# Polymorphism, Overloading, and Overriding in Java and Object Oriented Programming?

## Polymorphism vs Overloading vs Overriding

Someone asked me the other day, what are the difference between Polymorphism and Overriding in Java and the similar *difference between Polymorphism and Overloading*. After explaining to him personally, I thought to write a blog post about it and here we are. Well, they are not two different things, **Polymorphism** is an object-oriented or OOP concept much like Abstraction, Encapsulation, or Inheritance which facilitates the use of the interface and allows Java program to take advantage of dynamic binding in Java. Polymorphism adds flexibility to your code which makes it more extensible and maintainable.

Polymorphism is also a way through which a Type can behave differently than expected based upon which kind of Object it is pointing. Overloading and overriding are two forms of Polymorphism available in Java. Both overloading and the overriding concept are applied to methods in Java.

Since P`olymorphism` literally means taking multiple forms, So even though you have the name of the method the same in the case of overloading and overriding, an actual method called can be any of those multiple methods with the same name.

Let's see some more details on method overloading and overriding to understand how polymorphism relates to overloading and overriding and how they are different.

Btw, if you are new to Java and Object-Oriented programming then I highly recommend you to join a comprehensive course like **Java Programming for Complete Beginners - Java 16** by Ranga Karnam of In28Minutes on Udemy. This 30-hour course is great to learn both

Java and OOP fundamentals from scratch.

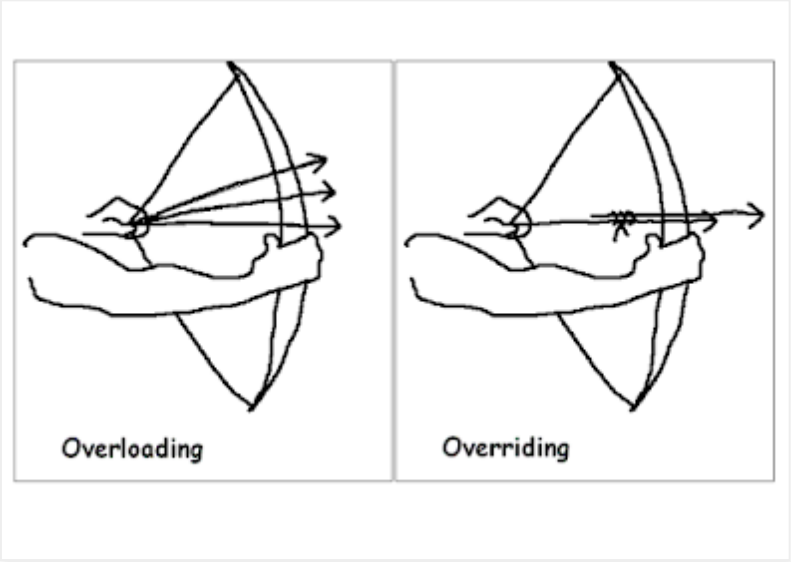
## Polymorphism and Overriding?

Overriding is a form of polymorphism that is used in Java to dynamically bind the method from the subclass in response to a method call from a subclass object referenced by superclass type. Method overriding is bonded using **dynamic binding in Java**.

Suppose you have two methods `size()` in both base class and derived class and base class variable is pointing to an object which happens to be subclass object at runtime then method from subclass will be called, I mean, the overridden method will be called.

This allows programming **for interface than implementation**, a popular OOP design principle because Polymorphism guarantees to invoke the correct method based upon the object. Method overriding is key for many flexible design patterns in Java, like Strategy and State Design Patterns in Java.

If you want to learn more about design patterns in Java then I highly recommend you to join this **Design Patterns in Java** course by Dmitri Nestruck on Udemy. It's a great course to learn the modern implementation of classic Java patterns, I highly recommend every Java programmer to join this course to learn patterns for writing better code.




## Polymorphism and Overloading

Method overloading is another form of Polymorphism though some people argue against that. In the case of overloading, you also got multiple methods with the same name but different method signatures but a call to correct method is resolved at compile time using static binding in Java.

Overloading is a compile-time activity as oppose to Overriding which is runtime activity. Because of this reason overloading is faster than method overriding in Java. Though beware of an overloaded method that creates conflict e.g. methods with only one parameter like `int` and long etc. See **What are method overloading in Java** for example and complete details.

An Example of Polymorphism in Java

## An example of Polymorphism in Java


Let's see a short example of Polymorphism in Java. In this example, the `Pet` variable behaves polymorphically because it can be either `Cat` or `Dog`. this is also an example of method overriding because the `makeSound()` method is overridden in subclass Dog and Cat.

```
import java.util.ArrayList;
import java.util.List;

abstract class Pet{
    public abstract void makeSound();
}

class Cat extends Pet{

    @Override
    public void makeSound() {
        System.out.println("Meow");
    }
```

```
}

class Dog extends Pet{

    @Override
    public void makeSound() {
        System.out.println("Woof");
    }

}
```

Let's test How the Polymorphism concept works in Java:

```
/**
 *
 * Java program to demonstrate What is Polymorphism
 * @author Javin Paul
 */
public class PolymorphismDemo{

    public static void main(String args[]) {
        //Now Pet will show How Polymorphism work in Java
        List<Pet> pets = new ArrayList<Pet>();
        pets.add(new Cat());
        pets.add(new Dog());

        //pet variable which is type of Pet behave different based
        //upon whether pet is Cat or Dog
        for(Pet pet : pets){
            pet.makeSound();
        }

    }
}
```

```
Output:
Meow
Woof
```

In Summary,