]) G...

Thread [Reference ...] RUNNING

Thread [Finalizer] RUNNING

Thread [Signal Disp...] RUNNING

Thread [Attach List...] RUNNING

Thread [Notificatio...] RUNNING

Thread [Common-...] RUNNING