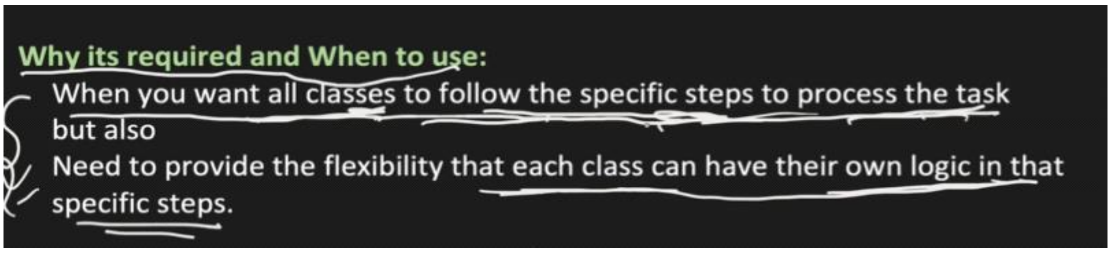
Template Pattern - Behavioural

11 January 2024 22:06



Why its required and When to use:

When you want all classes to follow the specific steps to process the task but also

Need to provide the flexibility that each class can have their own logic in that specific steps.

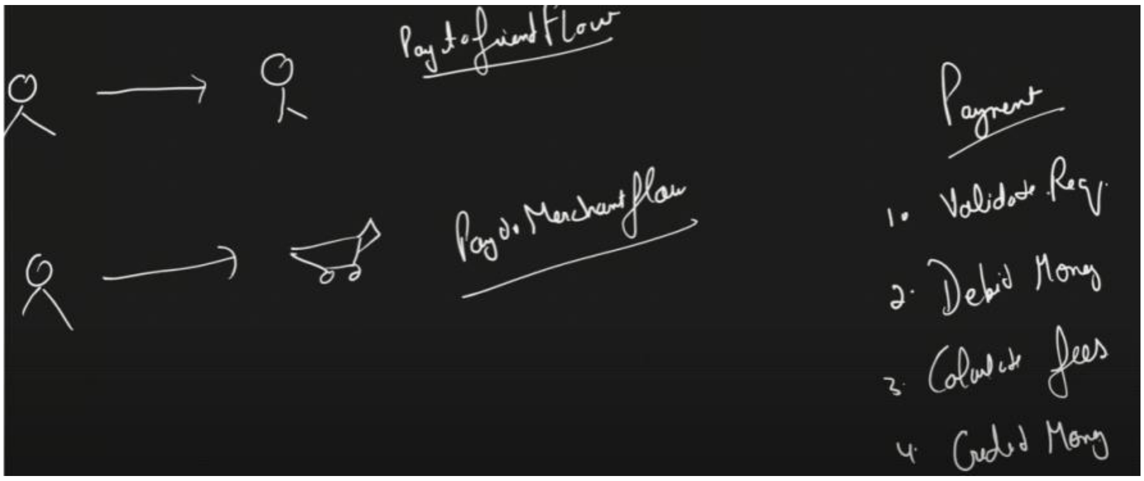
For Example:

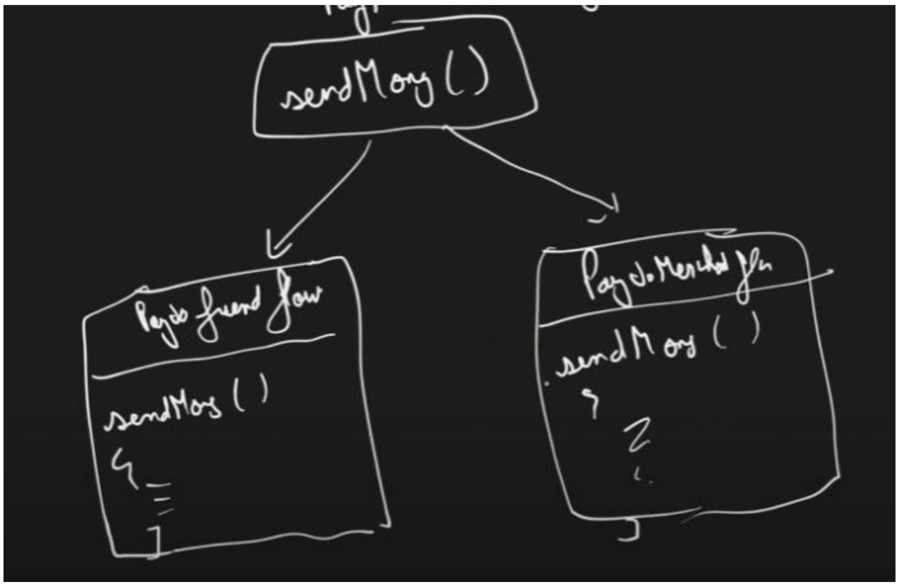
While doing payments there can be multiple scenario

1. Pay directly to someone account
2. Pay to merchant account

While doing above payment we want to enforce them a rule to be followed as below

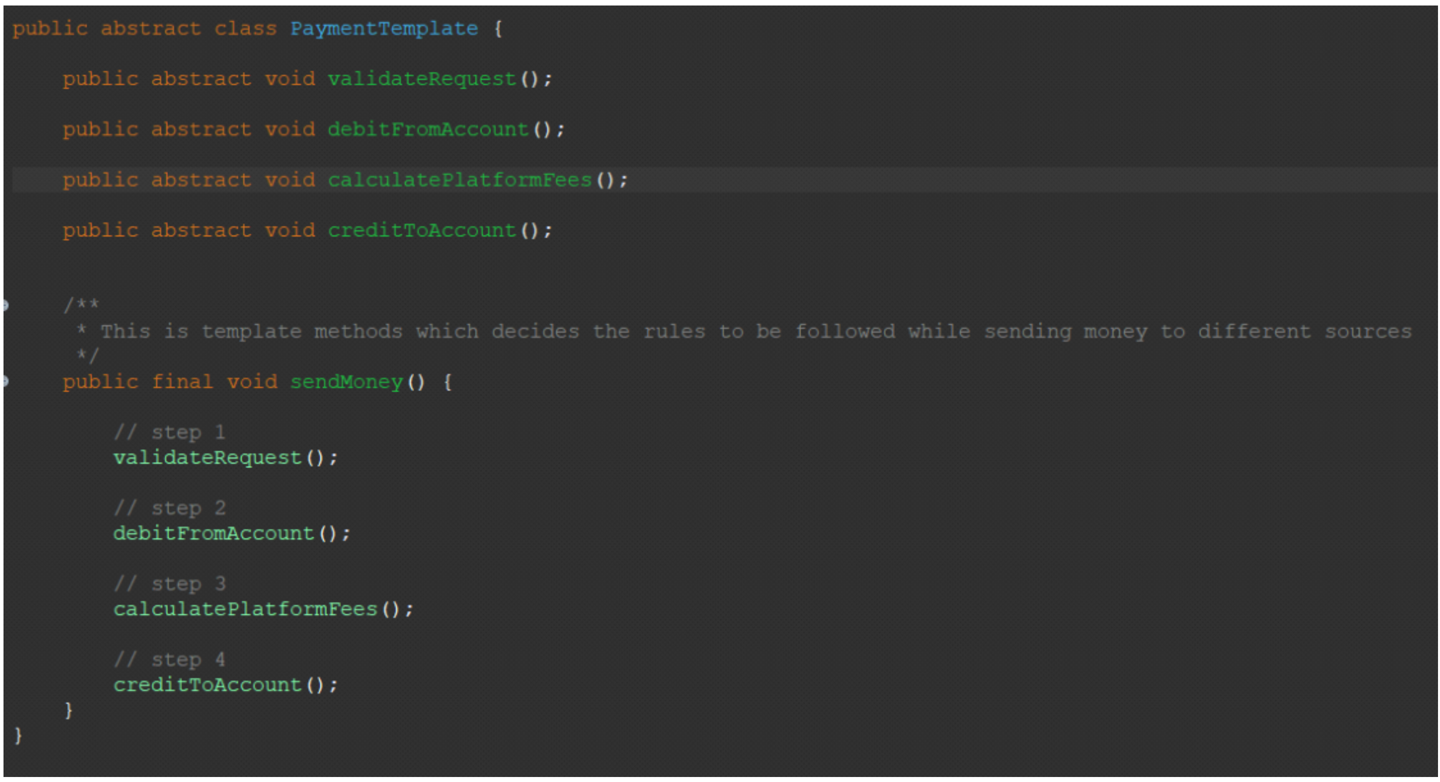
1. Validate request
2. Debit from account
3. Calculate platform fees
4. Credit to account





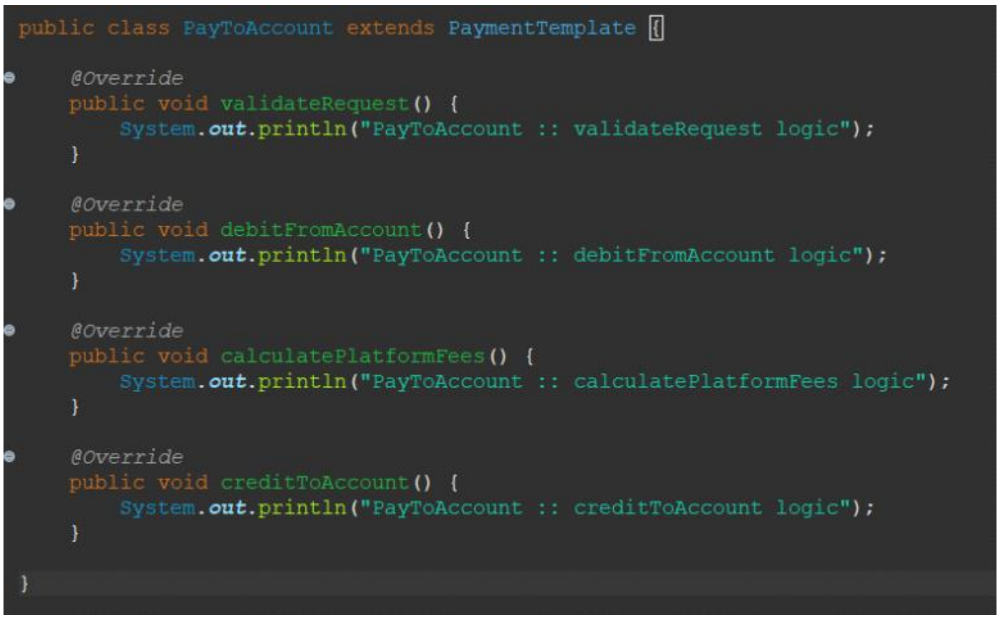
Implementation

Here we create a PaymentTemplate where we define a template method sendMoney()
which defines a set of rules to all the implentation class of PaymentTemplate.



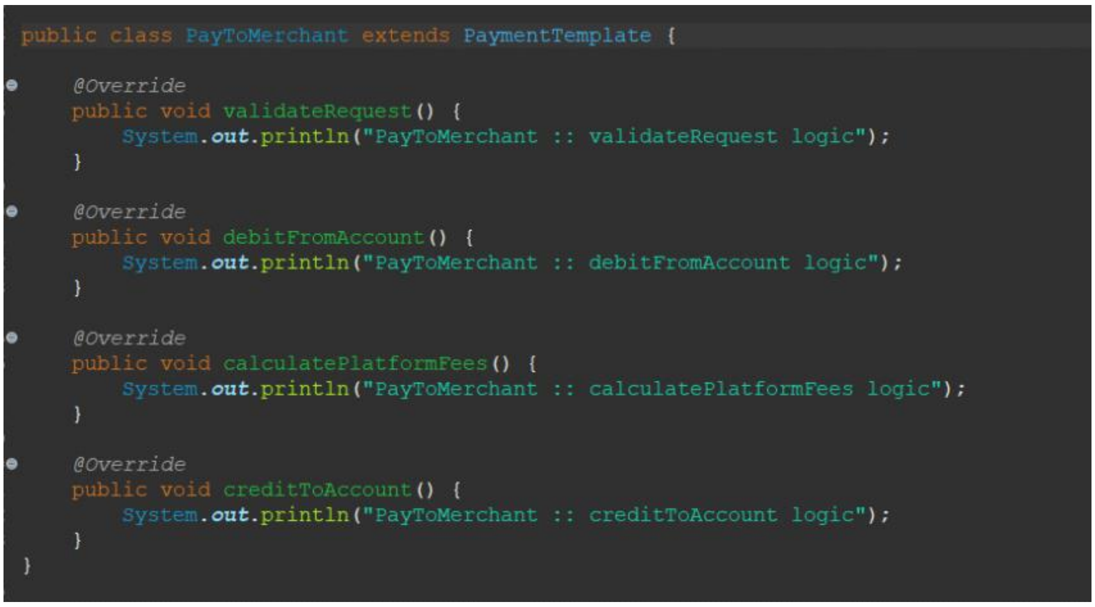
```java
public abstract class PaymentTemplate {

    public abstract void validateRequest();

    public abstract void debitFromAccount();

    public abstract void calculatePlatformFees();

    public abstract void creditToAccount();


    /**
     * This is template methods which decides the rules to be followed while sending money to different sources
     */
    public final void sendMoney() {

        // step 1
        validateRequest();

        // step 2
        debitFromAccount();

        // step 3
        calculatePlatformFees();

        // step 4
        creditToAccount();
    }
}
```

Pay To Account implementing PaymentTemplate and providing its own implementation for
all the rules to be followed for payment.



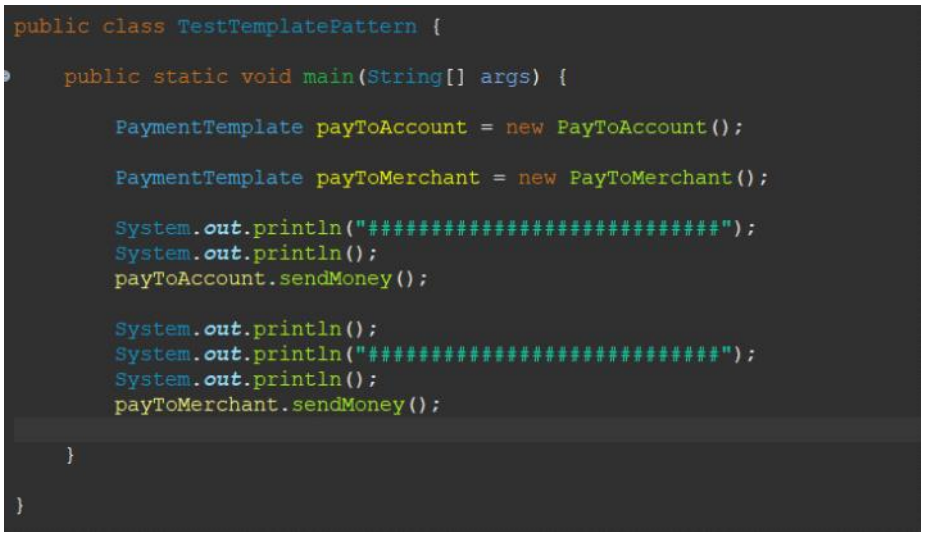
```java
public class PayToAccount extends PaymentTemplate {

    @Override
    public void validateRequest() {
        System.out.println("PayToAccount :: validateRequest logic");
    }

    @Override
    public void debitFromAccount() {
        System.out.println("PayToAccount :: debitFromAccount logic");
    }

    @Override
    public void calculatePlatformFees() {
        System.out.println("PayToAccount :: calculatePlatformFees logic");
    }

    @Override
    public void creditToAccount() {
        System.out.println("PayToAccount :: creditToAccount logic");
    }

}
```

Pay To Merchant implementing PaymentTemplate and providing its own implementation
for all the rules to be followed for payment.

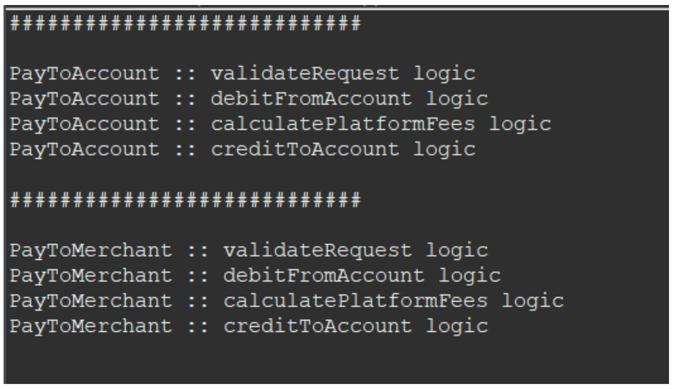


```java
public class PayToMerchant extends PaymentTemplate {

    @Override
    public void validateRequest() {
        System.out.println("PayToMerchant :: validateRequest logic");
    }

    @Override
    public void debitFromAccount() {
        System.out.println("PayToMerchant :: debitFromAccount logic");
    }

    @Override
    public void calculatePlatformFees() {
        System.out.println("PayToMerchant :: calculatePlatformFees logic");
    }

    @Override
    public void creditToAccount() {
        System.out.println("PayToMerchant :: creditToAccount logic");
    }
}
```

Testing Payment template where we check all the implemented class followed the rule of doing payment while all the rules are followed.



```java
public class TestTemplatePattern {

    public static void main(String[] args) {

        PaymentTemplate payToAccount = new PayToAccount();

        PaymentTemplate payToMerchant = new PayToMerchant();

        System.out.println("############################");
        System.out.println();
        payToAccount.sendMoney();

        System.out.println();
        System.out.println("############################");
        System.out.println();
        payToMerchant.sendMoney();

    }

}
```



```
############################

PayToAccount :: validateRequest logic
PayToAccount :: debitFromAccount logic
PayToAccount :: calculatePlatformFees logic
PayToAccount :: creditToAccount logic

############################

PayToMerchant :: validateRequest logic
PayToMerchant :: debitFromAccount logic
PayToMerchant :: calculatePlatformFees logic
PayToMerchant :: creditToAccount logic
```