# Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

**class** Operation{

**int** data=50;

**void** change(**int** data){

 data=data+100;//changes will be in the local variable only

 }

**public** **static** **void** main(String args[]){

   Operation op=**new** Operation();

   System.out.println("before change "+op.data);

   op.change(500);

   System.out.println("after change "+op.data);

 }

}

output:before change 50

after change 50

### Another Example of call by value in java

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Let's take a simple example:

**class** Operation2{

**int** data=50;

**void** change(Operation2 op){

 op.data=op.data+100;//changes will be in the instance variable

 }

**public** **static** **void** main(String args[]){

   Operation2 op=**new** Operation2();

   System.out.println("before change "+op.data);

   op.change(op);//passing object

   System.out.println("after change "+op.data);

 }

}

Output:before change 50

after change 150

# 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.

So, we perform method overloading to figure out the program quickly.

## 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

### 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.

**class** Adder{

**static** **int** add(**int** a,**int** b){**return** a+b;}

**static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;}

}

**class** TestOverloading1{

**public** **static** **void** main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

### 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.

**class** Adder{

**static** **int** add(**int** a, **int** b){**return** a+b;}

**static** **double** add(**double** a, **double** b){**return** a+b;}

}

**class** TestOverloading2{

**public** **static** **void** main(String[] args){

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}}

### 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.

**class** TestOverloading4{

**public** **static** **void** main(String[] args){System.out.println("main with String[]");}

**public** **static** **void** main(String args){System.out.println("main with String");}

**public** **static** **void** main(){System.out.println("main without args");}

Recursion in Java

Recursion in java is a process in which a method/function calls itself continuously. A method in java that calls itself is called recursive method/recursive function.

## Java Recursion Example 1: Infinite times

**public** **class** RecursionExample1

 {

**static** **void** p()

{

System.out.println("hello");

p();

}

**public** **static** **void** main(String[] args) {

p();

}

}

Output:

hello

hello

...

java.lang.StackOverflowError

## Java Recursion Example 2: Finite times

**public** **class** RecursionExample2

{

**static** **int** count=0;

**static** **void** p()

{

count++;

**if**(count<=5)

{

System.out.println("hello "+count);

p();

}

}

**public** **static** **void** main(String[] args) {

p();

}

}

## Java Recursion Example 3: Factorial Number

**public** **class** RecursionExample3 {

**static** **int** factorial(**int** n)

{

**if** (n == 1)

**return** 1;

**else**

**return**(n \* factorial(n-1));

    }

**public** **static** **void** main(String[] args) {

System.out.println("Factorial of 5 is: "+factorial(5));

}

}

## Java Recursion Example 4: Fibonacci Series

**public** **class** RecursionExample4 {

**static** **int** n1=0,n2=1,n3=0;

**static** **void** printFibo(**int** count){

**if**(count>0){

             n3 = n1 + n2;

             n1 = n2;

             n2 = n3;

             System.out.print(" "+n3);

             printFibo(count-1);

         }

     }

**public** **static** **void** main(String[] args) {

**int** count=15;

      System.out.print(n1+" "+n2);//printing 0 and 1

      printFibo(count-2);//n-2 because 2 numbers are already printed

}

}