In Java, the `final` keyword is used to indicate that something cannot be changed. When applied to a **class**, **method**, or **variable**, it has specific meanings:

1. `final` Class

A **final class** is a class that cannot be subclassed (inherited). Once a class is marked as `final`, no other class can extend it. This is useful when you want to make sure that the behavior of the class is not modified through inheritance.

Characteristics of a Final Class:

- **Cannot be subclassed**: Once a class is declared as `final`, no other class can inherit from it.
- **Used to protect the class**: By making a class `final`, you ensure that no subclass can change its functionality or behavior.
- **Used in performance-sensitive code**: The `final` keyword can also provide optimizations for the Java Virtual Machine (JVM) since it knows that no subclasses can exist, potentially allowing more efficient method calls.

Example of a Final Class:

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

// This will give an error:
// class Car extends Vehicle {}  // Error: cannot inherit from final
Vehicle class
```

In this example, the `Vehicle` class is marked as `final`, so it cannot be extended by any other class, like `Car` in the commented code.

2. `final` Method

A **final method** is a method that cannot be overridden by any subclass. Once a method is marked as `final`, its behaviour is fixed and cannot be changed, even in subclasses.

Characteristics of a Final Method:

- **Cannot be overridden**: If a method is `final`, any subclass that attempts to override it will result in a compile-time error.
- **Used for safety**: You may want to prevent subclasses from altering the behavior of critical methods.
- **Useful in method implementations**: If you have a method that implements critical functionality and you don't want it to be changed, you can make it `final`.

Example of a Final Method:

```
class Animal {
    // Final method cannot be overridden
    public final void sound() {
        System.out.println("Some animal sound");
    }
}

class Dog extends Animal {
    // Trying to override the final method will result in an error
    // public void sound() {  // Error: cannot override the final method
from Animal
    //     System.out.println("Woof!");
    // }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.sound();  // Outputs: Some animal sound
    }
}
```

- The `sound()` method in the `Animal` class is marked as `final`, so it cannot be overridden by the `Dog` class.
- If you try to override `sound()` in the `Dog` class, you will get a compile-time error.

3. `final` Variable

A **final variable** is a variable whose value cannot be changed after it has been initialized. It can be a **primitive** type or a **reference** type (like an object).

Characteristics of a Final Variable:

- **Cannot be reassigned**: Once assigned a value, the variable cannot be reassigned.
- **Constant Values**: Often used for constants, especially when you want to ensure that a value remains unchanged.

Example of a Final Variable:

```
class Test {
    public static final int MAX_VALUE = 100;  // Constant variable

    public static void main(String[] args) {
        // MAX_VALUE cannot be reassigned
        // MAX_VALUE = 200;  // This would result in a compile-time error
        System.out.println("Max Value: " + MAX_VALUE);
    }
}
```

In this example:

- `MAX_VALUE` is a `final` variable, and its value cannot be changed after initialization.
- It is also common to use `final` with `static` to create **constants** (like `MAX_VALUE`), which are often written in uppercase letters by convention.

When to Use the `final` Keyword:

1. **Final Classes**:
   - When you want to prevent inheritance. For example, you might create a class like `String` in Java that should not be sub classed to maintain its behaviour.
2. **Final Methods**:
   - When you want to ensure the method's behaviour cannot be changed by subclasses. For example, in security-related classes or core library classes, you may want to prevent altering behaviour.
3. **Final Variables**:
   - When you want to create constants or variables whose values should not be changed once initialized. This is common for configuration values or fixed constants in the code.

Example Combining All `final` Concepts:

```
// Final Class Example
final class Calculator {
    // Final method example
    public final int add(int a, int b) {
        return a + b;
    }

    public final int subtract(int a, int b) {
        return a - b;
    }
}

public class Main {
    public static void main(String[] args) {
        // Trying to subclass the final class will result in an error:
        // class AdvancedCalculator extends Calculator {}  // Error: cannot
inherit from final Calculator class

        Calculator calc = new Calculator();
        System.out.println("Sum: " + calc.add(5, 3));  // Outputs: Sum: 8
        System.out.println("Difference: " + calc.subtract(5, 3));  //
Outputs: Difference: 2
    }
}
```

In this example:

- `Calculator` is a **final class**, so it cannot be subclassed.
- The methods `add()` and `subtract()` are **final methods**, so they cannot be overridden in any subclass (if it were allowed).

1. **Final Class**: A class declared as `final` cannot be subclassed.

2. **Final Method**: A method declared as `final` cannot be overridden by subclasses.
3. **Final Variable**: A variable declared as `final` cannot be reassigned after initialization.

The `final` keyword in Java is a powerful tool that helps control how classes, methods, and variables behave, ensuring that certain aspects of the code remain unchanged, improving safety, performance, and predictability.