Java Method Overloading

In Java, two or more methods can have same name if they differ in parameters (different number of parameters, different types of parameters, or both). These methods are called overloaded methods and this feature is called method overloading. For example:

```
void func() { ... }
void func(int a) { ... }
float func(double a) { ... }
float func(int a, float b) { ... }
```

Here, the func() method is overloaded. These methods have the same name but accept different arguments.
Notice that, the return type of these methods is not the same. Overloaded methods may or may not have different return types, but they must differ in parameters they accept.

Why method overloading?

Suppose, you have to perform the addition of given numbers but there can be any number of arguments (let's say either 2 or 3 arguments for simplicity).

In order to accomplish the task, you can create two methods sum2num(int, int) and sum3num(int, int, int) for two and three parameters respectively. However, other programmers, as well as you in the future may get confused as the behavior of both methods are the same but they differ by name.

The better way to accomplish this task is by overloading methods. And, depending upon the argument passed, one of the overloaded methods is called. This helps to increase the readability of the program.

How to perform method overloading in Java?

If you observe the following example, Here we have created a class named Sample and this class has two methods with same name (add) and return type, the only difference is the parameters they accept (one method accepts two integer variables and other accepts three integer variables).

When you invoke the add() method based on the parameters you pass respective method body gets executed.

Example

```
public class Sample{
  public static void add(int a, int b){
    System.out.println(a+b);
  }
```

```
  public static void add(int a, int b, int c){
    System.out.println(a+b+c);
  }
  public static void main(String args[]){
    Sample obj = new Sample();
    obj.add(20, 40);
    obj.add(40, 50, 60);
  }
}
```

Output
60
150

Method overriding in java with example

Declaring a method in **sub class** which is already present in **parent class** is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class. In this case the method in parent class is called overridden method and the method in child class is called overriding method. In this guide, we will see what is method overriding in Java and why we use it.

Method Overriding Example

Lets take a simple example to understand this. We have two classes: A child class Boy and a parent class Human. The `Boy` class extends `Human` class. Both the classes have a common method `void eat()`. Boy class is giving its own implementation to the `eat()` method or in other words it is overriding the `eat()` method.

The purpose of Method Overriding is clear here. Child class wants to give its own implementation so that when it calls this method, it prints Boy is eating instead of Human is eating.

```
class Human{
  //Overridden method
  public void eat()
  {
    System.out.println("Human is eating");
  }
}
class Boy extends Human{
  //Overriding method
  public void eat(){
    System.out.println("Boy is eating");
  }
  public static void main( String args[]) {
    Boy obj = new Boy();
    //This will call the child class version of eat()
```

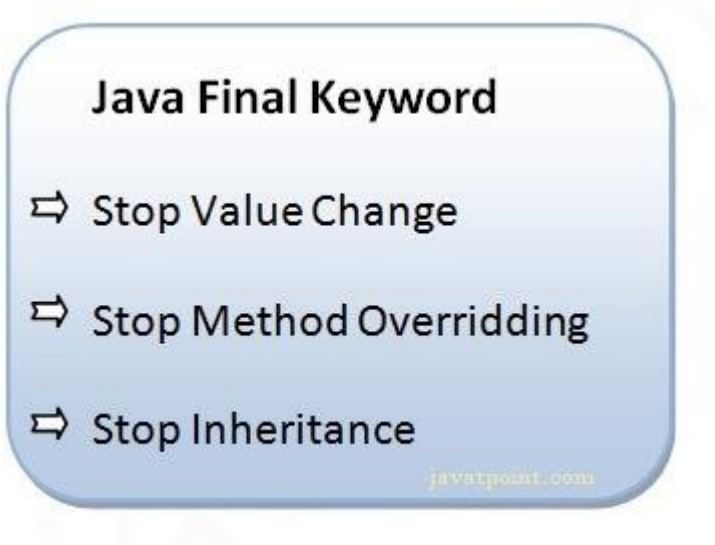
```
    obj.eat();
  }
}
```
Output:Boy is eating

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.



1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
1. class Bike9{
2.  final int speedlimit=90;//final variable
3.  void run(){
4.   speedlimit=400;
5.  }
6.  public static void main(String args[]){
7.  Bike9 obj=new  Bike9();
8.  obj.run();
9.  }
10. }//end of class
```

Output:Compile Time Error

---

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
1. class Bike{
2.   final void run(){System.out.println("running");}
3. }
4.
5. class Honda extends Bike{
```

```
6.    void run(){System.out.println("running safely with 100kmph");}
7.
8.    public static void main(String args[]){
9.    Honda honda= new Honda();
10.   honda.run();
11.   }
12. }
```

Output:Compile Time Error

---

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
1.  final class Bike{}
2.
3.  class Honda1 extends Bike{
4.    void run(){System.out.println("running safely with 100kmph");}
5.
6.    public static void main(String args[]){
7.    Honda1 honda= new Honda1();
8.    honda.run();
9.    }
10. }
```

Output:Compile Time Error

Difference between method overloading and method overriding in java

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.