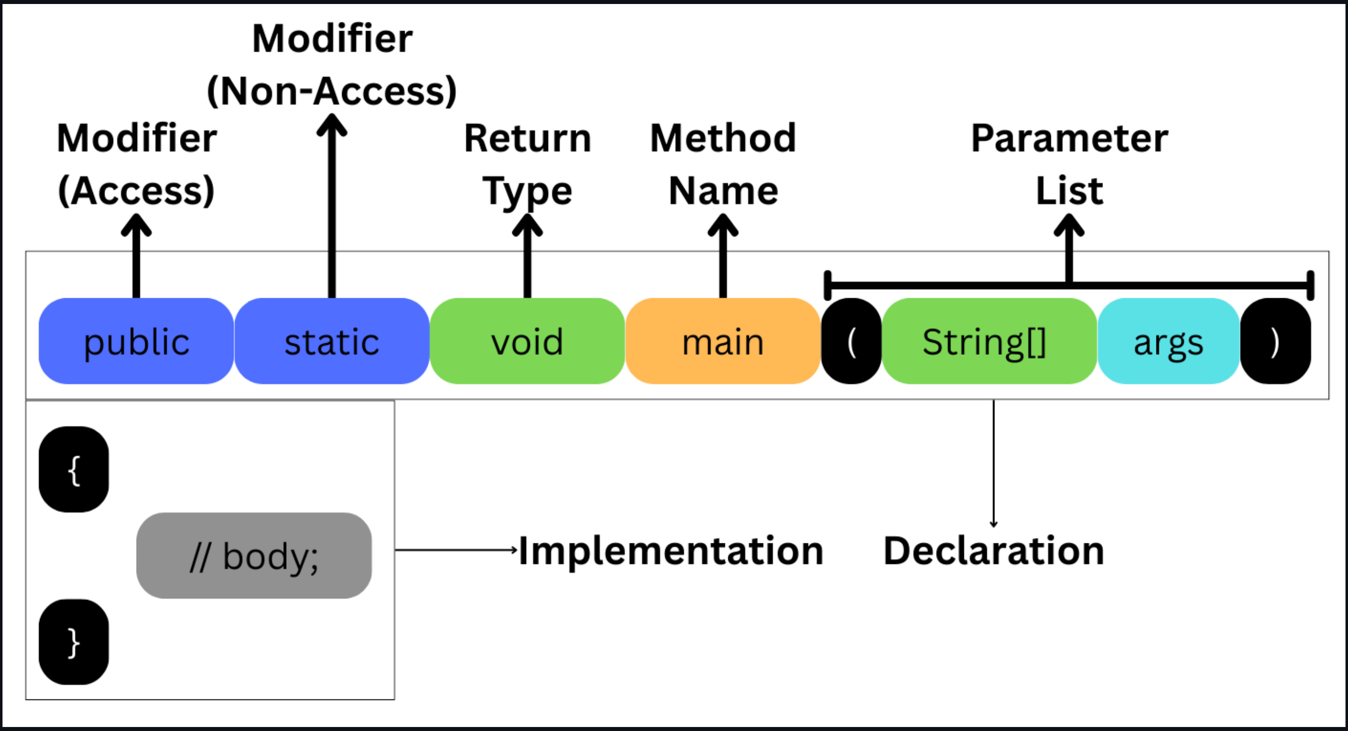
Object Oriented Programming Concepts

Class

- A `class` is a **blueprint** or **template** used to create objects.
- It defines the **properties** and **behaviors** shared by its instances.
- It represents a *conceptual boundary* encapsulating the logic and data.

Method

- A method represents the **behavior** of a class.
- It is a **block of code** that contains business logic or functionality.
- A method consists of:
  - *Declaration*
  - *Implementation*



- Method names follow the **camelCase** convention.
- Multiple methods can exist in a class if they differ by **name** or **parameter list**.
- The `main` method is the **entry point** of program execution by the **JVM**.

- Methods cannot be declared within another method.

Types of Methods

- **Static Methods** — declared with the `static` keyword.
- **Non-Static Methods** — declared without the `static` keyword.

```java
class TypesOfMethod {
    public static void staticMethod() {
        System.out.println("This is a Static Method.");
    }

    public void nonStaticMethod() {
        System.out.println("This is a Non-Static Method.");
    }
}
```

Calling Static and Non-Static Methods

- **Static methods** can be called:
  - Directly
  - Using the class name
  - Using an object reference
- **Non-static methods** can only be called using an object reference.

```java
class CallingOfMethod {
    public static void staticMethod() {
        System.out.println("This is a Static Method.");
    }

    public void nonStaticMethod() {
        System.out.println("This is a Non-Static Method.");
    }

    public static void main(String[] args) {
        CallingOfMethod ob = new CallingOfMethod();

        // Static Method calls
        staticMethod();
        CallingOfMethod.staticMethod();
        ob.staticMethod();
        (new CallingOfMethod()).staticMethod();

        // Non-Static Method calls
        ob.nonStaticMethod();
        (new CallingOfMethod()).nonStaticMethod();
    }
}
```

- Execution always begins from the `main` method.
- Methods can call other methods.

```java
class StaticNonStatic {
    public void func() {
        System.out.println("Function");
    }

    public void func1() {
        (new StaticNonStatic()).func();
        System.out.println("Function1");
    }

    public void func2() {
        StaticNonStatic obj = new StaticNonStatic();
        obj.func1();
        System.out.println("Function2");
    }

    public static void main(String[] args) {
        StaticNonStatic ob = new StaticNonStatic();
        ob.func2();
    }
}
```

- Declaring a method does not require invoking it.
- A method must be **declared before being called**; otherwise, it causes a **compilation error**.

```java
class CompilationError {
    public static void func2() {
        System.out.println("Hii");
    }

    public static void main(String[] args) {
        CompilationError ob = new CompilationError();
        ob.func1(); // Error: func1 not declared
    }
}
```

Arguments

- Arguments are **inputs** passed to a method.
- Method types based on arguments:
    - **No-argument Method** — has no parameters.
    - **Argumented Method** — has one or more parameters.

```java
class Arguments {
    public static void func(int x, char ch, String str, Arguments ob)
{
        System.out.println(x + " " + ch + " " + str + " " + ob);
    }

    public static void main(String[] args) {
        Arguments.func(12, 'q', "Sambit", null);
        Arguments.func('A', 'q', null, new Arguments());
        Arguments.func(12, 'q', new String(), null);
    }
}
```

Method Signature

- A method signature is defined by the **method name** and the **types of its parameters**.
- A class cannot have two methods with the **same signature**.

```java
class MethodSignature {
    public static void func(int x) {}
    public static void func(int x, int y) {}
    public static void func1(int x, String str) {}
    public static void main(String[] args) {}
}
```

Return Type

- Specifies the **type of value** a method returns.
- Valid return types:
  - `void`
  - **Primitive type**
  - **Non-primitive type**
- If not `void`, a **return statement** is mandatory.
- Returned value must match or be convertible to the declared type.

```java
class ReturnType {
    public int func1() {
        return 12;
    }

    public String func2() {
        return "12";
    }

    public static void main(String[] args) {
        ReturnType ob = new ReturnType();
        int result = ob.func1();
        System.out.println(result);
        System.out.println(ob.func2());
    }
}
```

- Return value can be printed directly or stored in a variable.
- Code after a `return` statement is **unreachable** and will cause a compile-time error.

Var-Arg (Variable-Length Argument)

- A **var-arg** can accept zero or more values.
- A **var-arg method** accepts a variable number of arguments.

```java
class VarArgMethod {
    public void varArgMethod(int... varArg) {
        System.out.println("Hello...");
        for (int i : varArg)
            System.out.println(i);
        for (int i = 0; i < varArg.length; i++)
            System.out.println(varArg[i]);
    }

    public static void main(String[] args) {
        VarArgMethod ob = new VarArgMethod();
        ob.varArgMethod();
        ob.varArgMethod(1);
        ob.varArgMethod(1, 2);
        ob.varArgMethod(1, 2, 3);
    }
}
```

Var-Arg Rules

- **Exact match** is prioritized over var-arg.

```java
class Rule1 {
    public void func(int... varArg) {
        System.out.println("Hello VAR ARG...");
    }

    public void func(int arg) {
        System.out.println("Hello Single ARG...");
    }

    public static void main(String[] args) {
        (new Rule1()).func(1); // Calls single argument version
    }
}
```

- **Up-cast match** is prioritized over var-arg.

```java
class Rule2 {
    public void func(int... varArg) {
        System.out.println("Hello VAR ARG...");
    }

    public void func(long arg) {
        System.out.println("Hello Single ARG...");
    }

    public static void main(String[] args) {
        (new Rule2()).func(1); // Calls long version
    }
}
```

- Only one var-arg parameter is allowed per method.
- The var-arg parameter must be the **last** in the parameter list.

```java
// Invalid declaration
// public void func(int... a, int... b); // Compile-time Error
```