
Java Main Method & Methods

1. Introduction to Java Program Execution

- **Application Software / Program** → All are the same.
- **Start point** must be defined → `main` method.
- The **Operating System (OS)** interacts with the program.

---

2. Java Program Execution Flow

1. **Java Code** → Compiled by **Java Compiler** → Converted to **Bytecode**.
2. **Java Virtual Machine (JVM)** executes the Bytecode.
3. **JVM** looks for the `main` method as the entry point.

---

3. What is a Method?

A **method** is a **task, work, or activity** that a program performs.

Method Syntax

```
returnType methodName(parameters) {
    // Body of the method
}
```

- **Return Type** → Specifies the type of value returned (e.g., `void`, `int`).
- **Method Name** → The identifier of the method.
- **Parameters** → Input values for the method.
- **Body** → Code that performs the task.

Example Method

```
void add(int a, int b) {
    int sum = a + b;
    System.out.println(sum);
}
```

- **Method Name**: `add`

- **Parameters**: `a`, `b`
- **Return Type**: `void` (no return value)

---

4. Java `main` Method

- The **main method** is the starting point of execution in a Java program.
- **Syntax:**

```
public class Example {
    public static void main(String[] args) {
        System.out.println("Hello, Java!");
    }
}
```

Key Points

- **`public`** → Accessible from anywhere.
- **`static`** → No need to create an object to call it.
- **`void`** → Doesn't return a value.
- **`main`** → Standard method name that JVM looks for.
- **`String[] args`** → Command-line arguments.

---

5. Java Program Structure

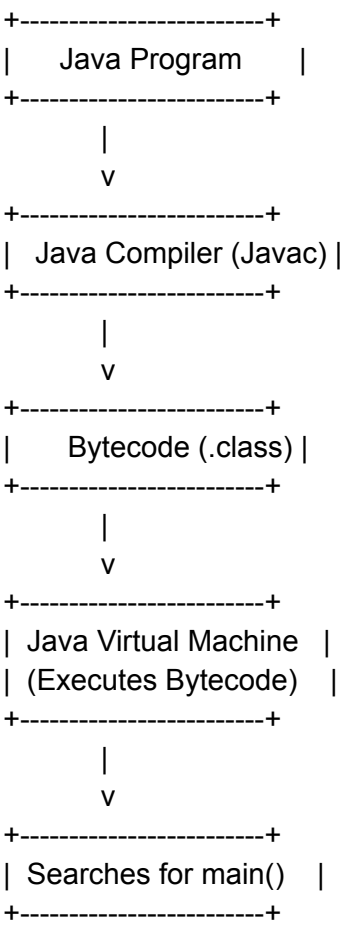
Example of a Java Class with Methods

```
class Launch {
    public static void main(String[] args) {
        System.out.println("Hello, Java!");
    }
}
```

- **Class Name**: `Launch`
- **Method**: `main`
- **Object**: JVM calls the `main` method to start execution.

Diagram Representation

Here’s a simplified version of how Java execution works:



I see more detailed notes on the **Java `main` method**, covering topics like `public`, `static`, `void`, and `String[] args`. I'll format this into **structured digital notes** with diagrams for clarity.

Java `main` Method - Detailed Explanation

1. Role of `main` Method

- The **Java Virtual Machine (JVM)** searches for the `main` method as the **entry point** of a program.
- **Syntax:**

```
public static void main(String[] args) {
   System.out.println("Hello, Java!");
}
```

---

2. Components of `main` Method

(a) `public` (Access Modifier)

- **Purpose:** Increases visibility, making `main` accessible from anywhere.
- **Why?** JVM needs to call `main` from outside the class.

(b) `static` (Method Type)

- **Purpose:** Allows calling `main` without creating an object of the class.
- **Why?** JVM directly calls `main` without needing an instance.

(c) `void` (Return Type)

- **Purpose:** Indicates `main` does **not return any value**.
- **Why?** The JVM does not expect any return from `main`.

(d) `String[] args` (Command-Line Arguments)

- **Purpose:** Accepts inputs from the command line as an array of strings.
- **Example Usage:**

```
public class Example {
    public static void main(String[] args) {
        System.out.println(args[0]); // Prints the first argument
    }
}
```

Command:

java Example Hello

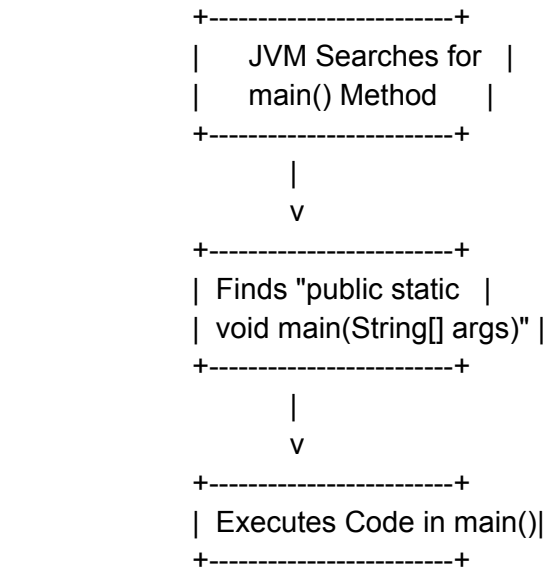
Output:

Hello

---

3. Execution Flow of `main` Method

Diagram:



---

4. Alternative Syntax for `main`

Both forms are valid:

```
public static void main(String args[]) { }
public static void main(String[] args) { } // Preferred
```

Why? No difference in execution—just stylistic variation.

---

5. Example Java Program Using `main`

```
public class Launch {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.println("First Argument: " + args[0]);
    }
}
```

Command to Run:

java Launch Java

Output:

Hello, World!
First Argument: Java

Your notes cover variables, assignment operations, and the difference between statically and dynamically typed languages. I'll structure them clearly with tables and diagrams for easy understanding.

Java Variables and Data Types

1. What is a Variable?

- **A variable is a container that stores data or information in memory.**
- **It consists of:**
  - **Variable name → Identifier (e.g., `a`)**
  - **Assignment operator (`=`) → Assigns a value**
  - **Value → Data stored in the variable**

Example:

```
int a = 10;
```

Here:

- **`int` → Data type (integer)**
- **`a` → Variable name**
- **`10` → Value assigned to `a`**

Diagram:

```
+---------+

|  a = 10 |

+---------+

  (Memory)
```

2. Statically Typed vs. Dynamically Typed Languages

Feature	Statically Typed	Dynamically Typed
Definition	Type is checked at compile time	Type is checked at runtime
Example Languages	C, C++, Java	Python, JavaScript, TypeScript
Declaration	Must specify data type (`int a = 10;`)	No need to specify type (`a = 10`)
Error Detection	Detects errors before execution	Errors may occur during execution

Example:

Statically Typed (Java)

```
int a = 10;

System.out.println(a);
```

Dynamically Typed (Python)

```
a = 10

print(a)
```

3. Key Differences

Concept	Statically Typed	Dynamically Typed
Type Checking	Compile-time	Runtime
Flexibility	Less flexible but safer	More flexible but riskier
Performance	Faster (predefined types)	Slower (checks type at runtime)

---

4. Why is Typing Important?

- **Statically typed languages help avoid errors before running the program.**
- **Dynamically typed languages provide flexibility but may cause unexpected runtime errors.**