Overloading in Java

Difficulty Level : Easy Last Updated : 17 Feb, 2021

Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both. Overloading is related to compile-time (or static) polymorphism.

```
// Java program to demonstrate working of method
// overloading in Java.

public class Sum {

    // Overloaded sum(). This sum takes two int parameters
    public int sum(int x, int y)
    {
        return (x + y);
    }

    // Overloaded sum(). This sum takes three int parameters
    public int sum(int x, int y, int z)
    {
        return (x + y + z);
    }

    // Overloaded sum(). This sum takes two double parameters
    public double sum(double x, double y)
    {
        return (x + y);
    }

    // Driver code
    public static void main(String args[])
    {
        Sum s = new Sum();
        System.out.println(s.sum(10, 20));
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }
}
```

Output :

```
30
60
31.0
```

Question Arises:

Q. What if the exact prototype does not match with arguments.

Ans.

Priority wise, compiler take these steps:

1. Type Conversion but to higher type(in terms of range) in same family.
2. Type conversion to next higher family(suppose if there is no long data type available for an int data type, then it will search for the float data type).

Let’s take an example to clear the concept:-

```
class Demo {
    public void show(int x)
    {
        System.out.println("In int" + x);
    }
    public void show(String s)
    {
        System.out.println("In String" + s);
    }
    public void show(byte b)
    {
        System.out.println("In byte" + b);
    }
}
class UseDemo {
    public static void main(String[] args)
    {
        byte a = 25;
        Demo obj = new Demo();
        obj.show(a); // it will go to
        // byte argument
        obj.show("hello"); // String
        obj.show(250); // Int
        obj.show('A'); // Since char is
        // not available, so the datatype
        // higher than char in terms of
        // range is int.
        obj.show("A"); // String
        obj.show(7.5); // since float datatype
// is not available and so it's higher
// datatype, so at this step their
// will be an error.
```

```
}
}
```

What is the advantage?

We don't have to create and remember different names for functions doing the same thing. For example, in our code, if overloading was not supported by Java, we would have to create method names like sum1, sum2, … or sum2Int, sum3Int, … etc.

Can we overload methods on return type?

We **cannot** overload by return type. This behavior is same in C++. Refer this for details

```
public class Main {
    public int foo() { return 10; }

    // compiler error: foo() is already defined
    public char foo() { return 'a'; }

    public static void main(String args[])
    {
    }
}
```

However, Overloading methods on return type are possible in cases where the data type of the function being called is explicitly specified. Look at the examples below :

```
// Java program to demonstrate the working of method
// overloading in static methods
public class Main {

    public static int foo(int a) { return 10; }
    public static char foo(int a, int b) { return 'a'; }

    public static void main(String args[])
    {
        System.out.println(foo(1));
        System.out.println(foo(1, 2));
    }
}
```

Output:

```
10
a
```

```
// Java program to demonstrate working of method
// overloading in  methods
class A {
    public int foo(int a) { return 10; }

    public char foo(int a, int b) { return 'a'; }
}

public class Main {

    public static void main(String args[])
    {
        A a = new A();
        System.out.println(a.foo(1));
        System.out.println(a.foo(1, 2));
    }
}
```

Output:

```
10
a
```

Can we overload static methods?

The answer is '**Yes**'. We can have two ore more static methods with same name, but differences in input parameters. For example, consider the following Java program. Refer this for details.

Can we overload methods that differ only by static keyword?

We **cannot** overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same). See following Java program for example.
Refer this for details.

Can we overload main() in Java?

Like other static methods, we **can** overload main() in Java. Refer overloading main() in Java for more details.

```
// A Java program with overloaded main()
import java.io.*;

public class Test {

    // Normal main()
    public static void main(String[] args)
    {
        System.out.println("Hi Geek (from main)");
        Test.main("Geek");
    }

    // Overloaded main methods
    public static void main(String arg1)
    {
        System.out.println("Hi, " + arg1);
        Test.main("Dear Geek", "My Geek");
    }
    public static void main(String arg1, String arg2)
    {
        System.out.println("Hi, " + arg1 + ", " + arg2);
    }
}
```

Output :

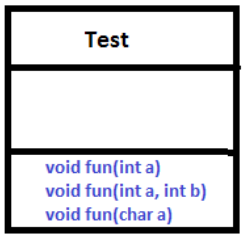
```
Hi Geek (from main)
Hi, Geek
Hi, Dear Geek, My Geek
```

Does Java support Operator Overloading?

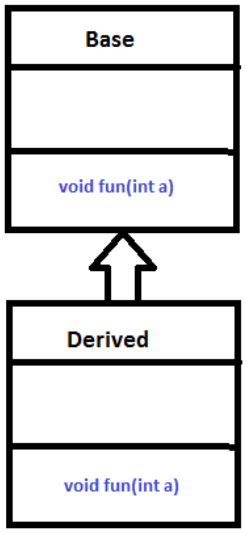
Unlike C++, Java doesn't allow user-defined overloaded operators. Internally Java overloads operators, for example, + is overloaded for concatenation.

What is the difference between Overloading and Overriding?

- Overloading is about same function have different signatures. Overriding is about same function, same signature but different classes connected through inheritance.



Overloading



Overriding

- Overloading is an example of compiler time polymorphism and overriding is an example of run time polymorphism.

Related Articles:

- Different ways of Method Overloading in Java
- Method Overloading and Null error in Java
- Can we Overload or Override static methods in java ?

This article is contributed by **Shubham Agrawal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.