1. Method

A **method** is a **block of code** that performs a specific task. It is **executed only when it is called** (or invoked) from another part of the program.

Methods help in:

* **Reusability** → You can write code once and use it many times.
* **Readability** → Code becomes easier to understand and maintain.
* **Modularity** → Each method handles one specific function.

Example Code :-

public class Example {

// Method definition

public static void greet() {

System.out.println("Hello, welcome to Java!");

}

public static void main(String[] args) {

// Calling the method

greet();

}

}

1. what is Method overloading and Method overriding

Method Overloading :-

When two or more methods in the same class have the **same name** but **different parameters**, it is called **method overloading.**

Example Code :-

class Calculator {

// Method 1

int add(int a, int b) {

return a + b;

}

// Method 2 (Overloaded)

double add(double a, double b) {

return a + b;

}

// Method 3 (Overloaded)

int add(int a, int b, int c) {

return a + b + c;

}

}

public class Main {

public static void main(String[] args) {

Calculator calc = new Calculator();

System.out.println(calc.add(5, 10)); // calls int add(int, int)

System.out.println(calc.add(5.5, 6.5)); // calls double add(double, double)

System.out.println(calc.add(1, 2, 3)); // calls int add(int, int, int)

}

}

Method Overriding :

*When a subclass (child class) defines a method with the* ***same name****,* ***same parameters****, and* ***same return type*** *as a method in its superclass (parent class), it is called* ***method overriding.***

class Animal {

void sound() {

System.out.println("Animal makes a sound");

}

}

class Dog extends Animal {

// Method overriding

void sound() {

System.out.println("Dog barks");

}

}

public class Main {

public static void main(String[] args) {

Animal obj = new Dog(); // parent reference, child object

obj.sound(); // calls Dog's version → "Dog barks"

}

}

**Return types**

In Java, every **method** has a **return type**, which tells the compiler **what kind of value** the method will give back after it finishes running. If a method **does not return anything**, its return type is written as **void**.

|  |  |  |
| --- | --- | --- |
| Return Type |  | Returns |
| void |  | Nothing |
| int |  | Integer (e.g. 15) |
| double |  | Decimal number (e.g. 99.99) |
| String |  | Text (e.g. “Hello”) |
| boolean |  | true / false |

**Recursion**

**Recursion** means a **method calls itself** to solve a problem. It continues calling itself **repeatedly** until a **base condition** is met. Without a base condition, the recursion would run **infinite times** and cause a **stack overflow error**.

Simple Example of Recursion:-

class RecursionExample {

static void printNumbers(int n) {

if (n > 5) // base condition (stop condition)

return;

System.out.println(n);

printNumbers(n + 1); // recursive call

}

public static void main(String[] args) {

printNumbers(1);

}

}

Output:

1

2

3

4

5

**Stack Trace:**

Each time a method calls itself, a new **stack frame** is created in the **memory**.

Java stores each function call in this **stack** until it completes. When the **base condition** is reached, the stack starts returning back.

Step-by-Step Stack Trace:

| **Step** | **Call** | **Action** | **Stack (top → bottom)** |
| --- | --- | --- | --- |
| 1 | printNumbers(1) | calls printNumbers(2) | 2 → 1 |
| 2 | printNumbers(2) | calls printNumbers(3) | 3 → 2 → 1 |
| 3 | printNumbers(3) | calls printNumbers(4) | 4 → 3 → 2 → 1 |
| 4 | printNumbers(4) | calls printNumbers(5) | 5 → 4 → 3 → 2 → 1 |
| 5 | printNumbers(5) | calls printNumbers(6) (base reached, returns) | Stack unwinds |

Important Questions:-

1. Write a Java program to perform addition of two numbers and return the value from the method.
2. Write a Java program to find the **factorial of a given number** using **recursion**.
3. Write a Java program to **display the Fibonacci series** up to ‘n’ terms using **recursion**.
4. Write a Java program to find the **GCD (Greatest Common Divisor)** of two numbers using **recursion**.
5. Write a Java program to perform the of two strings and return the combined value from the method.