Methods in Java

A method is a collection of statements that perform specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping the code. In Java, every method must be part of some class that is different from languages like C, C++, and Python.

Method Declaration

In general, method declarations have six components :

- Modifier-: Defines access type of the method i.e. from where it can be accessed in your application. In Java, there 4 types of access specifiers.
    - public: accessible in all classes in your application.
    - protected: accessible within the class in which it is defined and in its subclass(es)
    - private: accessible only within the class in which it is defined.
    - default (declared/defined without using any modifier): accessible within the same class and package within which its class is defined.
- The return type: The data type of the value returned by the method or void if does not return a value.
- Method Name: The rules for field names apply to method names as well, but the convention is a little different.
- Parameter list: Comma-separated list of the input parameters is defined, preceded by their data type, within

the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().

- Exception list: The exceptions you expect by the method can throw, you can specify these exception(s).
- Method body: It is enclosed between braces. The code you need to be executed to perform your intended operations.
- Method signature: It consists of method name and parameter list (number of parameters, type of the parameters and order of the parameters). Return type and exceptions are not considered as part of it.

Following examples demonstrates different types and usages of methods.

1. Basic Method Example

A simple method is one that performs a specific task when called. Below is an example where a method fun() is invoked from the main() method.

```
// Java Program to show

// Basic method calling

import java.io.*;

import java.math.*;

import java.util.*;
```

```
class GfG {

    public static void main(String[] args) {

        // Before calling the fun method

        System.out.println("Before Call");



        // Calling the fun method

        fun();



        // After calling the fun method

        System.out.println("After Call");

    }



    // fun method to print a simple message

    public static void fun() {

        // Printing message inside fun method

        System.out.println("Inside Fun");

    }

}
```

```
Output

Before Call

Inside Fun

After Call
```

2. Calling Multiple Methods

A method can call other methods, enabling nested functionality.

```
// Java Program

// Calling Multiple Methods

import java.io.*;

import java.math.*;

import java.util.*;



class GfG {

    public static void main(String[] args) {

        // Main method starts

        System.out.println("Main Begins");



        // Calling fun1 method

        fun1();
```

```java
        // Main method ends
        System.out.println("Main Ends");
    }

    // fun1 method that calls fun2
    public static void fun1() {
        // Printing message when fun1 starts
        System.out.println("fun1 Begins");

        // Calling fun2 method
        fun2();

        // Printing message when fun1 ends
        System.out.println("fun1 Ends");
    }

    // fun2 method that prints a message
    public static void fun2() {
        // Printing message inside fun2
        System.out.println("Inside fun2");

    }

}

```

```
Output

Main Begins

fun1 Begins

Inside fun2

fun1 Ends

Main Ends
```

3. Repeated Method Calls

The same method can be called multiple times to perform repetitive tasks.

```
// Java Program to show

// Repeated Method Calls

import java.io.*;

import java.math.*;

import java.util.*;
```

```java
public class GfG {

    public static void main(String[] args) {

        // Calling fun method twice

        fun();

        fun();

    }



    // fun method that adds two integers and prints the

    // result

    public static void fun() {

        int x = 5, y = 10;



        // Printing the sum of x and y

        System.out.println(x + y);

    }

}
```

Output

```
15

15
```

4. Methods with Object References

When objects are passed as arguments, changes made inside the method affect the original object.

```
// Java Program to show

// Methods with Object References

import java.math.*;

import java.io.*;

import java.util.*;



class Point {

    // Declaring two integer fields for x and y coordinates

    int x;

    int y;

}



public class Gfg {

    public static void main(String[] args) {

        // Creating a Point object

        Point p = new Point();
        // Assigning values to the coordinates

        p.x = 5;

        p.y = 10;



        // Calling the fun method and passing the Point object

        fun(p);



        // Printing the updated coordinates of p

        System.out.println(p.x + " " + p.y);

    }



    // fun method that modifies the fields of the passed Point object

    public static void fun(Point p) {

        // Modifying the fields of Point object p

        p.x = 10;

        p.y = 10;

    }

}
```

Output

```
10 10
```

This example shows how methods can modify the state of objects passed as arguments.

5. Passing Objects by Reference vs. Reassigning Objects

If an object reference is reassigned inside a method, it does not affect the original object.

```
// Java Program to show

// Passing Objects by Reference vs. Reassigning Objects

import java.io.*;

import java.math.*;

import java.util.*;



class Point {

    // Declaring two integer fields for x and y coordinates

    int x;

    int y;

}



public class Gfg {

    public static void main(String[] args) {
        // Creating a Point object

        Point p = new Point();



        // Assigning values to the coordinates

        p.x = 5;

        p.y = 10;



        // Calling the fun method and passing the Point

        // object

        fun(p);



        // Printing the unchanged coordinates of p

        System.out.println(p.x + " " + p.y);

    }



    // fun method that creates a new Point object and

    // modifies it

    public static void fun(Point p) {

        // Creating a new Point object

        p = new Point();
        // Modifying the new Point object
        p.x = 10;
        p.y = 10;
    }
}
```

Output

```
5 10
```

In this case, the original object remains unchanged because the reassignment inside fun() only affects the local reference.