`final` Modifier in Java

The `final` modifier in Java can be applied to classes, methods, and variables to indicate that they cannot be changed after their initial assignment. Here are real-world examples demonstrating the use of the `final` modifier for methods and classes.

Example 1: Final Modifier for Methods

Scenario: Secure Payment Gateway

In a secure payment gateway system, you might have a class `PaymentProcessor` that includes a critical method `validateTransaction()`. This method should not be overridden by any subclasses to ensure the validation process remains consistent and secure.

Code Example:

```
class PaymentProcessor {
    public final void validateTransaction(String transactionI
d) {
        // Secure validation logic
        System.out.println("Validating transaction with ID: "
+ transactionId);
    }

    public void processPayment(double amount) {
        System.out.println("Processing payment of $" + amoun
t);
    }
}

class CreditCardPayment extends PaymentProcessor {
    @Override
    public void processPayment(double amount) {
        System.out.println("Processing credit card payment of
```

```
$" + amount);
    }

    // The following method will cause a compile-time error b
ecause validateTransaction is final in the superclass
    // @Override
    // public void validateTransaction(String transactionId)
{
    //     System.out.println("Validating transaction in a di
fferent way.");
    // }
}

public class FinalMethodExample {
    public static void main(String[] args) {
        CreditCardPayment payment = new CreditCardPayment();
        payment.validateTransaction("TXN12345");
        payment.processPayment(150.0);
    }
}
```

Example 2: Final Modifier for Classes

Scenario: Immutable Class for Configuration

In a system where configuration settings should not be modified after creation, you can use a `final` class to create an immutable configuration class. This ensures that the configuration settings remain constant and cannot be altered by subclassing.

Code Example:

```
public final class Configuration {
    private final String url;
    private final int port;

    public Configuration(String url, int port) {
```

```
        this.url = url;
        this.port = port;
    }

    public String getUrl() {
        return url;
    }

    public int getPort() {
        return port;
    }
}

// The following class will cause a compile-time error becaus
e Configuration is final and cannot be subclassed
// class ExtendedConfiguration extends Configuration {
//     public ExtendedConfiguration(String url, int port) {
//         super(url, port);
//     }
// }

public class FinalClassExample {
    public static void main(String[] args) {
        Configuration config = new Configuration("<http://exa
mple.com>", 8080);
        System.out.println("URL: " + config.getUrl());
        System.out.println("Port: " + config.getPort());
    }
}
```

Explanation

1. **Final Methods**:
    - The `validateTransaction` method in the `PaymentProcessor` class is marked as `final` to prevent any subclass from overriding it. This ensures the

transaction validation logic remains unchanged and secure across all payment types.

2. **Final Classes**:
    - The `Configuration` class is marked as `final` to prevent subclassing. This is useful for creating immutable objects where the configuration settings should not be altered after creation. The class provides getter methods to access the configuration values but does not allow modification.

Summary

The `final` modifier is a powerful tool in Java for ensuring immutability and security in your code. By marking methods as `final`, you can prevent them from being overridden, preserving their intended behavior. By marking classes as `final`, you can prevent subclassing, ensuring that the class's design and behavior remain unchanged. These practices are particularly useful in scenarios where consistency and security are paramount.