Difference Between the Overloading and Overriding of Methods

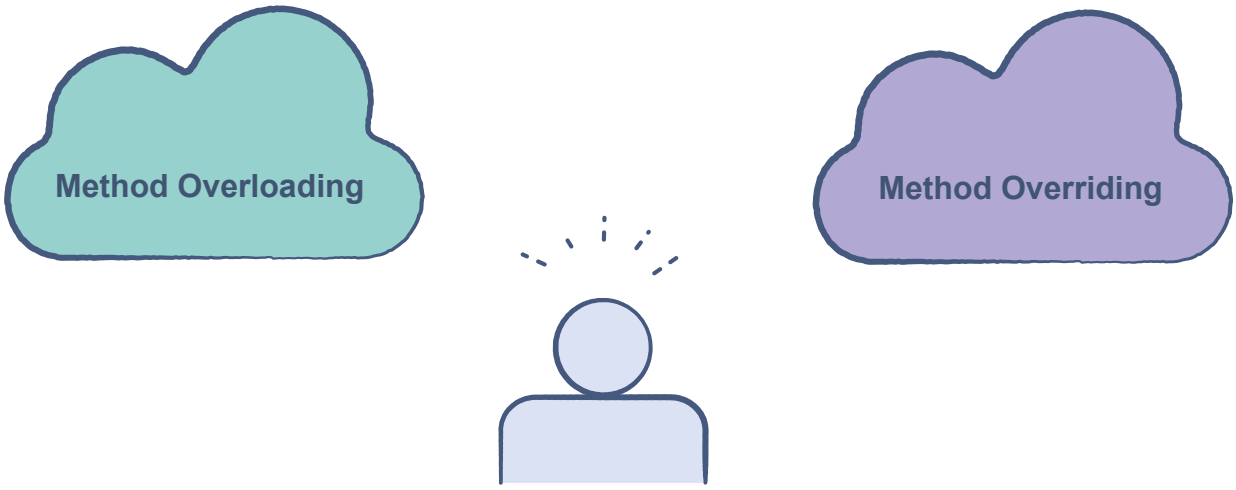
In this lesson, you will get familiar with the differences between method overloading and method overriding.

We'll cover the following

- Method Overloading & Method Overriding
- Method Overloading Example
- Method Overriding Example

Method Overloading & Method Overriding

Method overloading and overriding are two completely different concepts.



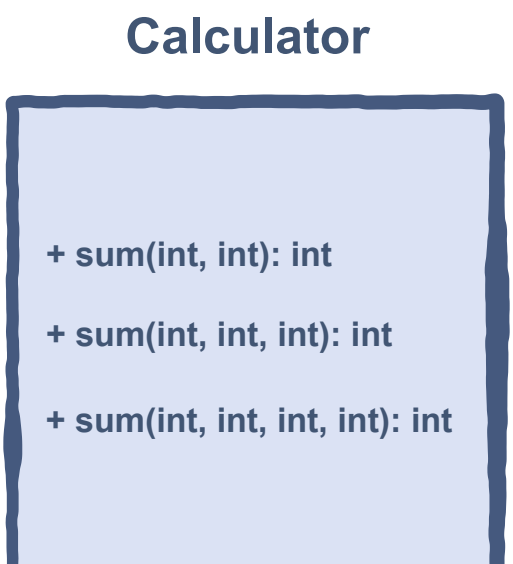
Let’s compare the differences below:

Method Overloading	Method Overriding
Overloading happens at **compile time.**	Overriding happens at **runtime**

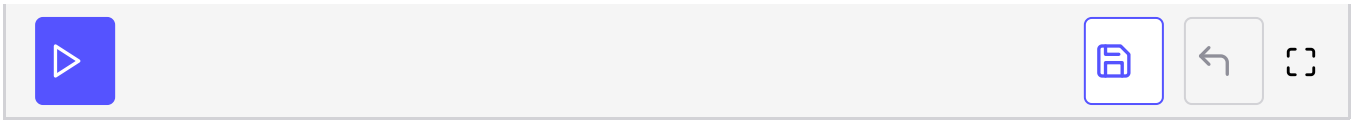
Gives better performance because the binding is being done at compile time.	Gives less performance because the binding is being done at run time.
Private and **final** methods can be overloaded.	**Private** and **final** methods can not be overridden.
Return type of method does not matter in case of method overloading.	Return type of method must be the same in the case of overriding.
Arguments must be different in the case of overloading.	Arguments must be the same in the case of overriding.
It is being done in the same class.	Base and derived classes are required here.
Mostly used to increase the readability of the code.	Mostly used to provide the implementation of the method that is already provided by its base class.

Method Overloading Example

Let's implement the calculator class in java:



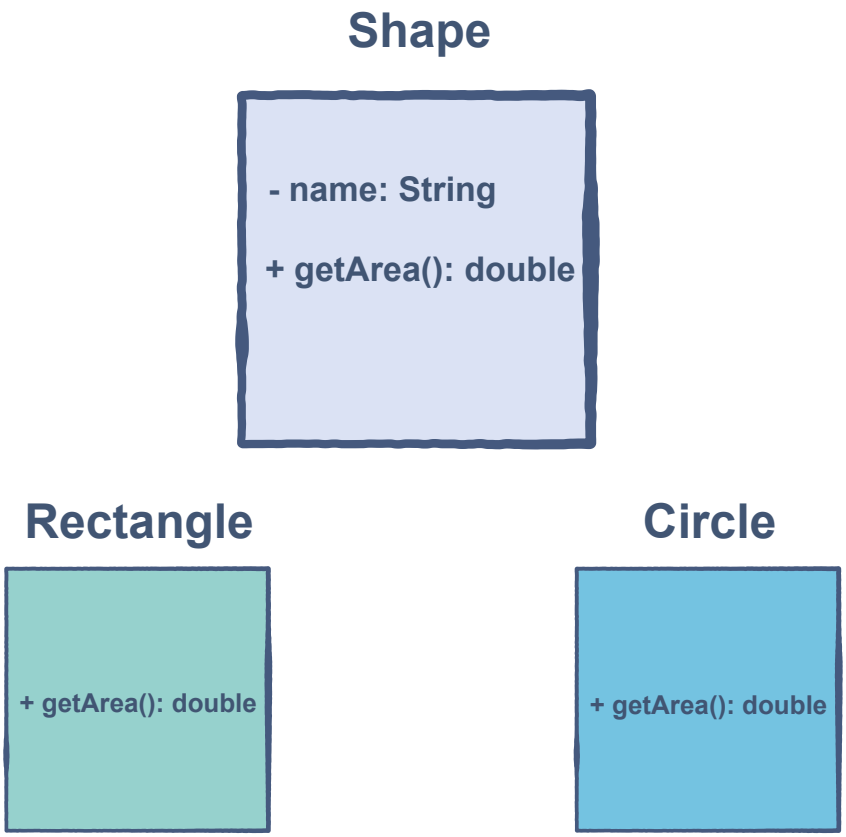
```
//Calculator Class
class Calculator {

  // Add funtions with two parameters
  int add(int num1, int num2) {
    return num1 + num2;
  }

  // Add funtions with three parameters
  int add(int num1, int num2, int num3 ) {
    return num1 + num2 + num3;
  }

  // Add funtions with four parameters
  int add(int num1, int num2, int num3, int num4 ) {
    return num1 + num2 + num3 + num4;
  }

  public static void main(String args[]) {
    Calculator cal = new Calculator();

    System.out.println("10 + 20 =  " + cal.add(10, 20));
    System.out.println("10 + 20 + 30 =  " + cal.add(10, 20, 30));
    System.out.println("10 + 20 + 30 + 40 =  " + cal.add(10, 20, 30, 40));
  }

}
```



Here we have **3** different versions of the `add()` function. The `add()` function is overloaded here.

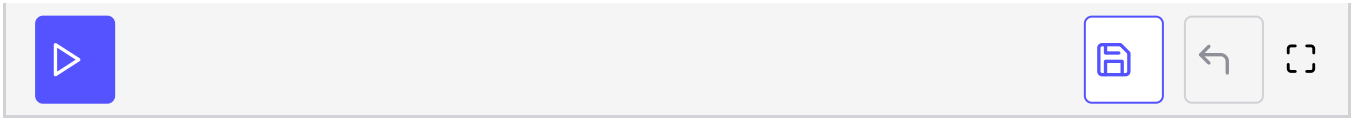
Method Overriding Example

Let's implement the shape class in java:




```
// Shape Class
class Shape {

  public double getArea() {
    return 0;
  }

}

// A Rectangle is a Shape
class Rectangle extends Shape {  // extended form the Shape class

  private double length;
  private double width;

  public Rectangle(double length, double width) {
    this.length = length;
    this.width = width;
  }

  public double getArea() {
    return this.width * this.length;
  }

}

// A Circle is a Shape
class Circle extends Shape {

  private double radius;
```

```
  public Circle(double radius) {
    this.radius = radius;
  }

  public double getArea() {
    return 3.13 * this.radius * this.radius;
  }


  public static void main(String args[]) {
    Shape[] shape = new Shape[2]; // Creating the shape array of size 2

    shape[0] = new Circle(3); // creating the circle object at index 0
    shape[1] = new Rectangle(2, 3); // creating the rectangle object at index 1

    System.out.println("Area of Circle: " + shape[0].getArea());
    System.out.println("Area of Rectangle: " + shape[1].getArea());
  }

}
```



We have a base class, `Shape`, and two derived classes `Rectangle` and `Circle.` Here, the `getArea()` method of the Shape class is overridden in the Rectangle and the Circle class.

We've learned the differences between method overloading and method overriding. In the next lesson, we'll take a look at dynamic polymorphism.