## **Java Methods**

A method is a block of code that performs a specific task.

Suppose you need to create a program to create a circle and color it. You can create two methods to solve this problem:

* a method to draw the circle
* a method to color the circle

Dividing a complex problem into smaller chunks makes your program easy to understand and reusable.

In Java, there are two types of methods:

* User-defined Methods: We can create our own method based on our requirements.
* Standard Library Methods: These are built-in methods in Java that are available to use.

Let's first learn about user-defined methods.

## **Declaring a Java Method**

The syntax to declare a method is:

returnType methodName() {

// method body

}

Here,

* Access/Non Access Modifier(Public or Private)
* returnType - It specifies what type of value a method returns For example if a method has an int return type then it returns an integer value.  
    
  If the method does not return a value, its return type is void.
* methodName - It is an [identifier](https://www.programiz.com/java-programming/keywords-identifiers#identifiers) that is used to refer to the particular method in a program.
* method body - It includes the programming statements that are used to perform some tasks. The method body is enclosed inside the curly braces { }.

For example,

int addNumbers() {

// code

}

In the above example, the name of the method is addNumbers(). And, the return type is int.

This is the simple syntax of declaring a method. However, the complete syntax of declaring a method is

modifier static returnType nameOfMethod (parameter1, parameter2, ...) {

// method body

}

Here,

* modifier - It defines access types whether the method is public, private, and so on.
* static - If we use the static keyword, it can be accessed without creating objects.  
    
  For example, the sqrt() method of standard [Math class](https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html) is static. Hence, we can directly call Math.sqrt() without creating an instance of Math class.
* Parameter1/parameter2 - These are values passed to a method. We can pass any number of arguments to a method.



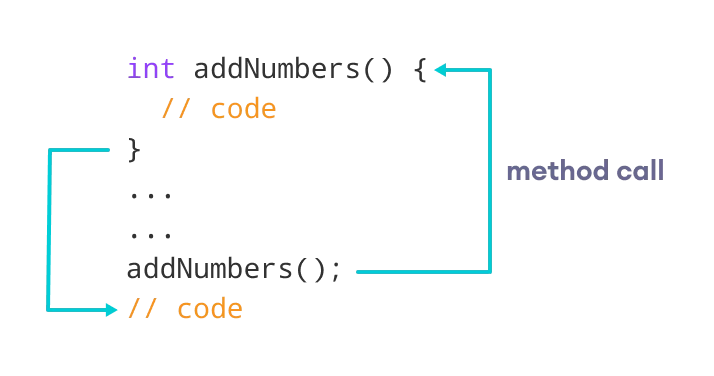
**Calling a Method in Java**

In the above example, we have declared a method named addNumbers(). Now, to use the method, we need to call it.

Here's is how we can call the addNumbers() method.

// calls the method

addNumbers();



## **Java Methods**

// create a method

**public** **int** addNumbers(**int** a, **int** b) {

**int** sum = a + b;

// return value

**return** sum;

}

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

**int** num1 = 25;

**int** num2 = 15;

// create an object of Main class

addnumber obj = **new** addnumber();

// calling method

**int** result = obj.addNumbers(num1, num2);

System.***out***.println("Sum is: " + result);

}

}

In the above example, we have created a method named addNumbers(). The method takes two parameters a and b. Notice the line,

int result = obj.addNumbers(num1, num2);

Here, we have called the method by passing two arguments num1 and num2. Since the method is returning some value, we have stored the value in the result variable.

Note: The method is not static. Hence, we are calling the method using the object of the class.

## **Java Method Return Type**

A Java method may or may not return a value to the function call. We use the return statement to return any value. For example,

int addNumbers() {

...

return sum;

}

Here, we are returning the variable sum. Since the return type of the function is int. The sum variable should be of int type. Otherwise, it will generate an error.

### **Method Return Type**

// create a method

**public** **static** **int** square(**int** num) {

// return statement

**return** num \* num;

}

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

**int** result;

// call the method

// store returned value to result

result = *square*(10);

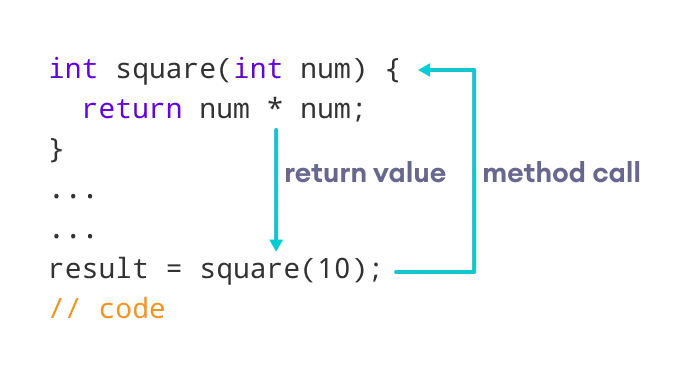
System.***out***.println("Squared value of 10 is: " + result);

}

}

In the above program, we have created a method named square(). The method takes a number as its parameter and returns the square of the number.

Here, we have mentioned the return type of the method as int. Hence, the method should always return an integer value.



Note: If the method does not return any value, we use the void keyword as the return type of the method. For example,

public void square(int a) {

int square = a \* a;

System.out.println("Square is: " + a);

}

## **Method Parameters in Java**

A method parameter is a value accepted by the method. As mentioned earlier, a method can also have any number of parameters. For example,

// method with two parameters

int addNumbers(int a, int b) {

// code

}

// method with no parameter

int addNumbers(){

// code

}

If a method is created with parameters, we need to pass the corresponding values while calling the method. For example,

// calling the method with two parameters

addNumbers(25, 15);

// calling the method with no parameters

addNumbers()

### **Method Parameters**

Here, the parameter of the method is int. Hence, if we pass any other data type instead of int, the compiler will throw an error. It is because Java is a strongly typed language.

Note: The argument 24 passed to the display2() method during the method call is called the actual argument.

The parameter num accepted by the method definition is known as a formal argument. We need to specify the type of formal arguments. And, the type of actual arguments and formal arguments should always match.

## **Standard Library Methods**

The standard library methods are built-in methods in Java that are readily available for use. These standard libraries come along with the Java Class Library (JCL) in a Java archive (\*.jar) file with JVM and JRE.

For example,

* print() is a method of java.io.PrintSteam. The print("...") method prints the string inside quotation marks.
* sqrt() is a method of Math class. It returns the square root of a number.

Here's a working example:

### **Java Standard Library Method**

public class Main {

public static void main(String[] args) {

// using the sqrt() method

System.out.print("Square root of 4 is: " + Math.sqrt(4));

}

}

## **What are the advantages of using methods?**

1. The main advantage is code reusability. We can write a method once, and use it multiple times. We do not have to rewrite the entire code each time. Think of it as, "write once, reuse multiple times".

### **Java Method for Code Reusability**

public class Main {

// method defined

private static int getSquare(int x){

return x \* x;

}

public static void main(String[] args) {

for (int i = 1; i <= 5; i++) {

// method call

int result = getSquare(i);

System.out.println("Square of " + i + " is: " + result);

}

}

}

In the above program, we have created the method named getSquare() to calculate the square of a number. Here, the method is used to calculate the square of numbers less than 6.

Hence, the same method is used again and again.

2. Methods make code more readable and easier to debug. Here, the getSquare() method keeps the code to compute the square in a block. Hence, makes it more readable.

# **Pass Arrays to Methods**

Arrays can be passed to other methods just like how you pass primitive data type’s arguments. To pass an array as an argument to a method, you just have to pass the name of the array without square brackets. The method prototype should match to accept the argument of the array type.

Given below is the method prototype:

void method\_name (int [] array);

This means method\_name will accept an array parameter of type int. So if you have an int array named myarray, then you can call the above method as follows:

method\_name (myarray);

The above call passes the reference to the array myarray to the method ‘method\_name’. Thus, the changes made to myarray inside the method will reflect in the calling method as well.

The following Java program demonstrates the passing of an array as a parameter to the function.

|  |
| --- |
| public class Main  {      //method to print an array, taking array as an argument      private static void printArray(Integer[] intArray){          System.out.println("Array contents printed through method:");          //print individual elements of array using enhanced for loop         for(Integer val: intArray)            System.out.print(val + " ");      }      public static void main(String[] args) {          //integer array              Integer[] intArray = {10,20,30,40,50,60,70,80};             //call printArray method by passing intArray as an argument           printArray(intArray);      }  } |

In the above program, an array is initialized in the main function. Then the method printArray is called to which this array is passed as an argument. In the printArray method, the array is traversed and each element is printed using the enhanced for loop.

### **Return An Array In Java**

Apart from all the primitive types that you can return from Java programs, you can also return references to arrays.

While returning a reference to an array from a method, you should keep in mind that:

* The data type that returns value should be specified as the array of the appropriate data type.
* The returned value from a method is the reference to the array.

The array is returned from a method in the cases where you need to return multiple values of the same type from a method. This approach becomes useful as Java doesn’t allow returning multiple values.

**The following program returns a string array from a method.**

|  |
| --- |
| import java.util.\*;  public class Main  {  public static String[] return\_Array() {         //define string array         String[] ret\_Array = {"Java", "C++", "Python", "Ruby", "C"};        //return string array        return ret\_Array;     }    public static void main(String args[]) {        //call method return\_array that returns array       String[] str\_Array = return\_Array();       System.out.println("Array returned from method:" + Arrays.toString(str\_Array));        }  } |