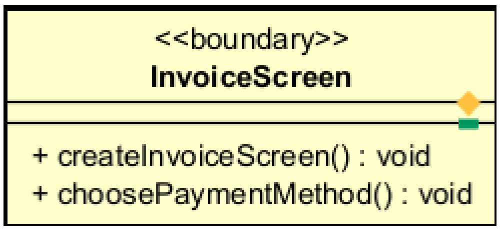
Design for class “InvoiceScreen”


<<boundary>>
InvoiceScreen
+ createInvoiceScreen() : void
+ choosePaymentMethod() : void

	Name	Return Type	Description
1	createInvoiceScreen()		Create invoice for customer
2	ChoosePaymentMethod()		Choose way to pay order

Table 2. Example of operation design

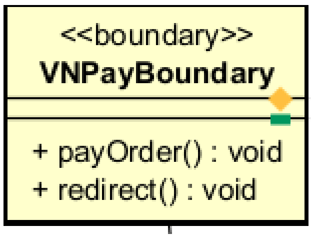
Method

```
 public static void createInvoiceScreen(Customer customer, List<Product> products,
double taxRate) {
        double subtotal = 0;
        System.out.println("INVOICE");
        System.out.println("Customer Information:");
        System.out.println(customer);
        System.out.println("\n");
        System.out.println("Purchased Products:");
        for (Product product : products) {
            System.out.println(product);
            subtotal += product.getTotalPrice();
        }
        double taxAmount = subtotal * taxRate;
        double totalAmount = subtotal + taxAmount;
        System.out.println("\nSubtotal: $" + String.format("%.2f", subtotal));
        System.out.println("Tax (" + (taxRate * 100) + "%): $" + String.format("%.2f",
taxAmount));
        System.out.println("Total Amount: $" + String.format("%.2f", totalAmount));
    }
```

```
 public static void choosePaymentMethod() {
        Scanner scanner = new Scanner(System.in);

        // Display available payment methods
        System.out.println("Please choose a payment method:");
        System.out.println("1. Credit Card");
        System.out.println("2. PayPal");
        System.out.println("3. Cash on Delivery");
        System.out.print("Enter the number corresponding to your choice: ");

        // Read the user's choice
        int choice = scanner.nextInt();

        // Handle the payment method choice
```

```
      switch (choice) {
         case 1:
            // If user chooses Credit Card
            handleCreditCardPayment();
            break;
         case 2:
            // If user chooses PayPal
            handlePaypalPayment();
            break;
         case 3:
            // If user chooses Cash on Delivery
            handleCashOnDelivery();
            break;
         default:
            System.out.println("Invalid choice. Please choose a valid payment
method.");
            choosePaymentMethod();  // Recursively prompt the user until a valid choice
is made
            break;
      }
```

Design for class "VNPayBoundary"



	Name	Return Type	Description
1	payOrder()		After the transaction request is sent, it will wait for the payment confirmation from the VNPay system.
2	redirect()		If the payment is successful, it may redirect the user to a confirmation page or display a success message. If the payment fails, it will direct the user to an error page or allow them to retry the payment.

Table 2. Example of operation design

Exception

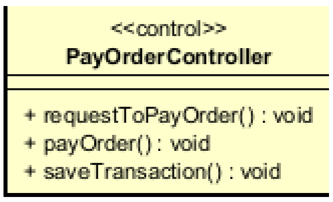
Name	Description
InvalidTransaction	Raise when the VNPay do not receive transaction

Method

```
public static void payOrder(Order order, String paymentMethod) {
      // 1. Validate Order
      if (order == null || order.getTotalAmount() <= 0) {
         System.out.println("Invalid order. Payment could not be processed.");
         return;
      }

      // 2. Prepare the payment data to be sent to VNPay
      Map<String, String> paymentData = new HashMap<>();
      paymentData.put("order_id", order.getOrderId());
      paymentData.put("total_amount", String.valueOf(order.getTotalAmount()));
      paymentData.put("payment_method", paymentMethod); // e.g., Credit Card,
PayPal
      paymentData.put("customer_email", order.getCustomerEmail());
      paymentData.put("currency", "VND"); // Assuming VND as currency

      // 3. Send the data to VNPay for payment processing (simulated)
      boolean paymentSuccessful = sendToVNPay(paymentData);

      // 4. Handle VNPay Response
      if (paymentSuccessful) {
         System.out.println("Payment request sent successfully. Redirecting to
payment gateway...");
         // Redirect user to VNPay payment page
         redirect(order);
      } else {
         System.out.println("Payment failed. Please try again.");
      }
   }
```

```
public void redirect(HttpServletResponse response, String paymentUrl) {
      try {
         // Redirect the user to VNPay's payment page
         response.sendRedirect(paymentUrl);
      } catch (IOException e) {
         e.printStackTrace();
         System.out.println("There are some errors.");
      }
```

Design for class “PayOrderController”



	Name	Return Type	Description
1	requestToPayOrder()		Responsible for initiating a payment request.
2	payOrder()		Handles the payment process after the user is redirected to the payment gateway.
3	saveTransaction()		Responsible for recording the transaction details.

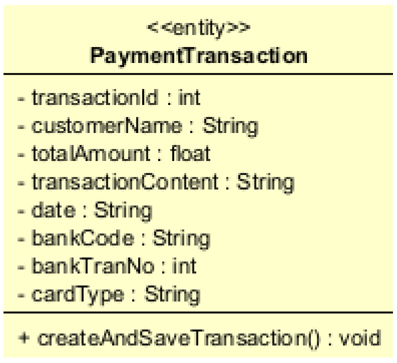
Method

```
public static void RequestToPayOrder(Order order, String paymentMethod) {
        // Validate the order (ensure the order is valid and total amount is greater than 0)
        if (order == null || order.getTotalAmount() <= 0) {
            System.out.println("Invalid order. Payment cannot be processed.");
            return;
        }

        // Prepare payment data to send to VNPay
        Map<String, String> paymentData = new HashMap<>();
        paymentData.put("order_id", order.getOrderId());
        paymentData.put("total_amount", String.valueOf(order.getTotalAmount()));
        paymentData.put("payment_method", paymentMethod);  // CreditCard, PayPal,
etc.
        paymentData.put("customer_email", order.getCustomerEmail());
        paymentData.put("currency", "VND"); // Assumes payment in VND
(Vietnamese Dong)

        // Send payment data to VNPay (simulate sending the payment request)
        String paymentURL = generatePaymentURL(paymentData);

        // Output the payment URL for the user to complete the payment
        System.out.println("Redirecting user to the payment gateway...");
        System.out.println("Payment URL: " + paymentURL);
    }
```

```
public static void payOrder(Order order, String paymentStatus) {
        // Validate payment status (e.g., "success", "failure")
        if ("success".equals(paymentStatus)) {
            // Update the order status to "paid"
            System.out.println("Payment successful for Order ID: " +
order.getOrderId());
            updateOrderStatus(order, "paid");
            // Save transaction details (for record-keeping)
            saveTransaction(order, paymentStatus);
        } else {
            System.out.println("Payment failed for Order ID: " + order.getOrderId());
            updateOrderStatus(order, "failed");
        }
    }
```

```
private static void saveTransaction(Order order, String paymentStatus) {
        // Logic to save transaction details (this is a simulation)
        System.out.println("Saving transaction for Order ID: " + order.getOrderId());
        System.out.println("Transaction details: ");
        System.out.println("Order ID: " + order.getOrderId());
        System.out.println("Total Amount: " + order.getTotalAmount());
        System.out.println("Payment Status: " + paymentStatus);
        System.out.println("Payment Date: " + java.time.LocalDateTime.now());

        // In a real-world application, you would store this data in a database or
transaction log.
    }
```

Design for class “PaymentTransaction”



	Name	Data Type	Default value	Description
1	transactionId	int		Id of the transaction
2	customerName	String		Name of the customer
3	totalAmount	float		Total amount of money that customer have to pay
4	transactionContent	String		Content, message when transaction
5	date	String		Day when the transaction takes place
6	bankCode	String		Code of the bank
7	bankTranNo	int		Number/Id of the transaction corresponding to the bank

8	cardType	String		Type of payment card

Table 1. Example of attribute design

	Name	Return Type	Description
1	createAndSaveTransaction		Create transaction and save for reviewing

Table 2. Example of operation design

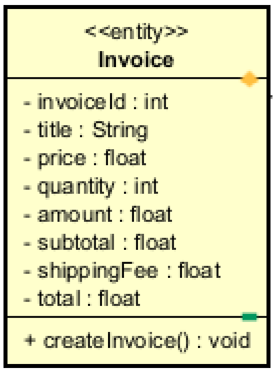
Method

```
public void createAndSaveTransaction() {
        // Simulate saving the transaction (e.g., saving to a database)
        System.out.println("Transaction Created and Saved Successfully:");
        System.out.println("Transaction ID: " + this.transactionId);
        System.out.println("Customer Name: " + this.customerName);
        System.out.println("Total Amount: " + this.totalAmount);
        System.out.println("Transaction Content: " + this.transactionContent);
        System.out.println("Date: " + this.date);
        System.out.println("Bank Code: " + this.bankCode);
        System.out.println("Bank Transaction Number: " + this.bankTranNo);
        System.out.println("Card Type: " + this.cardType);

        // code for saving these details to a database.
    }
```

How to use parameters / attributes

```
private int transactionId;
    private String customerName;
    private float totalAmount;
    private String transactionContent;
    private String date;
    private String bankCode;
    private int bankTranNo;
    private String cardType;

    // Constructor to initialize PaymentTransaction
    public PaymentTransaction(int transactionId, String customerName, float
totalAmount,
                              String transactionContent, String bankCode, int bankTranNo,
String cardType) {
        this.transactionId = transactionId;
        this.customerName = customerName;
        this.totalAmount = totalAmount;
        this.transactionContent = transactionContent;
        this.date = LocalDateTime.now().toString(); // Set current date and time
        this.bankCode = bankCode;
        this.bankTranNo = bankTranNo;
        this.cardType = cardType;
    }

    // Getter methods
    public int getTransactionId() {
        return transactionId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public float getTotalAmount() {
        return totalAmount;
    }

    public String getTransactionContent() {
        return transactionContent;
    }

    public String getDate() {
        return date;
    }

    public String getBankCode() {
        return bankCode;
    }

    public int getBankTranNo() {
        return bankTranNo;
    }

    public String getCardType() {
        return cardType;
    }
```

Design for class "Invoice"



	Name	Data Type	Default value	Description
1	invoiceId	int		Id of the invoice
2	title	String		The title of the invoice, which could represent the product or service.
3	price	float		The price of a single item or unit.
4	quantity	int		The quantity of items being purchased.
5	amount	float		The total cost of the items
6	subtotal	float		The subtotal of the order (before shipping fees or taxes)
7	shippingFee	float		The cost of shipping the items
8	total	float		The cost of shipping the itemsThe total amount to be paid, including the subtotal and shipping fee

Table 1. Example of attribute design

	Name	Return Type	Description
1	createInvoice()		Calculate and set the amount, subtotal, shippingFee, and total based on the provided data and logic.

Table 2. Example of operation design

Method

```
public void createInvoice() {
        // Calculate the amount for the current item (price * quantity)
        this.amount = price * quantity;

        // Calculate the subtotal (just the amount of the items)
        this.subtotal = amount;

        // Calculate the total (subtotal + shipping fee)
        this.total = subtotal + shippingFee;
    }
```

How to use parameters / attributes

```
private int invoiceId;
    private String title;
    private float price;
    private int quantity;
    private float amount;
    private float subtotal;
    private float shippingFee;
    private float total;

    // Constructor to initialize the Invoice object
    public Invoice(int invoiceId, String title, float price, int quantity, float shippingFee)
{
        this.invoiceId = invoiceId;
        this.title = title;
        this.price = price;
        this.quantity = quantity;
        this.shippingFee = shippingFee;
        this.createInvoice();  // Automatically calculate the amounts when the invoice is
created
    }
```