Lesson 03 Demo 08

Creating a Payment Gateway with Abstraction and Inheritance

Objective: Using the concept of Abstraction and Inheritance in Java **Tools required:** Eclipse IDE **Prerequisites:** None

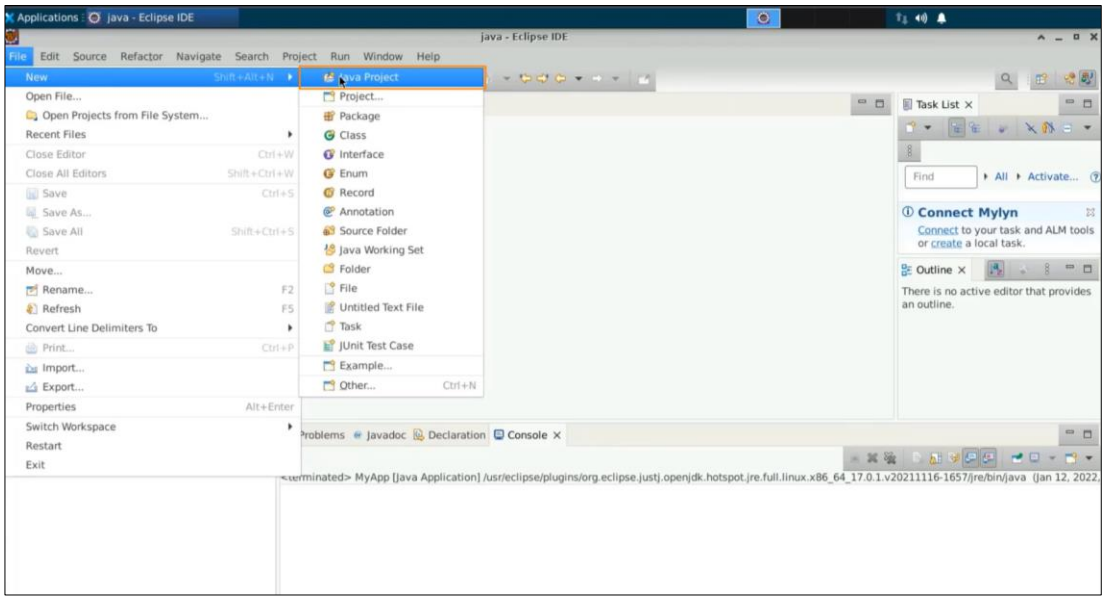
Steps to be followed:

1. Create a class called PaymentsApp, followed by selecting the main method
2. Create an abstract class PayTmPaymentgateway with a method bay and a private Boolean variable
3. Create a message for PayTmPaymentgateway, using a conditional loop
4. Integrate the payment gateway
5. Define rules and methods for success and failure, and create a constructor to execute the code
6. Understand the rule of inheritance and limitation of abstraction

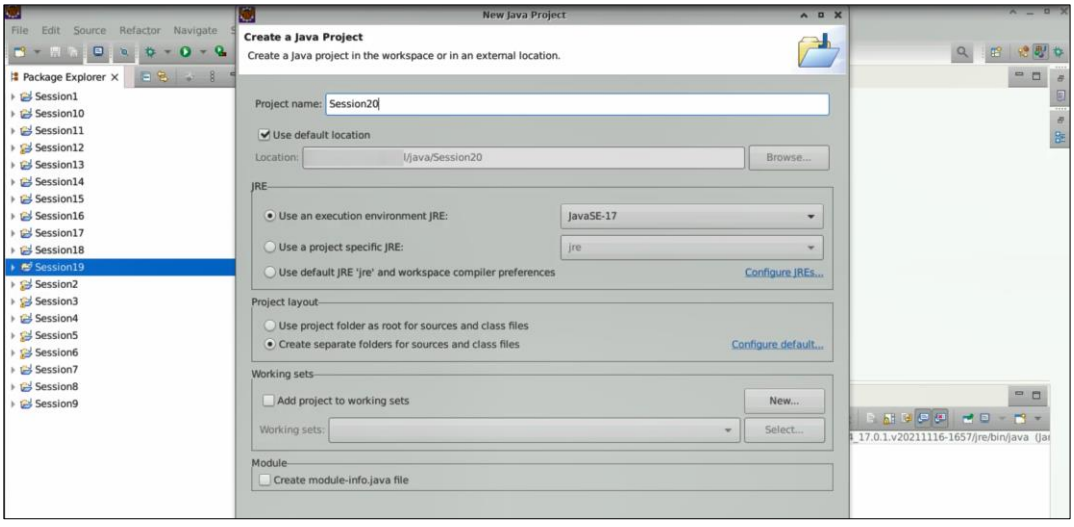
Step 1: Create a class called PaymentsApp, followed by selecting the main method

1.1 Open the **Eclipse IDE**

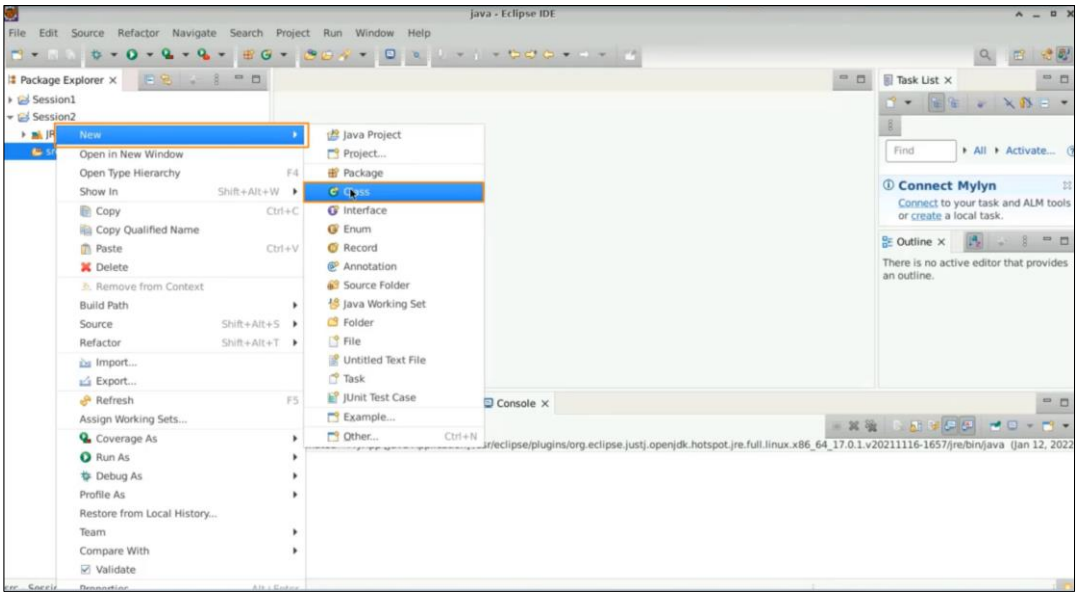
1.2. Select **File**, then **New,** and then **Java project**



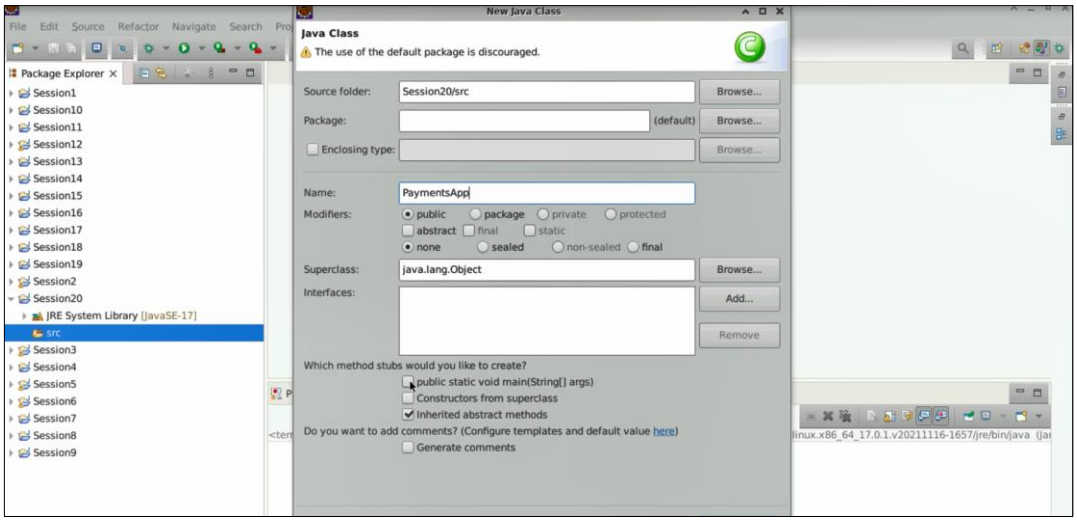
1.3 Name the project **"Session20",** uncheck **"Create a module info dot Java file"**, and press **Finish**



1.4 With a **Session20** on the src, do a right-click and create a **new class**

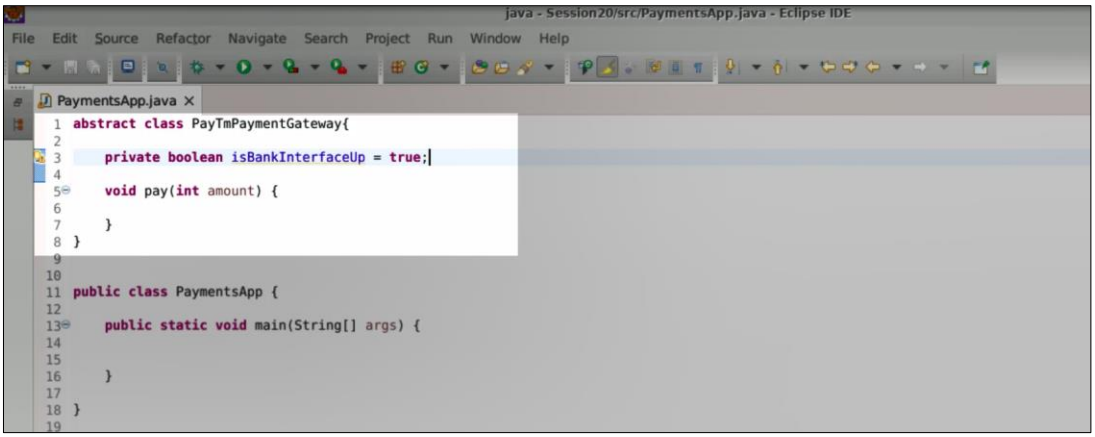


1.5 Name this class as a **PaymentsApp**, select the **main method,** and then select **finish**



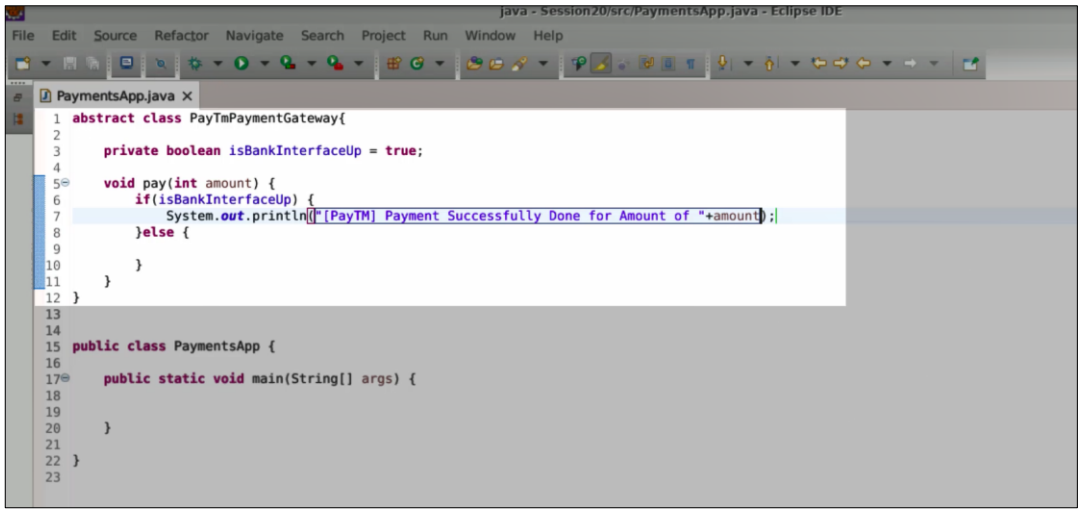
Step 2: Create an abstract class PayTmPaymentgateway with a method bay and a private Boolean variable

2.1 Create an abstract class called **PayTmPaymentGateway**. Define a method called pay. Inside the pay method, take one amount as input. To make this program a bit better, you can create a Boolean variable and mark it as private. This variable can be named **isBankInterfaceUp** and can be set to true

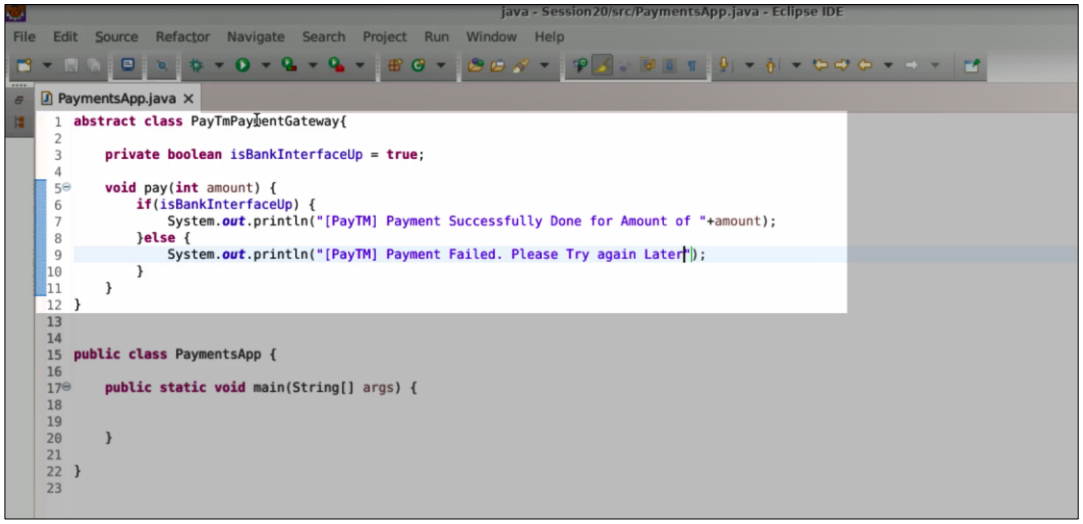


Step 3: Create a message for PayTmPaymentgateway, using a conditional loop

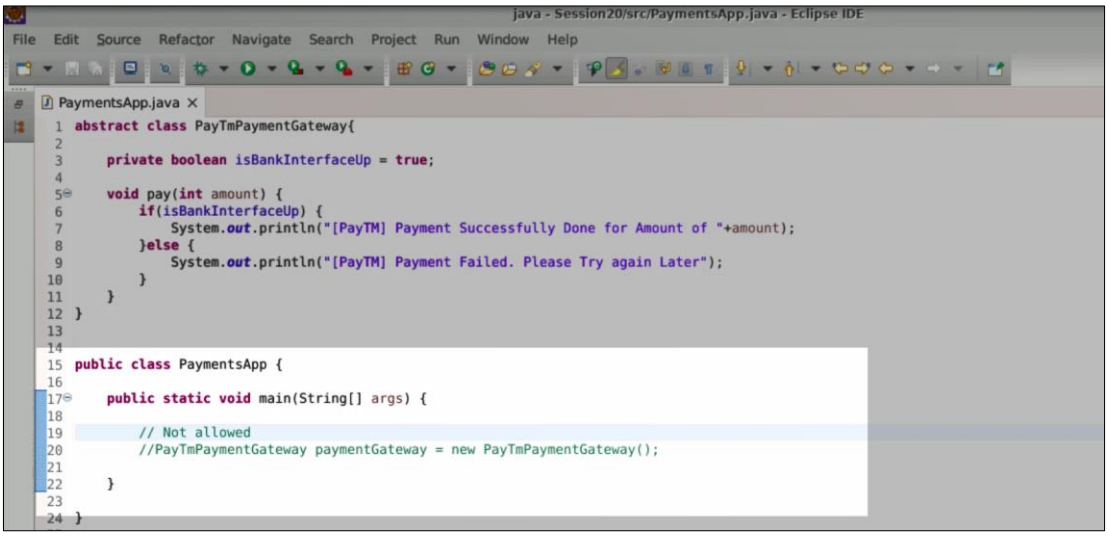
3.1 If the bank interface is up, the payment can be processed. Otherwise, the payment will fail. If the bank interface is up, that is one condition, and then you have an else condition. In the else condition, use **System.out.println** to display a message indicating the status in the **PayTmPaymentGateway**



3.2 In the other case, you are going to add a **System.out.println** statement from Paytm, which says that the payment failed. Please try again later

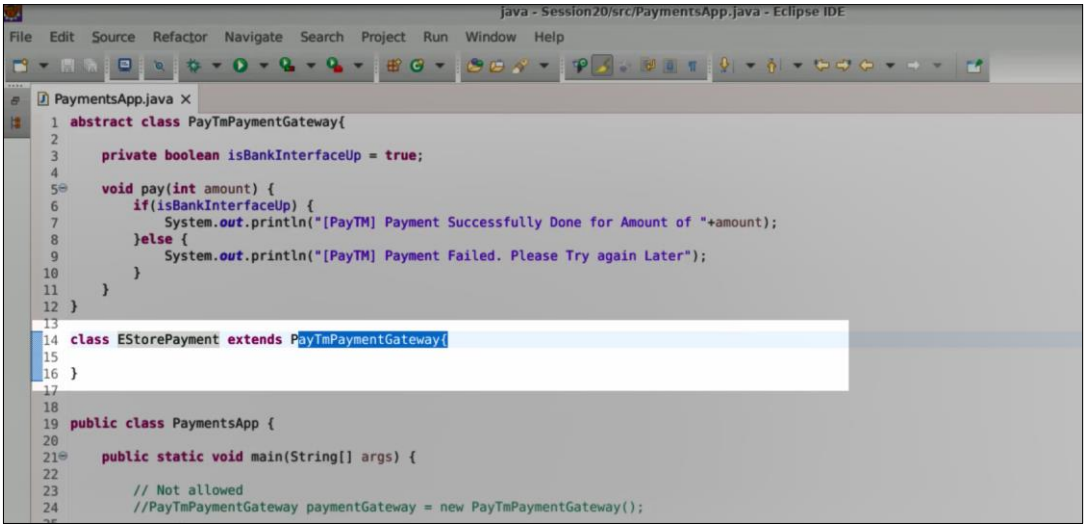


3.3 If you run the program here, you will see that a user object is constructed with serial number 1, and another user object is constructed with serial number 2. It is just a counter that you are using

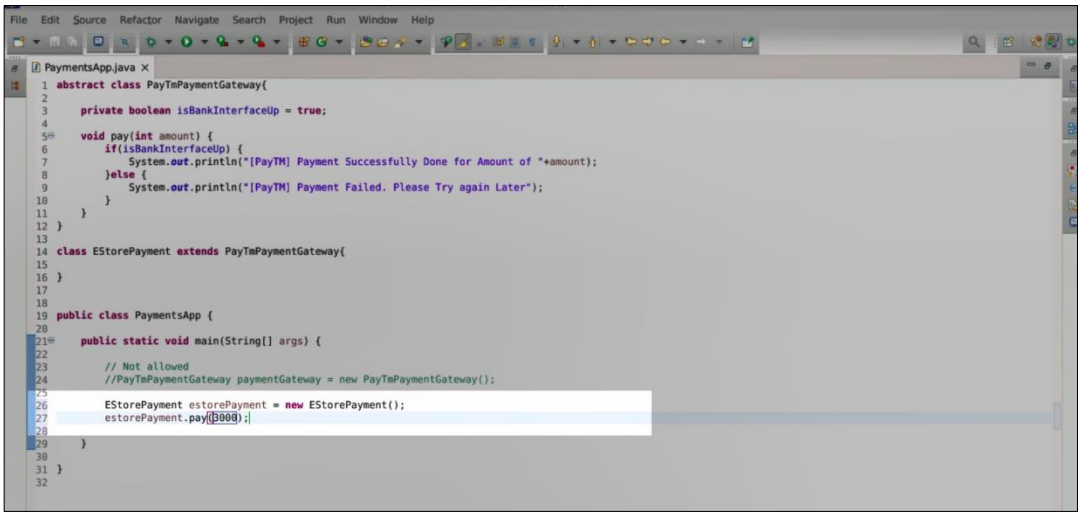


Step 4: Integrate the payment gateway

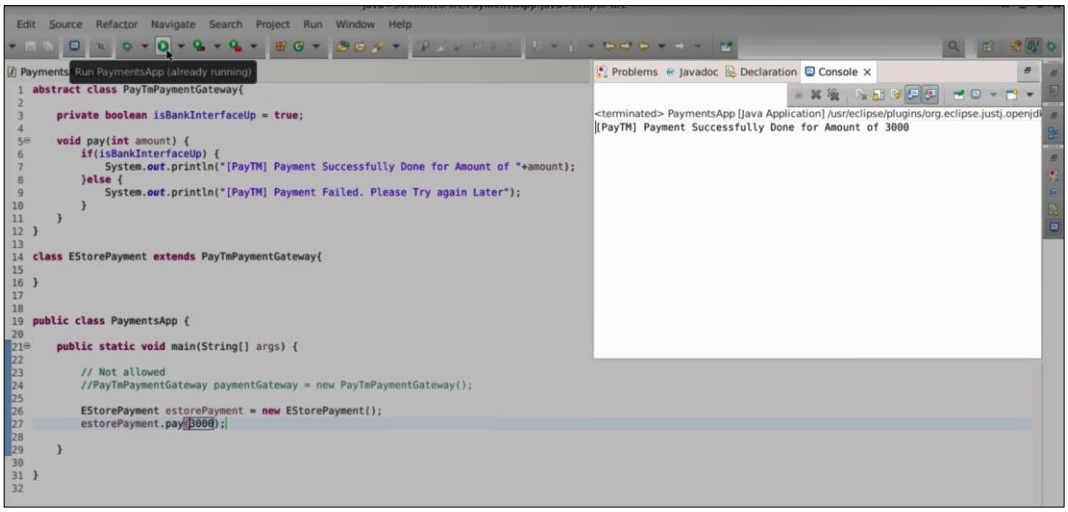
4.1 Now, in the next approach, consider you are creating an app, such as an e-store app. In your application, you would like to integrate a payment gateway. You will simply extend the payment gateway class



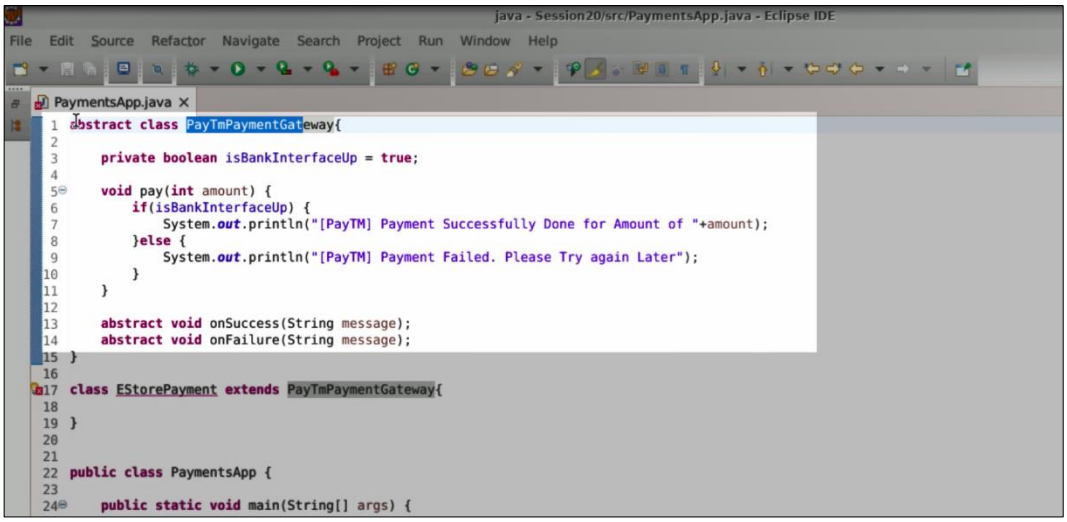
4.2 Now, whenever the user makes a transaction on the e-store, an object must be created for this e-store payment. Let us say a new **EStorePayment** object. With this e-store payment object, you will be able to directly execute a method called pay, for example, **pay(3000)**



4.3 When you run the code, it says that the **Payment Successfully Done for Amount of 3000**. But where is this message coming from? The message is coming from a class called **PaymentGateway**, not from your class called **EStorePayment**



4.4 You do not have any communication happening between the **PaymentGateway** and the **EStorePayment**. Once you inherit, you are done with it, and this message is handled entirely within the **PaymentGateway**, without coming back as an acknowledgment to the **EStorePayment**. What can be done? You should create two methods: one is an **abstract void onSuccess**, which takes a message, and another is an **abstract void onFailure**



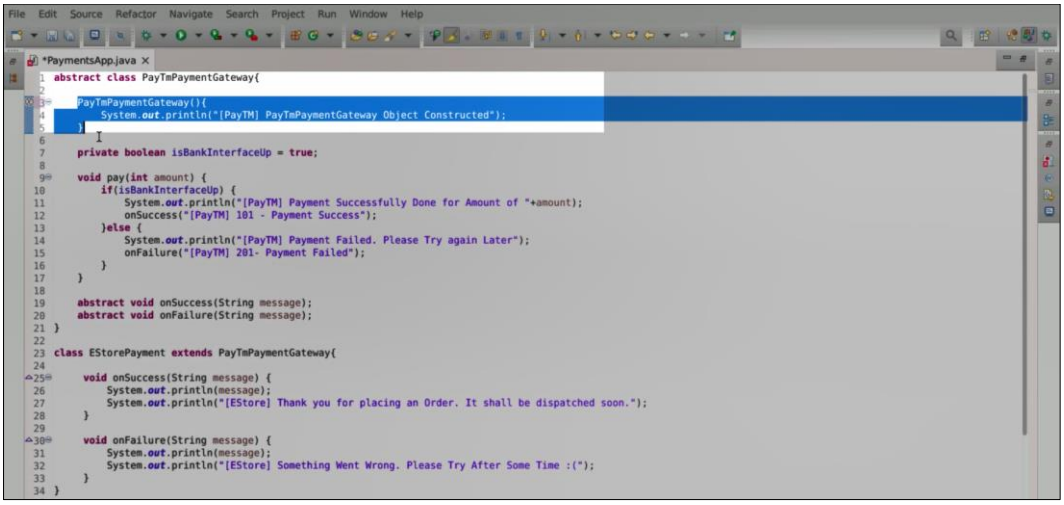
4.5 Let us now define the rules, the method for success, and another method for failure. Use **System.out.println** to print the message **"Thank you for placing an order. It shall be dispatched soon."** Similarly, add **System.out.println** in the failure method to notify the customer with **"Something went wrong. Please try again later**

```
    private boolean isBankInterfaceUp = true;

    void pay(int amount) {
        if(isBankInterfaceUp) {
            System.out.println("[PayTM] Payment Successfully Done for Amount of "+amount);
        }else {
            System.out.println("[PayTM] Payment Failed. Please Try again Later");
        }
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

class EStorePayment extends PayTmPaymentGateway{

    void onSuccess(String message) {
        System.out.println(message);
        System.out.println("[EStore] Thank you for placing an Order. It shall be dispatched soon.");
    }

    void onFailure(String message) {
        System.out.println(message);
        System.out.println("[EStore] Something Went Wrong. Please Try After Some Time :(");
    }
}


public class PaymentsApp {

    public static void main(String[] args) {

        // Not allowed
```

4.6 In case the bank interfaces are up, you will execute a method called **onSuccess**. The success message may have some code, let us say the code is **101** for **"payment success."** If there is a failure, you will execute a method called **onFailure**, with a message code like **201** for **"payment failed."** These two messages are coming from Paytm into your application

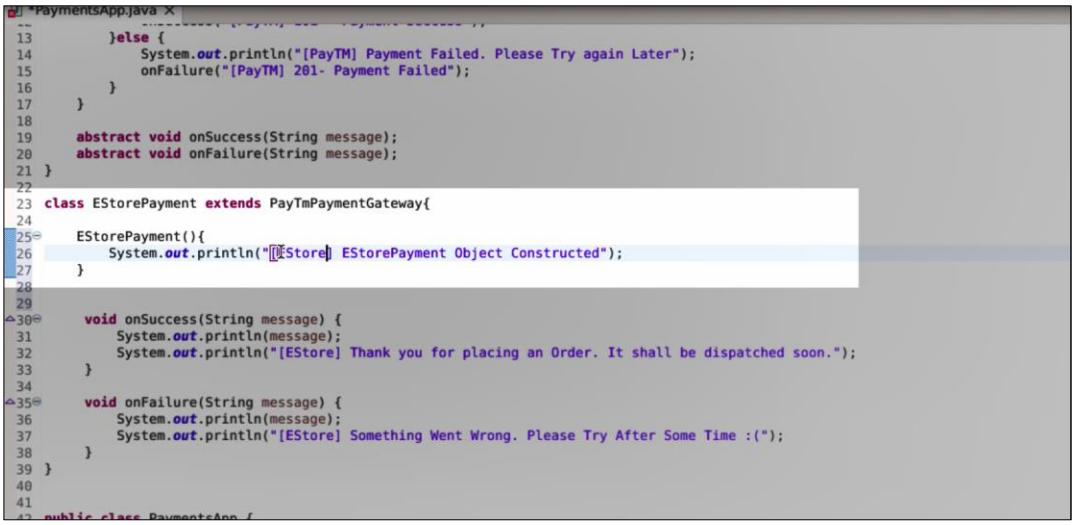
```
*PaymentsApp.java ×
abstract class PayTmPaymentGateway{

    private boolean isBankInterfaceUp = true;

    void pay(int amount) {
        if(isBankInterfaceUp) {
            System.out.println("[PayTM] Payment Successfully Done for Amount of "+amount);
            onSuccess("[PayTM] 101 - Payment Success");
        }else {
            System.out.println("[PayTM] Payment Failed. Please Try again Later");
            onFailure("[PayTM] 201- Payment Failed");
        }
    }

    abstract void onSuccess(String message);
    abstract void onFailure(String message);
}

class EStorePayment extends PayTmPaymentGateway{

    void onSuccess(String message) {
        System.out.println(message);
        System.out.println("[EStore] Thank you for placing an Order. It shall be dispatched soon.");
    }

    void onFailure(String message) {
        System.out.println(message);
        System.out.println("[EStore] Something Went Wrong. Please Try After Some Time :(");
    }
}
```

4.7 Now there may be some confusion about how these methods can be executed. Whenever you create the object of **EStorePayment** as a new **EStorePayment**, there is a parent object. You have two objects in memory. Let us define the constructor so that you understand it well. You are going to print a message like **"[PayTM] PayTmPaymentGateway object constructed"**

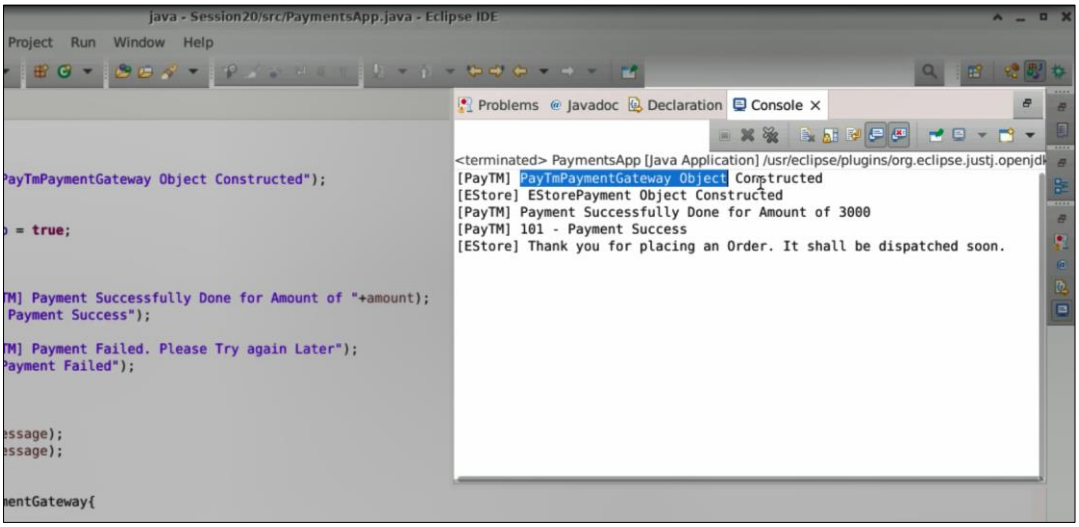


Step 5: Define rules and methods for success and failure, and create a constructor to execute the code

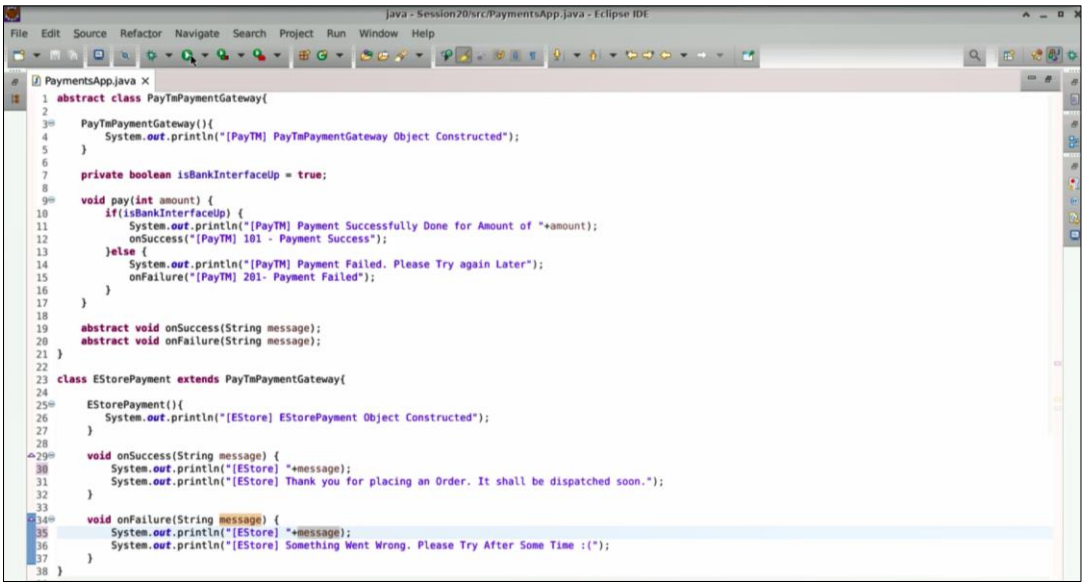
5.1 Similarly, you are going to create a constructor which **prints "The object is constructed for the eStore."** According to the rules of inheritance, whenever an **EStore** object is created, the **PaymentGateway** object should be created first, followed by the **EStorePayment** object



5.2 Now, when you run the code, you will see that before the **EStorePayment** object is constructed, the **PaymentGateway** object has been constructed. You send a request to the payment gateway, and it confirms that the payment was successful. This message is received by Paytm into the eStore

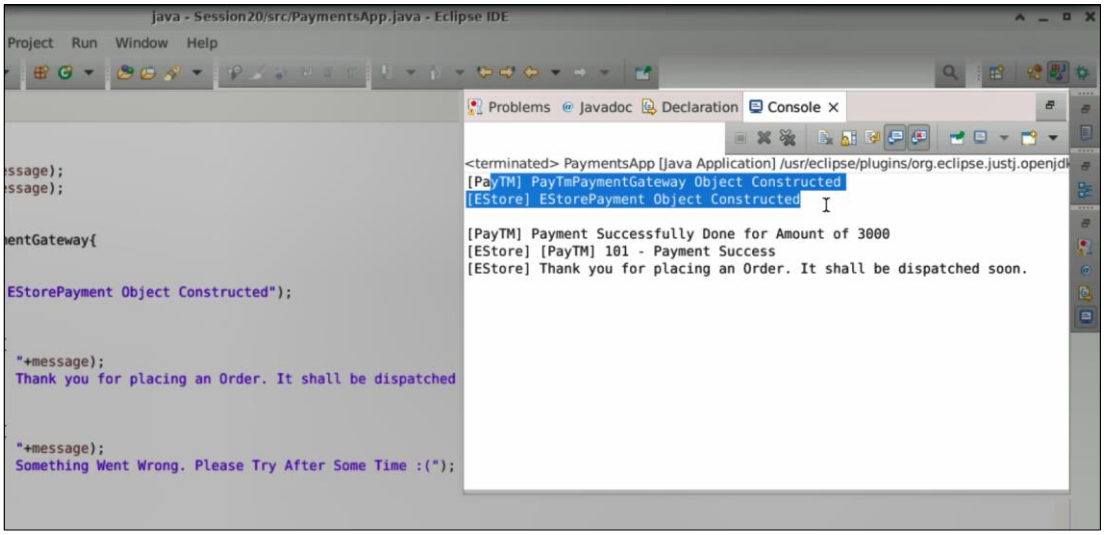


5.3 You can even come here and mark this message, which was received in the eStore from the payment gateway. Add an empty print line for better readability. These are the objects being constructed

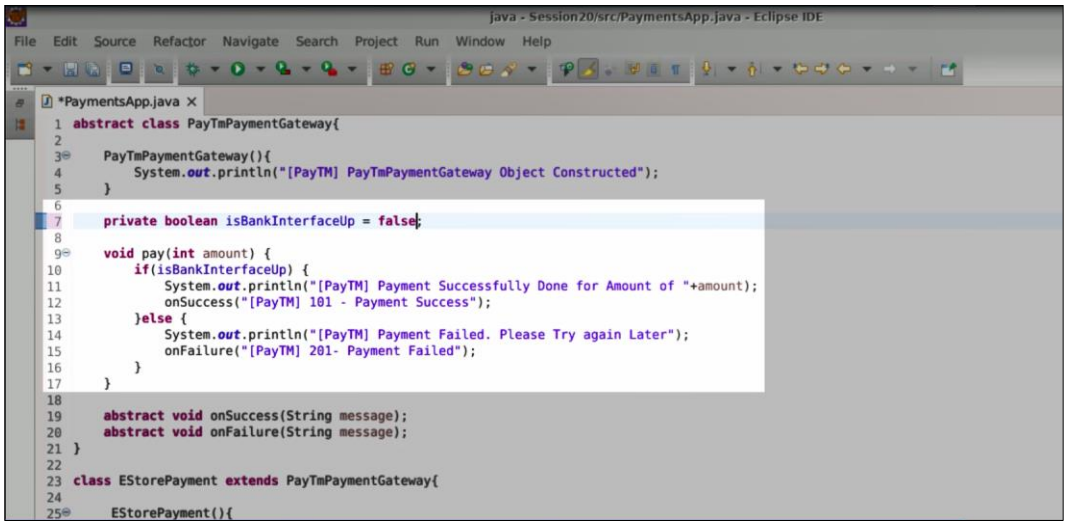


Step 6: Understand the rule of inheritance and limitation of abstraction

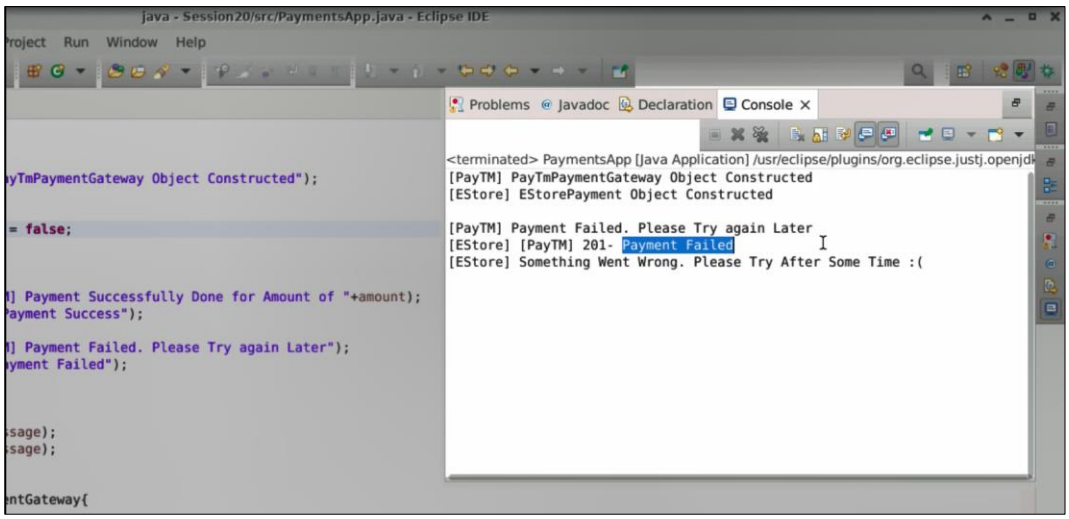
6.1 The rule of inheritance: The class is an abstract class, so whenever you perform a transaction on Paytm, you receive a message from Paytm. This message can be processed and sent to your customer with the eStore, saying **"Thank you for placing an order. It shall be dispatched soon"**



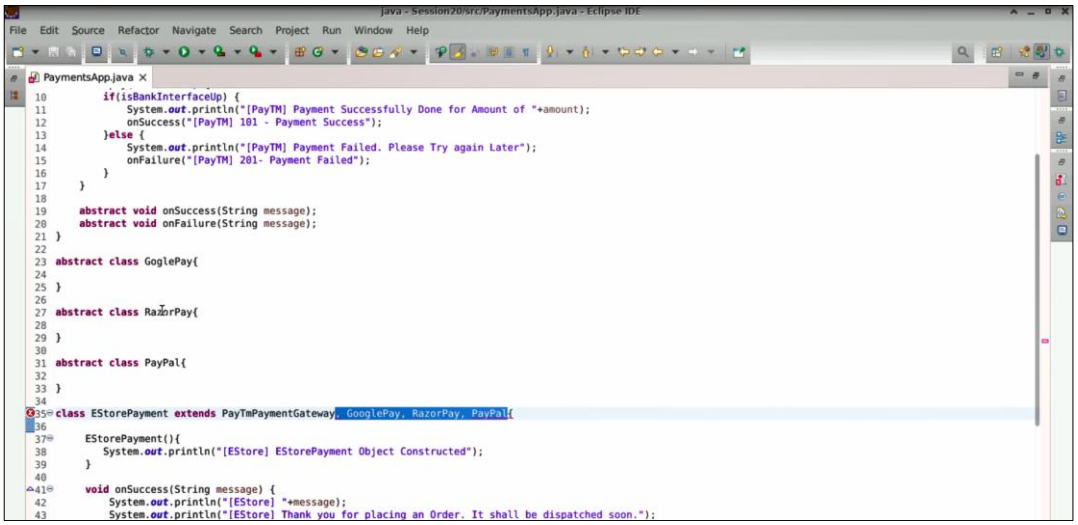
6.2 Let us consider the other scenario. If the bank interface is down, you will set the variable **isBankInterfaceUp** to **false**



6.3 When you run the program now, you will see that the payment has failed in Paytm. In the eStore, you receive a message with an error code, perhaps indicating that the Paytm payment failed. The eStore then informs the customer with a message saying, **"Something went wrong. Please try again later."** This is another simple illustration of how you can implement an abstract class and work with an abstraction



6.4 What is the problem? The problem arises if you have another class called Google Pay, or perhaps another class called Razorpay, and another class called PayPal. You cannot implement multiple inheritance here, so this is a limitation. You cannot say extends GooglePay, Razorpay, or PayPal. This is not allowed. This is a major limitation when it comes to abstraction, as you can inherit from only one class



By following the above steps, you have successfully created a Payment Gateway using abstraction and inheritance.