

Method Overloading in Java

If a [class](#) has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the [program](#).

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

In java, Method Overloading is not possible by changing the return type of the method only.

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating [static methods](#) so that we don't need to create instance for calling methods.

1. **class** Adder{

```
2. static int add(int a,int b){return a+b;}
3. static int add(int a,int b,int c){return a+b+c;}
4. }
5. class TestOverloading1{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(11,11,11));
9. }}
```

Output:

```
22
33
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in [data type](#). The first add method receives two integer arguments and second add method receives two double arguments.

```
1. class Adder{
2. static int add(int a, int b){return a+b;}
3. static double add(double a, double b){return a+b;}
4. }
5. class TestOverloading2{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(12.3,12.6));
9. }}
```

Output:

```
22
24.9
```

Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
1. class Adder{
2. static int add(int a,int b){return a+b;}
3. static double add(int a,int b){return a+b;}
4. }
5. class TestOverloading3{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));//ambiguity
8. }}
```

Output:

```
Compile Time Error: method add(int,int) is already defined in class
Adder
```

System.out.println(Adder.add(11,11)); //Here, how can java determine which sum() method should be called?

Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters.

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But [JVM](#) calls main() method which receives string array as arguments only. Let's see the simple example:

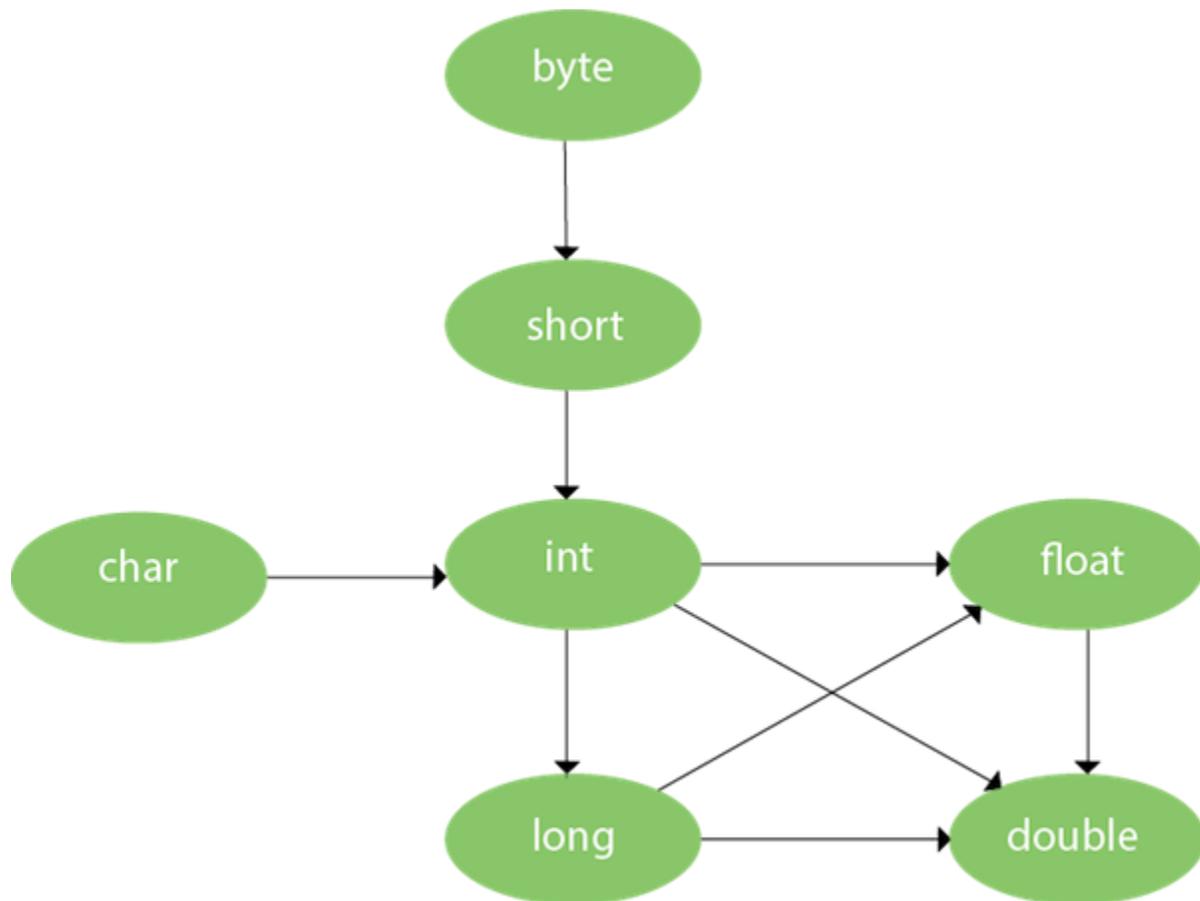
```
1. class TestOverloading4{
2. public static void main(String[] args){System.out.println("main with String[]");}
3. public static void main(String args){System.out.println("main with String");}
4. public static void main(){System.out.println("main without args");}
5. }
```

Output:

```
main with String[]
```

Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on.

Example of Method Overloading with TypePromotion

```
1. class OverloadingCalculation1{
2. void sum(int a,long b){System.out.println(a+b);}
3. void sum(int a,int b,int c){System.out.println(a+b+c);}
4.
5. public static void main(String args[]){
6. OverloadingCalculation1 obj=new OverloadingCalculation1();
7. obj.sum(20,20);//now second int literal will be promoted to long
8. obj.sum(20,20,20);
9.
10. }
11. }
```

```
Output:40
        60
```

Example of Method Overloading with Type Promotion if matching found

If there are matching type arguments in the method, type promotion is not performed.

1. **class** OverloadingCalculation2{
2. **void** sum(**int** a,**int** b){System.out.println("int arg method invoked");}
3. **void** sum(**long** a,**long** b){System.out.println("long arg method invoked");}
- 4.
5. **public static void** main(String args[]){
6. OverloadingCalculation2 obj=**new** OverloadingCalculation2();
7. obj.sum(20,20);*//now int arg sum() method gets invoked*
8. }
9. }

```
Output:int arg method invoked
```

Example of Method Overloading with Type Promotion in case of ambiguity

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

1. **class** OverloadingCalculation3{
2. **void** sum(**int** a,**long** b){System.out.println("a method invoked");}
3. **void** sum(**long** a,**int** b){System.out.println("b method invoked");}
- 4.
5. **public static void** main(String args[]){
6. OverloadingCalculation3 obj=**new** OverloadingCalculation3();
7. obj.sum(20,20);*//now ambiguity*
8. }
9. }

```
Output:Compile Time Error
```