13-Final Variable, Method & Class

Author: Vincent Lau

Learning Objectives

- Use the final keyword for variables, methods & class
- Understand the difference between instance and static variables/methods
- Declare constant variables
- Understand static initialization blocks and initializer blocks
- Understand order of execution of initialization blocks and result of initialization

Final Variables

In Java, a `final` variable is a **variable that can only be assigned a value once**, and its value **cannot be changed** thereafter. Once a final variable is assigned a value, it becomes a constant and remains unchanged throughout the program's execution.

Key points about final variables:

- A final variable **must be initialized at the time of declaration or in the constructor of the class**.
- The value of a final variable cannot be modified once it is assigned.
- Final variables are typically written in **uppercase snake letters** to indicate that they are constants.
- Final variables are useful when you want to create variables with constant values that should not be modified, such as mathematical constants or configuration values.

Examples

```java
public class MyClass {
    private final int MAX_VALUE = 10;

    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.MAX_VALUE); // Output: 10

        // Compilation error: Cannot assign a new value to a final variable
        obj.MAX_VALUE = 20;
    }
}
```

In the example above, the `MAX_VALUE` variable is declared as final and assigned a value of 10. Since it is marked as final, any attempt to modify its value will result in a **compilation error**.

Static Final Variables

`Static final` variables, also known as **`constant`** variables, are final variables that are associated with the class itself rather than instances of the class. They have the same properties as final variables but are shared among all instances of the class.

Key points about static final variables:

- Static final variables are declared using the `static final` keywords.
- They are typically used to define **`class-level constants`** that should not be modified.
- Static final variables must be assigned a value either at the time of **declaration or within a static initializer block**.
- Final variables are typically written in **uppercase snake letters** to indicate that they are constants.

- They can be accessed using the class name followed by the variable name (e.g., `ClassName.VARIABLE_NAME`).

Example 1

```
1 public class MyClass {
2     private static final int MAX_VALUE = 10;
3 
4     public static void main(String[] args) {
5         System.out.println(MyClass.MAX_VALUE); // Output: 10
6     }
7 }
```

In the example above, the `MAX_VALUE` variable is declared as static final. It is associated with the class `MyClass` rather than instances of the class. Therefore, it can be accessed using the class name, as demonstrated in the `main` method.

Other Examples

```
1 static final double PI = 3.141592653589793;
2 static final int MAXIMUM_NUMBER_OF_ORDERS_PER_DAY = 1000;
```

To conclude, `static final` variables are commonly used for constants that are **shared among multiple instances of a class** or accessed at the class level.

Final Methods

When a method is declared as `final`, it means that the **method cannot be overridden by any subclass**. Once a method is marked as `final`, its implementation is considered final and cannot be changed or overridden in any subclass that extends the class containing the final method.

The main purpose of using `final` methods is to **prevent subclasses from modifying the behavior of the method, ensuring that the method's implementation remains consistent across all subclasses**.

Examples

```
1 class Superclass {
```

```
    public final void finalMethod() {
        // This method cannot be overridden in any subclass
        System.out.println("This is a final method.");
    }
}

class Subclass extends Superclass {
    // Compilation error: Cannot override the final method
    @Override
    public void finalMethod() {
        ...
    }
}
```

In the example above, the `finalMethod()` in the `Superclass` is marked as `final`. As a result, any attempt to override this method in a subclass, as demonstrated in the `Subclass`, will result in a **compilation error**.

Notes

- It's important to use the `final` keyword carefully, as it restricts the flexibility of your class design.
- Only mark a method as `final` if you have a strong reason to prevent its overriding, such as protecting critical behavior or ensuring consistency across subclasses.

Final Classes

In Java, a final class is a **class that cannot be subclassed or extended**.

1. **Subclassing Restriction**: A final class cannot be inherited or extended by any other class.
2. **Preventing Overrides**: Final classes are often used to prevent modifications to the class's behavior or to ensure that the implementation remains unchanged.
3. **Design Integrity**: By marking a class as final, you can ensure that the class's functionality, contracts, or business logic remain consistent and unaltered.
4. Efficiency: Final classes can offer performance benefits since they cannot be extended or overridden, allowing the compiler to make certain optimizations.

Examples

```
final class FinalClass {
    // Class implementation
```

```
}

// Compilation error: Cannot inherit from final FinalClass
class Subclass extends FinalClass {
    // Subclass implementation
}
```

In the example above, the class `FinalClass` is declared as `final`. This means that it cannot be subclassed or extended. Any attempt to inherit from it, as demonstrated by the `Subclass`, will result in a compilation error.

Notes

- It's important to note that final classes can still have non-final methods, fields, and constructors. The `final` keyword applies to the class itself, restricting subclassing, but it does not prevent the class from containing other non-final members.
- Final classes are commonly used in scenarios where you want to ensure that the **whole** class's behavior remains unchanged, preventing any modifications that could potentially compromise the integrity of the class's design or functionality.
- If you just want to make some methods (class behavior) unchangable, just make the method final, instead of final the whole class.
- Final classes are often used in Java standard library classes, such as **`String`** & **`Math`**, where it's crucial to maintain the consistency and integrity of the class's implementation.

Challenge: Final vs Static Final

- Think about the difference between `final` and `static final`

```
public class MyClass {
    final int instanceFinal; // if instanceFinal is static final, compile error. Why?

    public MyClass(int value) {
        instanceFinal = value;
    }

    public static void main(String[] args) {
        MyClass obj1 = new MyClass(10);
        MyClass obj2 = new MyClass(20);

        System.out.println(obj1.instanceFinal); // Output: 10
        System.out.println(obj2.instanceFinal); // Output: 20
```

```
14      }
15  }
```