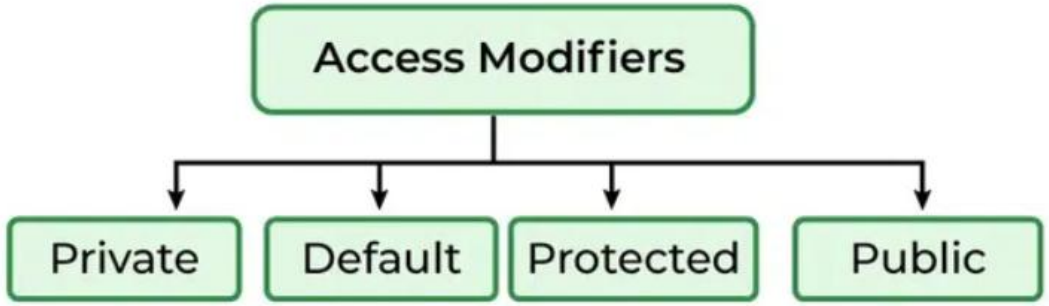
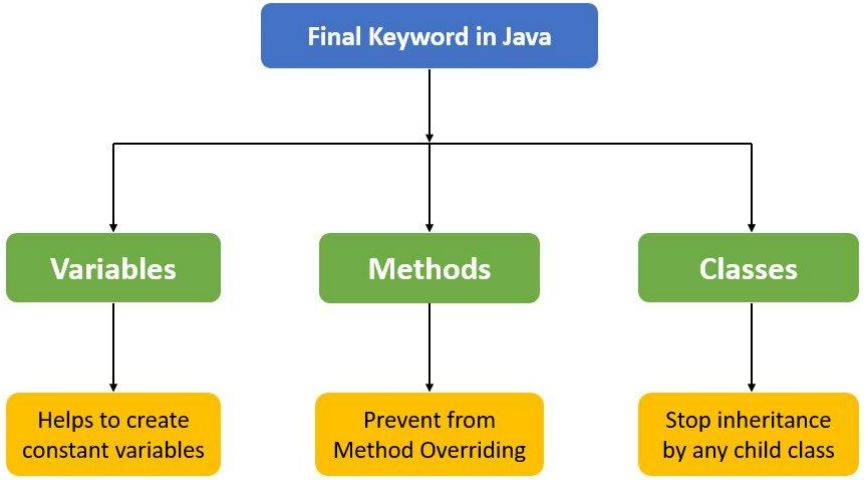


Access Modifiers, Methods and Final Keyword



Nimesha Hewawasam
Lecturer
nimesha.h@nsbm.ac.lk

Access Modifiers in Java

- **public**: *Accessible everywhere*
  - Specified using the keyword ***public***
  - Has widest scope among all other access modifiers.
  - Classes, methods, or data members that are **declared as public** are **accessible from everywhere in the program.**
  - No restriction on the scope of public data members.

```
class Student {
    public String name = "Nimal";
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        System.out.println(s.name); // Allowed
    }
}
```

Access Modifiers in Java

- **private**: *Accessible only within the class*
  - specified using the keyword ***private***
  - methods or data members **declared as private** are **accessible only within the class** in which they are declared.
  - Any other class of the **same package will not be able to access** these members.
  - Top-level classes or interfaces can not be declared as private because, private means "**only visible within the enclosing class**".

```java
class Student {
    private String name = "Nimal";

    public String getName() {
        return name;
    }
}

public class Main {
    public static void main(String[] args) {
        Student s = new Student();
        // System.out.println(s.name); // Error: name has private access
        System.out.println(s.getName()); // OK
    }
}
```

Access Modifiers in Java

- **protected**: *Accessible in same package and subclasses*
  - specified using the keyword ***protected***
  - The methods or data members declared as protected are **accessible within the same package or subclasses in different packages.**

```
class Person {
    protected String name = "Nimal";
}

class Student extends Person {
    void display() {
        System.out.println(name); // Accessible in subclass
    }
}
```

Access Modifiers in Java

- **default** (no keyword): **Accessible within same package**
  - When ***no access modifier*** is specified for a class, method, or data member, it is said to have the **default access** modifier by default.
  - This means only classes within the same package can access it.

```
class Student {
    String name = "Nimal"; // default access
}

class Main {
    public static void main(String[] args) {
        Student s = new Student();
        System.out.println(s.name); // OK (same package)
    }
}
```

Comparison Table of Access Modifiers in Java

Context	Default	Private	Protected	Public
Same Class	☑ Yes	☑ Yes	☑ Yes	☑ Yes
Same Package Subclass	☑ Yes	❌ No	☑ Yes	☑ Yes
Same Package Non-Subclass	☑ Yes	❌ No	☑ Yes	☑ Yes
Different Package Subclass	❌ No	❌ No	☑ Yes	☑ Yes
Different Package Non-Subclass	❌ No	❌ No	❌ No	☑ Yes

Java Method Parameters

- *Parameters and Arguments*
  - Information can be passed to methods as a parameter.
  - Parameters act as variables inside the method.
  - Parameters are specified after the method name, inside the parentheses.
  - You can add as many parameters as you want, just separate them with a comma.

```java
public class Main {

    // This method takes one parameter: 'fname' (a String)
    static void myMethod(String fname) {
        // Prints the value of fname with the word 'Refsnes' added
        System.out.println(fname + " Refsnes");
    }

    public static void main(String[] args) {
        // Passing different arguments to the method
        myMethod("Liam");   // Output: Liam Refsnes
        myMethod("Jenny");  // Output: Jenny Refsnes
        myMethod("Anja");   // Output: Anja Refsnes
    }
}
```

Return Values

- We used the **void** keyword in all examples, which indicates that **the method should not return a value**.
- If you want the method to **return a value**, you **can use a primitive data type** (such as **int, char**) instead of void, and use the **return** keyword inside the method.

```java
public class Calculator {

    // Parameters: 'int a' and 'int b'
    public int add(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        Calculator calc = new Calculator();

        // Arguments: 5 and 3
        int result = calc.add(5, 3);

        System.out.println("Sum is: " + result); // Output: Sum is: 8
    }
}
```

Method Overloading

- Allows us to define multiple methods with the same name but different parameters within a class. This difference may include:
  - The number of parameters
  - The types of parameters
  - The order of parameters

```java
public class Calculator {

    // Method 1: add two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Method 2: add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // Method 3: add two doubles
    public double add(double a, double b) {
        return a + b;
    }

    public static void main(String[] args) {
        Calculator calc = new Calculator();

        System.out.println("add(5, 10): " + calc.add(5, 10));             // 15
        System.out.println("add(5, 10, 20): " + calc.add(5, 10, 20));     // 35
        System.out.println("add(5.5, 2.3): " + calc.add(5.5, 2.3));       // 7.8
    }
}
```

final Keyword in Java

- final keyword in Java is a non-access modifier used to prevent modification.
- It can be applied to variables (value cannot change), methods (cannot be overridden) and classes (cannot be extended).

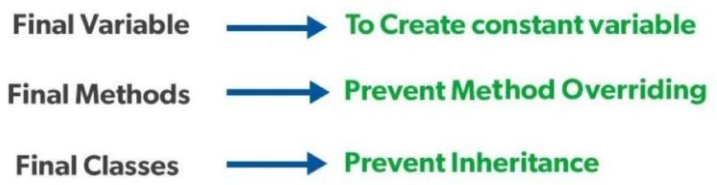
```java
public class Constants {

    // Declare a final variable (constant)
    final int MAX_STUDENTS = 100;

    public static void main(String[] args) {
        Constants obj = new Constants();

        // ❌ This line will cause a compile-time error
        obj.MAX_STUDENTS = 200; // Cannot assign a value to final variable

        System.out.println("Max students: " + obj.MAX_STUDENTS);
    }
}
```



Thank you!