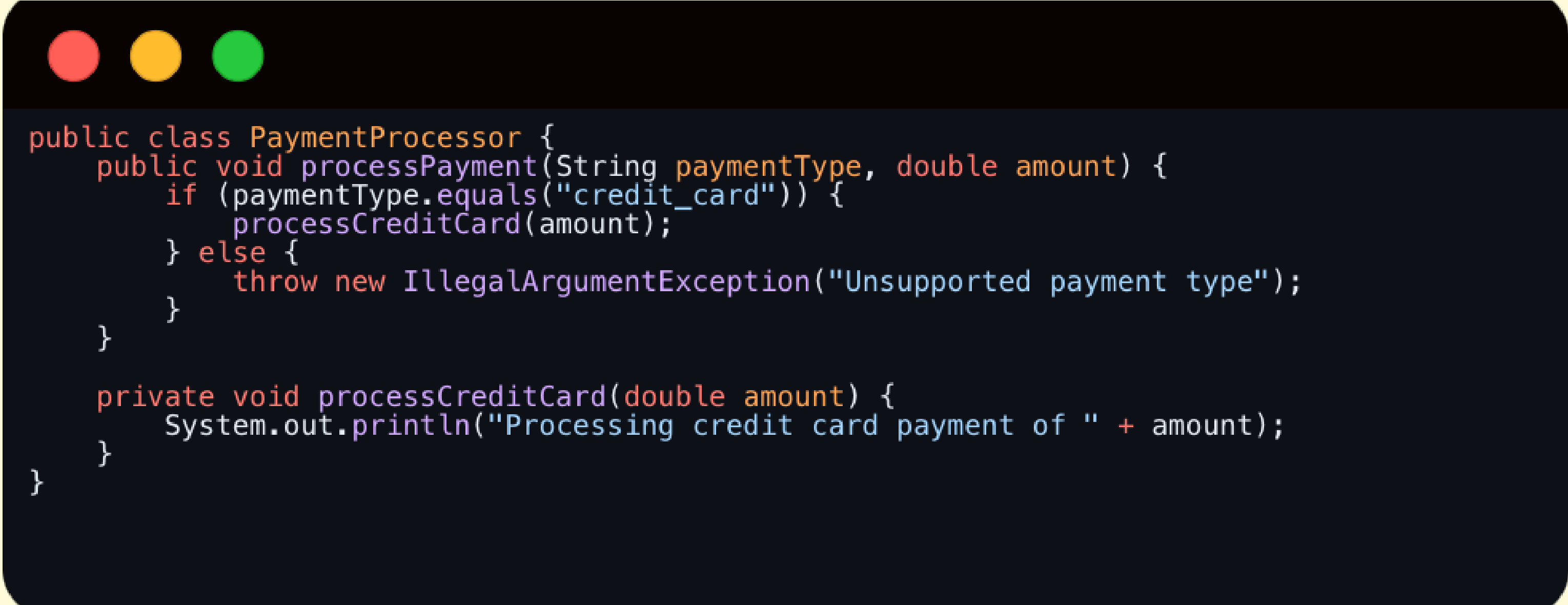


OPEN-CLOSE  
PRINCIPLE

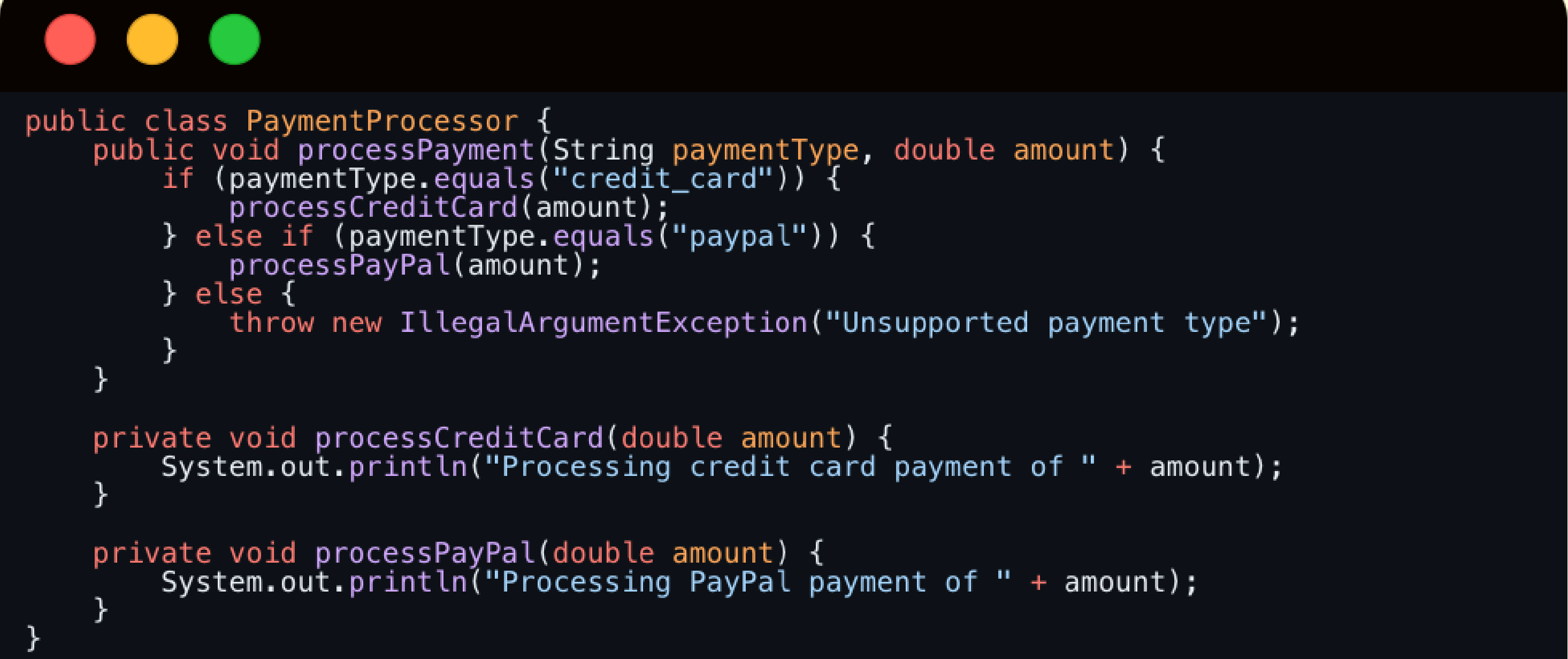
Start

Scenario: A payment processing system initially supporting only credit card payments.



```
public class PaymentProcessor {  
    public void processPayment(String paymentType, double amount) {  
        if (paymentType.equals("credit_card")) {  
            processCreditCard(amount);  
        } else {  
            throw new IllegalArgumentException("Unsupported payment type");  
        }  
    }  
  
    private void processCreditCard(double amount) {  
        System.out.println("Processing credit card payment of " + amount);  
    }  
}
```

Problem: To add a new payment type, such as PayPal, the PaymentProcessor class must be modified.



```
public class PaymentProcessor {  
    public void processPayment(String paymentType, double amount) {  
        if (paymentType.equals("credit_card")) {  
            processCreditCard(amount);  
        } else if (paymentType.equals("paypal")) {  
            processPayPal(amount);  
        } else {  
            throw new IllegalArgumentException("Unsupported payment type");  
        }  
    }  
  
    private void processCreditCard(double amount) {  
        System.out.println("Processing credit card payment of " + amount);  
    }  
  
    private void processPayPal(double amount) {  
        System.out.println("Processing PayPal payment of " + amount);  
    }  
}
```

This approach violates the Open/Closed Principle because modifying the class is necessary for adding new payment types.

## Solution - Following the OCP: Implementing the OCP using an abstract class or interface for payment processing.

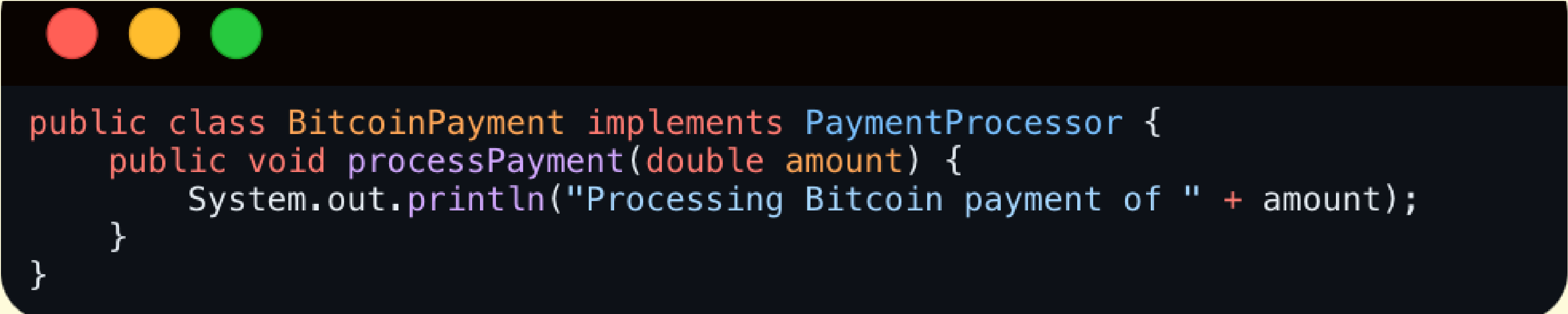
```
// Define an interface for payment processing
public interface PaymentProcessor {
    void processPayment(double amount);
}

// Implement the interface for credit card payments
public class CreditCardPayment implements PaymentProcessor {
    public void processPayment(double amount) {
        System.out.println("Processing credit card payment of " + amount);
    }
}

// Implement the interface for PayPal payments
public class PayPalPayment implements PaymentProcessor {
    public void processPayment(double amount) {
        System.out.println("Processing PayPal payment of " + amount);
    }
}

// A factory class to get the appropriate payment processor
public class PaymentProcessorFactory {
    public static PaymentProcessor getProcessor(String paymentType) {
        switch (paymentType) {
            case "credit_card":
                return new CreditCardPayment();
            case "paypal":
                return new PayPalPayment();
            default:
                throw new IllegalArgumentException("Unsupported payment type");
        }
    }
}
```

Solution: Adding a new payment type, such as Bitcoin, involves creating a new class implementing the `PaymentProcessor` interface without changing the existing code:



```
public class BitcoinPayment implements PaymentProcessor {  
    public void processPayment(double amount) {  
        System.out.println("Processing Bitcoin payment of " + amount);  
    }  
}
```

This design adheres to the Open/Closed Principle by allowing the system to be extended with new payment types without modifying existing code, ensuring stability and maintainability.