1. Defining and Calling Methods

Defining a Method

A method is defined with the following components:

Return Type: The type of value the method will return (e.g., int, void for no return value).

Method Name: A unique name to identify the method.

Parameters: Optional inputs (arguments) the method may take.

Method Body: The code block that contains the logic of the method.

Syntax:

java

Copy

Edit

returnType methodName(parameter1, parameter2, ...) {

// Method body

// Code that performs some operation

}

Example:

java

Copy

Edit

// Method that adds two numbers

public int add(int a, int b) {

return a + b;

}

Calling a Method

To call a method, you simply use the method name, passing the required arguments.

Example:

java

Copy

Edit

public class Main {

public static void main(String[] args) {

Main obj = new Main(); // Create an object of the class

int sum = obj.add(10, 20); // Call the add method

System.out.println("The sum is: " + sum); // Output the result

}

public int add(int a, int b) {

return a + b;

}

}

In this example, the add() method is called with 10 and 20 as parameters, and the result is printed.

2. Method Parameters (Pass-by-Value)

In Java, method parameters are passed by value. This means the method receives a copy of the argument’s value, not the actual variable.

Primitive types (like int, float, boolean) are passed by value. Changes to the parameter inside the method do not affect the original argument.

Objects are also passed by value, but the value is a reference to the object. While the reference itself is passed by value, changes to the object’s attributes inside the method will affect the original object.

Example with Primitive Type:

java

Copy

Edit

public class Main {

public static void main(String[] args) {

int number = 10;

modifyNumber(number);

System.out.println("After modification: " + number); // Prints 10

}

public static void modifyNumber(int num) {

num = 20; // Change is local to the method

}

}

Here, num is a copy of number. Modifying num does not affect number in the main() method.

Example with Object:

java

Copy

Edit

public class Person {

String name;

public Person(String name) {

this.name = name;

}

}

public class Main {

public static void main(String[] args) {

Person p = new Person("John");

changeName(p);

System.out.println("Name after change: " + p.name); // Prints "Jane"

}

public static void changeName(Person person) {

person.name = "Jane"; // Modifies the original object

}

}

In this example, the object p is passed to the changeName() method. The method modifies the name attribute of the original p object.

3. Method Overloading

Method overloading in Java allows you to define multiple methods with the same name but different parameter lists (different number or type of parameters). The return type can be the same or different.

Java decides which method to call based on the method signature (the name and the number or type of parameters).

Syntax:

java

Copy

Edit

returnType methodName(parameter1) {

// Method body

}

returnType methodName(parameter1, parameter2) {

// Method body

}

Example:

java

Copy

Edit

public class Main {

public static void main(String[] args) {

System.out.println(add(10, 20)); // Calls the two-parameter method

System.out.println(add(5)); // Calls the one-parameter method

}

public static int add(int a, int b) {

return a + b;

}

public static int add(int a) {

return a + 10; // Adds 10 to the passed number

}

}

In this example, both add() methods have the same name but different parameters, so they are considered overloaded methods.

4. Return Keyword

The return keyword is used to exit a method and optionally return a value to the caller. If the method's return type is void, the return statement is optional.

For non-void methods: You must use return to return a value.

For void methods: You can use return to exit the method early, but you do not need to return anything.

Syntax:

java

Copy

Edit

return value; // For non-void methods

Example (Non-void method):

java

Copy

Edit

public class Main {

public static void main(String[] args) {

int result = multiply(5, 4);

System.out.println("Multiplication result: " + result);

}

public static int multiply(int a, int b) {

return a \* b; // Return the product of a and b

}

}

In this example, the multiply() method returns the product of a and b.

Example (Void method with early exit):

java

Copy

Edit

public class Main {

public static void main(String[] args) {

printMessage(5);

}

public static void printMessage(int count) {

if (count <= 0) {

return; // Exit if count is less than or equal to 0

}

System.out.println("Message printed " + count + " times.");

}

}

Here, the printMessage() method exits early if count is 0 or less.

5. Variable Arguments (Varargs)

Varargs allows you to pass a variable number of arguments to a method. Varargs is used when you don’t know how many arguments will be passed, and it allows you to pass any number of arguments (including none).

Syntax:

java

Copy

Edit

public static void methodName(dataType... varargs) {

// varargs is treated as an array inside the method

}

Example:

java

Copy

Edit

public class Main {

public static void main(String[] args) {

printNumbers(10, 20, 30); // Three arguments

printNumbers(5, 10); // Two arguments

printNumbers(); // No arguments

}

public static void printNumbers(int... numbers) {

for (int num : numbers) {

System.out.println(num); // Prints each number

}

}

}

Here, the printNumbers() method can accept any number of int arguments. The numbers parameter is treated as an array within the method.

Summary of Key Topics:

Defining and Calling Methods: Methods allow code reusability, where you define the logic and call the method with the required parameters.

Method Parameters (Pass-by-Value): Java passes parameters by value, meaning that primitive types are copied, and object references are copied (but the object itself can still be modified).

Method Overloading: You can have multiple methods with the same name but different parameter lists, allowing you to perform similar tasks with different inputs.

Return Keyword: The return keyword exits the method and optionally returns a value. For void methods, it's used to exit early if needed.

Variable Arguments (Varargs): Varargs allows passing a variable number of arguments to a method, which is treated as an array.

**Summary of Java Methods with Examples**

1. **Defining and Calling Methods**:  
   Methods consist of:
   * **Return Type**: Type of value returned (e.g., int, void).
   * **Method Name**: Unique identifier.
   * **Parameters**: Optional inputs.
   * **Body**: Logic inside the method.

**Example**:

java

CopyEdit

public int add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

Main obj = new Main();

int sum = obj.add(10, 20);

System.out.println("Sum: " + sum); // Output: Sum: 30

}

1. **Method Parameters (Pass-by-Value)**:
   * **Primitive types**: Passed as copies; changes don't affect the original.
   * **Objects**: References are passed, so changes to the object’s attributes affect the original.

**Primitive Example**:

java

CopyEdit

public static void modifyNumber(int num) {

num = 20; // Local change

}

public static void main(String[] args) {

int number = 10;

modifyNumber(number);

System.out.println(number); // Output: 10

}

**Object Example**:

java

CopyEdit

public static void changeName(Person person) {

person.name = "Jane";

}

public static void main(String[] args) {

Person p = new Person("John");

changeName(p);

System.out.println(p.name); // Output: Jane

}

1. **Method Overloading**:  
   Multiple methods with the same name but different parameter lists.  
   **Example**:

java

CopyEdit

public static int add(int a, int b) {

return a + b;

}

public static int add(int a) {

return a + 10;

}

public static void main(String[] args) {

System.out.println(add(10, 20)); // Output: 30

System.out.println(add(5)); // Output: 15

}

1. **Return Keyword**:  
   Used to exit a method and optionally return a value.

**Non-void Example**:

java

CopyEdit

public static int multiply(int a, int b) {

return a \* b;

}

public static void main(String[] args) {

System.out.println(multiply(5, 4)); // Output: 20

}

**Void Example**:

java

CopyEdit

public static void printMessage(int count) {

if (count <= 0) return; // Early exit

System.out.println("Count: " + count);

}

1. **Variable Arguments (Varargs)**:  
   Allows passing a variable number of arguments.

**Example**:

java

CopyEdit

public static void printNumbers(int... numbers) {

for (int num : numbers) {

System.out.println(num);

}

}

public static void main(String[] args) {

printNumbers(10, 20, 30); // Output: 10, 20, 30

printNumbers(5, 10); // Output: 5, 10

printNumbers(); // No output

}

**Key Points:**

* **Methods** promote reusability and flexibility.
* Java uses **pass-by-value** for primitives and object references.
* **Method overloading** enables multiple methods with the same name but varied parameters.
* The **return** keyword ensures control over method outputs and flow.
* **Varargs** simplifies handling methods with flexible argument counts.