MOD -02 -> OOPs

LUMINAR
TECHNOLAB

Final keyword

Final keyword

Final is a keyword in java which can be used with:

1. Instance variables
2. Static variables
3. Local variables
4. Method parameters
5. Methods
6. Classes
7. Object/array references

1. Final instance variables

1. An instance variable declared with the final keyword is called a final instance variable
2. When you declare an instance variable as final, you must initialize it either at the point of declaration or in the constructor of the class.
3. final variables are treated as constants in java they can’t reassign, and they are generally declared with static keyword

```java
//This program is used to show that the value of final variable can't be change.

package finalpack;

class Test {
    // final variable
    final int num = 100;

    // method for try to change the value of final variable.
    public void show() {
        // error because value of final variable can't be change.
        // num=200;
        System.out.println("Num = " + num);
    }
}

public class FinalVariable {

    public static void main(String args[]) {
        // creating object of Test Class
        Test obj = new Test(); // method call
        obj.show();
    }
}
```

```java
package finalpack;

class Example {
    final int instanceVariable;

    public Example(int value) {
        this.instanceVariable = value; // Must be initialized in the constructor
    }

    public void display() {
        System.out.println("The value of the final instance variable is: " + instanceVariable);
    }
}

public class FinalVariableConstructorAssign {

    public static void main(String[] args) {
        Example ex = new Example(20);
        ex.display();
    }
}
```

Why is the final keyword often used together with static in Java? Can you explain their combined usage with a practical example?

A final variable in Java means its value cannot be changed once assigned. A static variable means it belongs to the class rather than any specific object. They are often used together for constants. Ie. We use static final in Java to define class-level constants. final ensures the value is immutable, and static ensures the value is loaded only once per class — saving memory and promoting consistency across all instances. It also improves code readability and maintenance by clearly indicating that the value is a universal constant.

```
public class  Account{
    public static final int MIN_BALANCE= 1000;
}
```

- ✓ MIN_BALANCE cannot be changed (final)
- ✓ It’s accessible without creating an object (static), like: Account. MIN_BALANCE

3. Local final variables

A local final variable is a variable declared inside a method (or block) using the final keyword. Once assigned a value, it cannot be changed.

```
void show() {
    final int temp = 10;
    temp= 20; //  Compile-time error: cannot
assign a value to final variable temp
    System.out.println(temp);
}
```

```
{
    final int temp = 10;
    temp= 20; //  Compile-time error: cannot
assign a value to final variable temp
    System.out.println(temp);
}
```

Why Use Local Final Variables?

- Immutability – prevents accidental changes within the method.
- Required in anonymous classes or lambda expressions (Java 8+).
- Useful in inner classes.

4. Final Method Parameters

A final method parameter means the parameter cannot be reassigned within the method body.

Why use final with method parameters?

- Protects parameter from modification inside the method.
- Improves code safety and clarity.
- Relevant in multithreading, Useful in anonymous inner classes, where the parameter must be final or effectively final.(**Even if not explicitly marked final, a local variable that isn’t reassigned is considered "effectively final" in Java 8 and above.

```java
package finalpack;

class TestFinalParameter {
    public void showValue(final int num) {
        // num=num*5;
        System.out.println("Num=" + num);
    }
}

public class FinalParameter {
    public static void main(String[] args) {
        TestFinalParameter obj = new TestFinalParameter();
        obj.showValue(25);
    }
}
```

5. Final Methods

A method declared with final keyword is known as final method.

- Final method can be inherited.
- cannot be overridden by any subclass. This is used to:
  - Preserve method behavior (prevent alteration in child classes).
  - Ensure security or stability of critical logic.

```java
package finalpack;

/**
 * This program is used to show that final method can't be override.
 */

class Show {
    public final void show() {
        System.out.println("Hello world.");
    }
}

class Display extends Show {
    //error because final method can't be override.
    /*
     * public void show() {
        System.out.println("Hello");
    }
     */
}
```

```java
public class FinalMethod {

    public static void main(String args[]) {
        // creating object of Display class
        Display obj = new Display(); // method call
        obj.show();
    }

}
```

6. Final Class

A class declared with final keyword is known as final class. A final class is a class that cannot be extended (i.e., no subclass can inherit from it).

Why use a final class?

- Security: Prevent altering via subclassing.
- Immutability: Often used in immutable classes like String.
- Performance: JVM can make certain optimizations if it knows the class won’t be subclassed.
- Design Intent: Clearly communicates “this class is complete, do not extend it.”

```java
package finalpack;

final class Trial {
    public void show() {
        System.out.println("Hello world.");
    }
}

class Demo extends Trial {
    public void show() {
        System.out.println("hiii");
    }
}

public class FinalClass {
    public static void main(String[] args) {
        Demo obj=new Demo();
        obj.show();
    }
}
```

7. Final Object and Array References

When an object or array reference is declared final, it means:

The reference cannot point to another object,But the contents of the object or array can still change (unless they are also immutable e.g. String class).

```
final Student s = new Student(“Arun");
s.name = “Bini";          //  Allowed
// s = new Student("Neha"); //  Not allowed — cannot reassign the reference
```

```
final int[] numbers = {1, 2, 3};
numbers[0] = 99;          //  Allowed — modifying content
// numbers = new int[5];   //  Not allowed — cannot reassign the array reference
```

8. Blank final variable in java

A blank final variable is a final variable declared but not initialized at the time of declaration.

It must be initialized exactly once, either :

- In the constructor, or
- In an instance initializer block (rarely used).

```
class Student {
    final int rollNumber; // blank final

    Student(int rollNumber) {
        this.rollNumber = rollNumber; // must initialize here
    }
}
```

They are initialized at the time of object creation in constructor and can’t change after that.

In the Context of Multiple Objects:

Each object will have its own copy of the blank final variable.

```
Student s1 = new Student(101);
Student s2 = new Student(102);
```

This allows each object to have a unique final value that can't change after initialization.

Who Initializes It?

- You, the programmer, must initialize it — usually in the constructor.
- If you forget, the compiler throws an error.
- Each object gets its own unique blank final variable.
- You must initialize it in the constructor or instance block.
- It's readable(can use) wherever the variable is visible(depends on access and inheritance), but immutable(can't change) after initialization.

```java
package finalpack;

class TestTrial {
    final int num; // The blank final field num may not have been initialized

    TestTrial(int n) {
        num = n; //blank final variable which can only be initialized through constructor
    }

    public void show() {
        // num=23; //The final field TestTrial.num cannot be assigned
        System.out.println("Num=" + num);
    }

}

public class BlankFinalVariable {
    public static void main(String[] args) {
        TestTrial obj=new TestTrial(5);
        obj.show();
    }
}
```

What is an Instance Initializer Block?

An instance initializer block is a block of code in a class that runs every time an object is created, before the constructor.

```
{
   // This is an instance initializer block
   System.out.println("Object is being initialized.");
}
```

When is it executed?

- Every time a new object is created.
- It runs before the constructor code.
- It is executed after field initializers, but before the constructor body.

Why and When is it Used?

Mainly in rare cases like:

- Initializing blank final instance variables, especially when multiple constructors exist and you don’t want to repeat the same logic.(You have a blank final instance variable, and your class has multiple constructors. Since final means "must be initialized once," you’d have to repeat the same assignment in every constructor — which violates DRY (Don’t Repeat Yourself).)

```
class Book {
    final int pages;

    Book() {
        pages = 100;   // ✅ must initialize
    }

    Book(String title) {
        pages = 100;   // ✅ must repeat same logic
    }
}
```

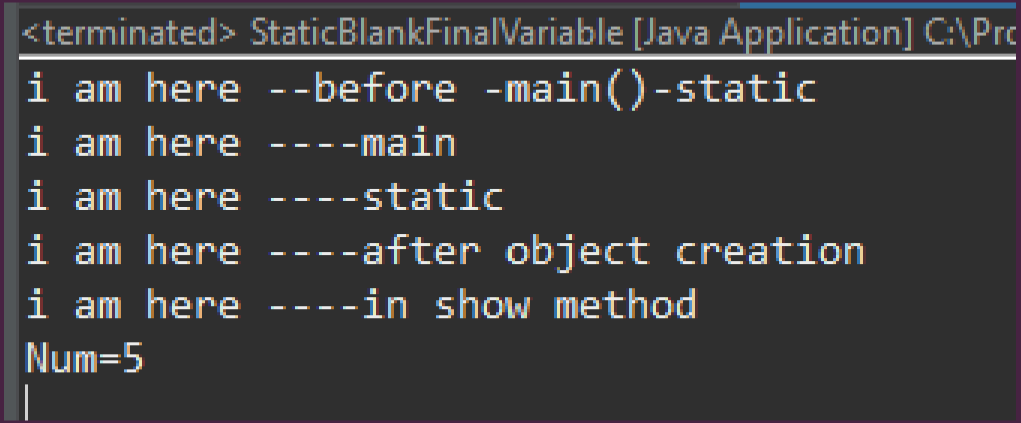
9. Static blank final variable in java

A static variable declared with final keyword but not initialized at declaration time is known as static blank final variable. It can be initialized in a static initializer block(static block) only.

```java
package finalpack;

class TestTrialDemo {
    static final int num; // The static blank final field num may not have been initialized

    /*
     * TestTrialDemo() { num = 5; //The final field TestTrialDemo.num cannot be
     * assigned }
     */
    static {
        num = 5; // static blank final variable which can only be initialized only in static
        // block
        // System.out.println("Num=" + num););  // you can print the value of variable
        // here. Not a good practice.
        // Coz static block/ static initializer block is just for initialization

    }
    public void show() {
        // num=23; //The static final field TestTrial.num cannot be assigned

        System.out.println("Num=" + num);
    }
}

public class StaticBlankFinalVariable {

    public static void main(String[] args) {
        TestTrialDemo obj = new TestTrialDemo();
        obj.show();
    }

}
```

Try static block usage

```java
package finalpack;

class TestTrialDemo {
    static final int num; // The static blank final field num may not have been initialized

    /*
     * TestTrialDemo() { num = 5; //The final field TestTrialDemo.num cannot be
     * assigned }
     */
    static {
        System.out.println("i am here ----static");
        num = 5; // static blank final variable which can only be initialized only in static
        // block
        // System.out.println("Num=" + num););  // you can print the value of variable
        // here. Not a good practice.
        // Coz static block/ static initializer block is just for initialization

    }
    public void show() {
        // num=23; //The static final field TestTrial.num cannot be assigned

        System.out.println("i am here ----in show method");
        System.out.println("Num=" + num);
    }
}

public class StaticBlankFinalVariable {
    static {
        System.out.println("i am here --before -main()-static");
    }

    public static void main(String[] args) {
        System.out.println("i am here ----main");
        TestTrialDemo obj = new TestTrialDemo();
        System.out.println("i am here ----after object creation");
        obj.show();
    }

}
```



```
<terminated> StaticBlankFinalVariable [Java Application] C:\Pro
i am here --before -main()-static
i am here ----main
i am here ----static
i am here ----after object creation
i am here ----in show method
Num=5
```

Advantages/Benefits of final keyword

1. Performance: JVM kept in cache if variables, methods and classes are declared final.
2. Immutable classes: With the help of final keyword we can made immutable classes.

Points to remember

1. A final method can be inherited but can't be override.

3. A final class can't be inherited.(stop inheritance)

4. Abstract class and constructor can't be final. (

Why Abstract Classes can't be final:

- An abstract class is meant to be inherited and implemented by subclasses.
- A final class cannot be extended.

Why Constructors can't be final

- Declaring a constructor final is meaningless (Not inherited — final has no effect)and not allowed.

Thank you ☺ Happy learning ☺