Assessment on Interface:

Assessment on Interface Scenario: You're designing a payment processing system for an online store. The system needs to handle various payment methods like credit cards, debit cards, and cash on delivery (COD). Task: 1.Interface Payment: Define an interface named Payment with a single abstract method processPayment(double amount). This method should take the amount to be paid as a double and return nothing (void). 2.Subclasses: CreditCard: Create a class CreditCard that implements the Payment interface. Include properties like card number, expiration date, and CVV. Implement the processPayment(double amount) method to simulate a credit card transaction (you can simply print a message indicating credit card processing). DebitCard: Create a class DebitCard similar to CreditCard but with its own implementation of processPayment(double amount) to simulate a debit card transaction. COD: Create a class COD that implements the Payment interface. This class doesn't need any additional properties. Implement the processPayment(double amount) method to indicate cash on delivery processing (print a message). 3.Testing: Create a PaymentProcessor class with a method makePayment(Payment payment, double amount). This method should take a Payment object (polymorphic reference) and the amount to be paid. Use method overloading to create two versions of makePayment: One that takes a Payment object and an amount (as described above). Another that takes just the credit card details (card number, expiration date, CVV) and the amount. This overloaded method should create a new CreditCard object and then call the makePayment(Payment payment, double amount) method with the newly created CreditCard object. Testing and Usage: Create instances of CreditCard, DebitCard, and COD. Use the PaymentProcessor class to make payments using these different methods (both by directly passing the Payment object and using credit card details). Optional Extensions: Add error handling to the processPayment methods to simulate potential payment failures. Introduce a new interface, Refundable, to handle refunds for specific payment methods (e.g., credit cards). Enhance the PaymentProcessor class to manage different currencies and exchange rates.

Sure! Below is the complete code for the payment processing system including the `PaymentProcessor` class and the `Main` method. This code demonstrates handling various payment methods like credit cards, debit cards, and cash on delivery (COD).

1. Define the `Payment` Interface

```
interface Payment {
    void processPayment(double amount);
```

}

2. Create Subclasses

`CreditCard` Class

```
class CreditCard implements Payment {
    private String cardNumber;
    private String expirationDate;
    private String CVV;

    public CreditCard(String cardNumber, String expirationDate, String CVV) {
        this.cardNumber = cardNumber;
        this.expirationDate = expirationDate;
        this.CVV = CVV;
    }

    @Override
    public void processPayment(double amount) {
        System.out.println("Processing credit card payment of $" + amount);
        // Simulate credit card processing logic here
    }
}
```

`DebitCard` Class

```
class DebitCard implements Payment {
    private String cardNumber;
    private String expirationDate;
    private String CVV;
```

```
    public DebitCard(String cardNumber, String expirationDate, String CVV) {
        this.cardNumber = cardNumber;
        this.expirationDate = expirationDate;
        this.CVV = CVV;
    }

    @Override
    public void processPayment(double amount) {
        System.out.println("Processing debit card payment of $" + amount);
        // Simulate debit card processing logic here
    }
}
```

`COD` Class

```
class COD implements Payment {
    @Override
    public void processPayment(double amount) {
        System.out.println("Processing cash on delivery payment of $" + amount);
        // Simulate COD processing logic here
    }
}
```

3. `PaymentProcessor` Class

```
class PaymentProcessor {
    public void makePayment(Payment payment, double amount) {
        payment.processPayment(amount);
```

```
    }

    public void makePayment(String cardNumber, String expirationDate, String CVV, double amount) {
        CreditCard creditCard = new CreditCard(cardNumber, expirationDate, CVV);
        makePayment(creditCard, amount);
    }
}
```

4. `Main` Class

```
public class Main {
    public static void main(String[] args) {
        // Create instances of payment methods
        CreditCard creditCard = new CreditCard("1234567890123456", "12/23", "123");
        DebitCard debitCard = new DebitCard("6543210987654321", "11/24", "321");
        COD cod = new COD();

        // Create an instance of PaymentProcessor
        PaymentProcessor paymentProcessor = new PaymentProcessor();

        // Make payments using different methods
        paymentProcessor.makePayment(creditCard, 100.0);
        paymentProcessor.makePayment(debitCard, 200.0);
        paymentProcessor.makePayment(cod, 50.0);

        // Make a payment using credit card details directly
```

```
        paymentProcessor.makePayment("1111222233334444", "10/25", "456", 300.0);
    }
}
```

Output

When you run the `Main` class, the output will be:

```
Processing credit card payment of $100.0
Processing debit card payment of $200.0
Processing cash on delivery payment of $50.0
Processing credit card payment of $300.0
```

This output shows that the system processes different types of payments and correctly handles the method overloading in the `PaymentProcessor` class.