**Flow of execution of function**

The flow of execution of a function (or method) in Java refers to the order in which statements are executed within methods and how control moves between different methods during a program's execution. Here's a step-by-step explanation of the flow of execution in Java:

**1. Program Starts Execution from the main Method**

The main method is the entry point of any Java program. Execution begins from this method.

public class ExecutionFlowExample {

public static void main(String[] args) {

System.out.println("Main method starts.");

exampleMethod();

System.out.println("Main method ends.");

}

public static void exampleMethod() {

System.out.println("Example method is executed.");

}

}

**2. Method Invocation**

When a method is called, control is transferred from the calling method to the called method. The statements inside the called method are executed sequentially.

public class ExecutionFlowExample {

public static void main(String[] args) {

System.out.println("Main method starts."); // Step 1

exampleMethod(); // Step 2

System.out.println("Main method ends."); // Step 4

}

public static void exampleMethod() {

System.out.println("Example method is executed."); // Step 3

}

}

**3. Returning from a Method**

After the called method completes execution, control returns to the calling method, right after the point where the method was called.

**4. Calling Methods with Parameters and Return Values**

Methods can take parameters and return values. When a method is called with parameters, the provided arguments are passed to the method. If the method returns a value, the control returns to the calling method along with the returned value.

public class ExecutionFlowExample {

public static void main(String[] args) {

System.out.println("Main method starts."); // Step 1

int result = add(5, 3); // Step 2

System.out.println("The result is: " + result); // Step 4

System.out.println("Main method ends."); // Step 5

}

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

return a + b; // Step 3

}

}

**5. Nested Method Calls**

Methods can call other methods, creating a nested flow of execution. Control transfers to the innermost method first and then returns back outward as each method completes.

public class ExecutionFlowExample {

public static void main(String[] args) {

System.out.println("Main method starts."); // Step 1

firstMethod(); // Step 2

System.out.println("Main method ends."); // Step 6

}

public static void firstMethod() {

System.out.println("First method starts."); // Step 3

secondMethod(); // Step 4

System.out.println("First method ends."); // Step 5

}

public static void secondMethod() {

System.out.println("Second method is executed."); // Step 4

}

}

**6. Recursive Method Calls**

A method can call itself. This is known as recursion. Care must be taken to ensure that there is a base case to stop the recursive calls and avoid infinite recursion.

public class ExecutionFlowExample {

public static void main(String[] args) {

System.out.println("Factorial of 5 is: " + factorial(5)); // Step 1

}

public static int factorial(int n) {

if (n == 0) { // Base case

return 1; // Step 3 (when n reaches 0)

} else {

return n \* factorial(n - 1); // Recursive call

}

}

}