Static vs. Non-Static method

You will often see Java programs that have either **static or public attributes and methods**. In the following example:

```
public class Main {
 //Static method
  static void myMethod (){
    System.out.println("Hello World!") ;}

  public static void main(String[] args) {
    myMethod();} }
```

We created a **static method**, which **means that it can be accessed without creating an object of the class**, unlike **public**, which **can only be accessed by objects**.

Example

An example to demonstrate the **differences between static and public methods**:

```
public class Main {
 //Static method
static void myStaticMethod (){
System.out.println("Static methods can be called without creating objects");}

 //Public method
public void myPublicMethod (){
System.out.println("Public methods must be called by creating objects");}

public static void main(String[] args) {

myStaticMethod(); // Call the static method
myPublicMethod(); //This would compile an error, comment this statement

Main myObj = new Main(); // Create an object of Main
myObj.myPublicMethod(); // Call the public method on the object
}}
```

Example of static variable

```
class Student8{
int rollno;
String name ;

 //Static variable

static String college ="ITS ";

// Constructor with two parameters
Student8(int r,String n){
rollno = r ;
name = n ;}

void display (){
   System.out.println(rollno+" "+name+" "+college);}

public static void main(String[] args){
Student8 s1 = new Student8(111,"Karan") ;
Student8 s2 = new Student8(222,"Aryan") ;

s1.display ();
s2.display (); }}
```

Output:

```
111 Karan ITS
222 Aryan ITS
```

The final Keyword

Modifiers like final play a crucial role in **improving code stability and preventing unexpected changes**. The final keyword in Java is **widely used by developers** to **secure variables**, **methods**, and **classes from being altered**.

It is a powerful tool to enhance code readability, enforce immutability, and **avoid accidental overrides or reassignments**. The use of final in Java helps in **building safer and more efficient applications**.

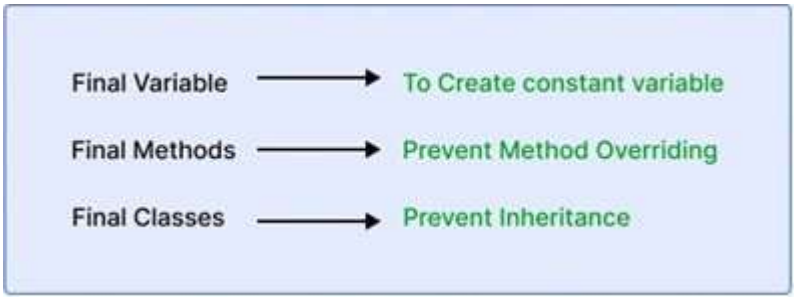
Let's learn how **final works with variables, methods, and classes**, common mistakes to avoid, real-world examples, and best practices every Java developer should know.

What is final Keyword in Java?

The **final** keyword in Java is a **non-access modifier** used to **restrict the user from making further changes to variables, methods, or classes**.

- **When a variable is declared final, its value cannot be reassigned**.
- **When a method is marked final**, **it cannot be overridden by subclasses**.
- And when a **class is declared final**, it **cannot be extended or inherited**.

The Java **final keyword** is essential for **writing secure**, **predictable**, and **maintainable code**. It is commonly used to **define constants**, **prevent method overriding, and ensure immutability in classes and objects**.



Final Variable in Java

A final variable in Java is a variable whose **value cannot be changed once it is assigned**. Declaring a variable as **final means you are telling the compiler that this variable is a constant**. It must be **initialized either at the time of declaration or inside the constructor** (in case of instance variables).

If you try to reassign a new value to a final variable, **the Java compiler will throw an error**. The use of final keyword in Java for variables ensures data consistency, prevents accidental modifications, and is **good practice when working with constants or fixed configuration values**.

Syntax of final variable

final dataType variableName = value;

Components:

- final: keyword to make variable constant
- dataType: like int, String, float, etc.
- variableName: any valid identifier
- value: value assigned once and never changed

Example of final variable

1.

```
public class BankAccount {

   // final variable

   final String BANK_NAME = "Secure Bank Ltd.";

   private String accountHolder;
   private double balance;

   public BankAccount(String accountHolder, double initialBalance) {
      this.accountHolder = accountHolder;
      this.balance = initialBalance;    }

   public void displayAccountInfo() {
      System.out.println("Bank Name      : " + BANK_NAME);
      System.out.println("Account Holder : " + accountHolder);
      System.out.println("Balance        : $" + balance);     }

   public static void main(String[] args) {
     // create object
      BankAccount account = new BankAccount("Riya Sharma", 15000.0);
```

```
        account.displayAccountInfo();
        // Trying to change the final variable will cause a compile-time error:
        // account.BANK_NAME = "Other Bank";    } }
```

Output:

```
Bank Name      : Secure Bank Ltd.
Account Holder : Riya Sharma
Balance        : $15000.0
```

2.

```
public class Circle {

  // final variable

    private final double PI = 3.14159; // Constant variable
    private double radius;

    public Circle(double radius) {
        this.radius = radius;   }

    public double calculateArea() {
        return PI * radius * radius;    }

    public static void main(String[] args) {
        Circle circle = new Circle(5.0);
        double area = circle.calculateArea();
        System.out.println("Area of the circle: " + area); }}
```

Output

```
78.53975
```

Key Features of Final Variables in Java

- **Must be initialized only once**
- **Cannot be reassigned later**
- Improves code safety and clarity
- Final instance variables can be initialized in constructor
- **Final static variables act like constants**

Final Method in Java

A final method in Java is a **method that cannot be overridden by subclasses**. When a method is declared using the **final keyword**, it locks the implementation and prevents any changes through inheritance. **This is especially useful when you want to enforce business rules, critical logic, or base functionality that should remain unchanged.**

The use of final keyword in Java for methods ensures consistency, prevents misuse through subclass modifications, and improves security in code. It also gives the compiler freedom to optimize performance since the method's behavior is guaranteed not to change.

Syntax of final method

```
final returnType methodName(parameters) {
   // method body
}
```

Components:

- **final:** prevents overriding
- **returnType:** like void, int, String, etc.
- **methodName:** name of the method
- **parameters:** method input (optional)

Example of final method

1.

```
class ATM {
  // final method
   final void showWelcomeMessage() {
      System.out.println("Welcome to Secure ATM!");    }

   void withdraw(double amount) {
      System.out.println("Please collect $" + amount);  }}
```

```
class UserATM extends ATM {
   // Overriding final method below would cause an error

    void showWelcomeMessage() {
       System.out.println("Hacked ATM Message!");}

   void withdraw(double amount) {
      super.withdraw(amount);
      System.out.println("Transaction Complete.");  }}

public class FinalMethodExample {
   public static void main(String[] args) {
      UserATM user = new UserATM();
      user.showWelcomeMessage();
      user.withdraw(2000);  }}
```

Output:
Overriding final method below would cause an error

2.

```
public class Parent {

  // final method

   public final void printMessage() {
   System.out.println("This is a final method in the Parent class"); }

public class Child extends Parent{
        // Overriding final method below would cause an error

    void printMessage() {
         System.out.println("This is a final method in the Child class.");}}

public static void main(String[] args) {
Child child = new Child();
child.printMessage();}}
```

In this example, the printMessage() method in the Parent class is declared final. This means that any subclass cannot override it.

Key Features of Final Method in Java

- **Cannot be overridden in subclasses**
- Ensures consistent behavior in inherited classes
- Enhances security and integrity of business-critical logic
- Allows compiler optimizations
- Helps in API development where some methods should not be changed

Final Class in Java

The **final class in Java is a class that cannot be extended (inherited) by any other class.** When a class is declared with the final keyword, it effectively prevents subclassing and modification of its behavior. This is **useful for creating secure**, immutable, and reliable code, especially in libraries and utility classes.

The use of final keyword in Java for classes ensures that the core functionality of the class remains intact and protected from unintended changes through inheritance. The most well-known example is the String class in Java, which is declared final to maintain immutability and prevent tampering.

Syntax

```
final class ClassName {
    // class members
}
```

Components:

- **final:** keyword to prevent inheritance
- **class ClassName:** name of the class

Example of final class in Java

```
final class Student {
    private final String name;
    private final int rollNumber;
```

```
    public Student(String name, int rollNumber) {
        this.name = name;
        this.rollNumber = rollNumber;     }

    public void displayInfo() {
        System.out.println("Name      : " + name);
        System.out.println("Roll No.  : " + rollNumber);  }}

 // This will cause a compile-time error

// class EngineeringStudent extends Student { }

public class FinalClassExample {
    public static void main(String[] args) {
        Student s1 = new Student("Aryan Verma", 101);
        s1.displayInfo(); }}
```

Output:

```
Name      : Aryan Verma
Roll No.  : 101
```

Key Features of Final Classes in Java

- **Cannot be inherited by any other class**
- Secure against alteration through subclassing
- **Often combined with final variables for immutability**
- Useful in API design to lock behavior
- **Compile-time error on any attempt to extend**

Examples

1.

```
public class FinalVariableExample {
    public static void main(String[] args) {
        // Declare and initialize a final variable
        final int MAX_RETRIES = 3;
        System.out.println("Maximum retries allowed: " + MAX_RETRIES);
```

```
        // Attempting to change the value results in a compilation error:
        // MAX_RETRIES = 5; // Error: cannot assign a value to final variable
MAX_RETRIES
    }
}
```

2.

```
public class StudentData {
    // Blank final instance variable (not initialized at declaration)
    final int ROLL_NO;

   // Constructor to initialize the blank final variable

    public StudentData(int rollNum) {
        ROLL_NO = rollNum; // Value assigned here, only once
    }

    public void displayRollNo() {
        System.out.println("Roll no is: " + ROLL_NO); }

    public static void main(String[] args) {
        StudentData obj = new StudentData(12345);
        obj.displayRollNo();

        // obj.ROLL_NO = 67890; // Error: cannot assign a value to final variable
ROLL_NO
    }
}
```

Examples of Java final Method

1.

```
class FinalDemo {

   // create a final method

   public final void display() {
      System.out.println("This is a final method.");   }}
```

```
class Main extends FinalDemo {
  // try to override final method

  public final void display() {
    System.out.println("The final method is overridden.");  }

  public static void main(String[] args) {
    Main obj = new Main();
    obj.display(); }}
```

output

```
// display() in Main cannot override display() in FinalDemo
  public final void display() {
                    ^
  overridden method is final
```

Examples of Java final Class

```
// create a final class

final class FinalClass {
  public void display() {
    System.out.println("This is a final method.");  }}

// try to extend the final class
class Main extends FinalClass {
  public  void display() {
    System.out.println("The final method is overridden.");  }

  public static void main(String[] args) {
    Main obj = new Main();
    obj.display(); }}
```

output

```
cannot inherit from final FinalClass
class Main extends FinalClass {
                   ^
```