Java Functions (Methods) — Detailed Guide

In Java, what many languages call a **function** is usually called a **method**. A method is a block of code designed to perform a particular task. Once you create it, you can call it whenever you need it.

For example, instead of writing the addition code multiple times, you can create an `add()` method and reuse it.

---

1. Basic Structure of a Java Method

Syntax

```
accessModifier static returnType methodName(parameters) {
   // method body
   return value;
}
```

For example:

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

Let's understand every part:

Part	Meaning
static	Allows us to call the method from `main()` without creating an object
int	Return type
add	Method name
int a, int b	Parameters
{ }	Method body
return	Sends a value back

---

2. Your First Function

```
public class Main {

   static void hello() {
      System.out.println("Hello Java");
   }

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

Output

```
Hello Java
```

What happened?

First, we created:

```
static void hello()
```

Then we called it inside `main()`:

```
hello();
```

The program executes the code inside the `hello()` method.

---

3. Function With Parameters

A **parameter** allows us to send information to a method.

Example

```
public class Main {

   static void greet(String name) {
      System.out.println("Hello " + name);
   }

   public static void main(String[] args) {
      greet("Rahul");
      greet("Amit");
   }
}
```

Output

```
Hello Rahul
Hello Amit
```

Here:

`String name`

is the parameter.

When we call:

`greet("Rahul");`

`"Rahul"` is passed into `name`.

You can think of it like:

```
greet("Rahul")
     ↓
  name = "Rahul"
```

---

4. Function That Returns a Value

Sometimes you don't want a method to simply print something. You want it to **calculate something and give you the result**.

Example:

```
public class Main {

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

   public static void main(String[] args) {

      int result = add(10, 20);

      System.out.println("Result = " + result);
   }
}
```

Output

```
Result = 30
```

The flow is:

```
add(10, 20)
   ↓
10 + 20
   ↓
30
```

```
  ↓
result
```

---

5. void vs int

This is very important.

```
void
```

`void` means the method **does not return a value**.

```
static void hello() {
   System.out.println("Hello");
}
```

```
int
```

`int` means the method returns an integer.

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

For example:

```
int x = add(5, 10);
```

Now `x` contains:

```
15
```

---

6. Four Common Types of Methods

You should learn these four patterns.

Type 1: No parameter + no return

```
static void display() {
   System.out.println("Hello");
}
```

Call:

```
display();
```

Type 2: Parameter + no return

```
static void displayName(String name) {
   System.out.println(name);
}
```

Call:

```
displayName("Rahul");
```

Type 3: No parameter + return

```
static int getNumber() {
   return 100;
}
```

Call:

```
int x = getNumber();
```

Type 4: Parameter + return

```
static int multiply(int a, int b) {
   return a * b;
}
```

Call:

```
int result = multiply(5, 4);
```

Result:

```
20
```

Type 4 is especially important because you'll use it frequently in real programs.

---

7. Example: Even or Odd

Let's make a useful method.

```
public class Main {

   static void checkEvenOdd(int number) {

      if (number % 2 == 0) {
         System.out.println("Even");
      } else {
         System.out.println("Odd");
      }
   }

   public static void main(String[] args) {

      checkEvenOdd(10);
      checkEvenOdd(7);
   }
}
```

Output

```
Even
Odd
```

The % operator gives the remainder.

```
10 % 2 = 0 → Even

7 % 2 = 1 → Odd
```

---

8. Example: Find the Largest Number

```
public class Main {

   static int maximum(int a, int b) {

      if (a > b) {
         return a;
      } else {
         return b;
      }
   }

   public static void main(String[] args) {

      int answer = maximum(25, 40);

      System.out.println("Maximum = " + answer);
   }
}
```

Output

```
Maximum = 40
```

Notice that the method doesn't print the answer.

It **returns** the answer:

```
return b;
```

Then `main()` prints it:

```
System.out.println("Maximum = " + answer);
```

9. Method With Multiple Parameters

You can have multiple parameters.

```
static int calculateSum(int a, int b, int c) {
   return a + b + c;
}
```

Call:

```
int result = calculateSum(10, 20, 30);
```

Output:

```
60
```

10. Method With double

Methods aren't limited to integers.

```
public class Main {

   static double calculateArea(double length, double width) {
      return length * width;
   }

   public static void main(String[] args) {

      double area = calculateArea(10.5, 5.0);

      System.out.println("Area = " + area);
   }
}
```

Output

```
Area = 52.5
```

11. Method Calling Another Method

A method can call another method.

```
public class Main {

   static int square(int n) {
      return n * n;
   }

   static int cube(int n) {
      return square(n) * n;
   }

   public static void main(String[] args) {

      System.out.println(square(5));
      System.out.println(cube(5));
   }
}
```

Output

```
25
125
```

Here:

```
cube(5)
```

calls:
square(5)

---

12. Why Do We Use Functions?

Without methods, you might write:

```
int a = 10;
int b = 20;
System.out.println(a + b);

int x = 30;
int y = 40;
System.out.println(x + y);
```

With a method:

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

You can simply do:

```
System.out.println(add(10, 20));
System.out.println(add(30, 40));
```

Advantages

- **Reusability** — write code once and use it many times.
- **Readability** — programs become easier to understand.
- **Organization** — divide a large program into smaller pieces.
- **Debugging** — easier to find and fix problems.
- **Maintenance** — easier to modify code later.

---

13. `static` — What Does It Mean?

You'll often see:

```
static int add(int a, int b)
```

when learning Java.

`static` allows you to call the method directly from `main()`:

```
add(10, 20);
```

Without `static`, you generally need an object:

```
Main obj = new Main();

obj.add(10, 20);
```

You'll learn more about this when studying **classes and objects**.

---

14. `return` Statement

The `return` statement sends a value back to the place where the method was called.

Example:

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

When you write:

```
int result = add(5, 10);
```

Java effectively gets:

```
add(5, 10)
  ↓
5 + 10
  ↓
15
  ↓
result = 15
```

A method with a non-`void` return type must return a compatible value.

For example:

```
static int getAge() {
   return 20;
}
```

Correct.

But:

```
static int getAge() {
   return "20";  // Error
}
```

is incorrect because `"20"` is a `String`, not an `int`.

---

15. Complete Beginner Example

Here's a small program using several methods:

```
public class Main {

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

   static int subtract(int a, int b) {
      return a - b;
   }

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

   static void showResult(String operation, int result) {
      System.out.println(operation + " = " + result);
   }

   public static void main(String[] args) {

      int sum = add(10, 5);
      int difference = subtract(10, 5);
      int product = multiply(10, 5);

      showResult("Addition", sum);
      showResult("Subtraction", difference);
      showResult("Multiplication", product);
   }
}
```

Output

```
Addition = 15
Subtraction = 5
Multiplication = 50
```