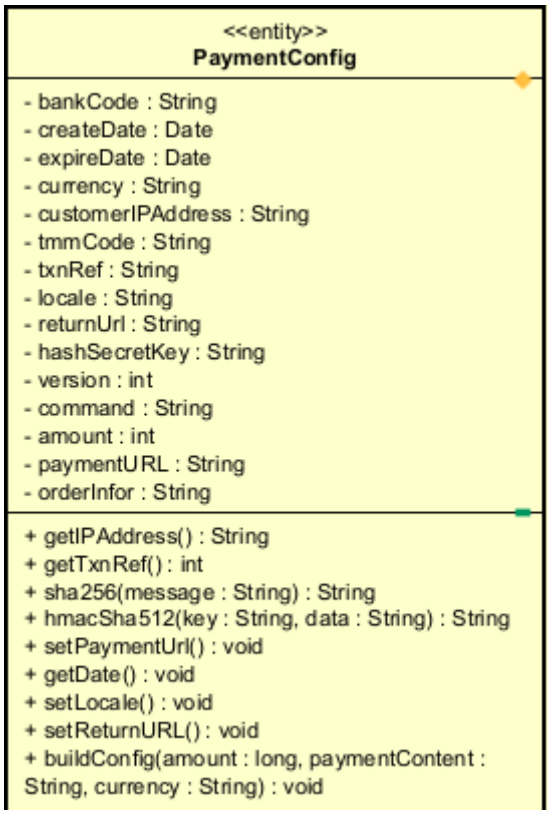
Design for class “PaymentConfig”



	Name	Data Type	Default value	Description
1	bankCode	String		Bank code used for payment
2	createDate	Date		Timestamp when the transaction is created.
3	expireDate	Date		The expiration time of the payment link
4	currency	String		Currency unit(VND)
5	customerIPAddress	String		IP address of the end-user initiating the transaction. Required for security validation.
6	tmnCode	String		VNPay terminal code provided to the merchant.
7	txnRef	String		Unique transaction reference generated by the merchant for tracking the order.
8	locale	String		Language code for VNPay interface
9	returnUrl	String		Merchant URL to which VNPay redirects after payment completion.

10	hashSecretKey	String		Secret key provided by VNPay used to create digital signatures for secure requests.
11	version	int		API version being used
12	command	String		Command type for the transaction
13	amount	Int		Payment amount in the smallest currency unit
14	paymentURL	String		The complete VNPay payment URL generated from the config.
15	orderInfor	String		Description or note for the transaction, visible to the payer.

Table 1. Example of attribute design

	Name	Return Type	Description
1	getIPAddress()	String	Retrieves the IP address of the customer.
2	getTxnRef()	String	Returns the transaction reference code.
3	sha256(message)	String	Computes a SHA-256 hash of the given message. Useful for signing requests.
4	hmacSha512(key, data)	String	Generates an HMAC-SHA512 hash used for secure signing of VNPay requests.
5	setPaymentUrl()	void	Builds and assigns the full payment URL based on current configuration.
6	getDate()	void	Initializes or returns the current creation date for the transaction.
7	setLocale()	void	Sets the language/locale preference for the transaction.
8	setReturnURL()	void	Sets the redirect URL for after the payment is completed.
9	buildConfig(amount, paymentContent, currency)	void	Initializes payment parameters with provided values to prepare the request configuration.

Table 2. Example of operation design

Parameter

Name	Default Value	Description
vnp_Version	"2.1.0"	The version of VNPay

		API to be used.
vnp_Command	"pay"	The command for the transaction, usually 'pay'.
vnp_TmnCode	"ABC123"	Merchant's terminal code issued by VNPay.
vnp_HashSecret	"SECRET_KEY"	Secret key used for secure hash generation and validation.
vnp_PayUrl	"https://sandbox.vnpayment.vn/paymentv2/vpcpay.html"	VNPay payment gateway URL.
vnp_ReturnUrl	"https://yourdomain.com/payment-return"	URL to redirect to after transaction is completed.
vnp_apiUrl	"https://sandbox.vnpayment.vn/merchant_webapi/merchant.html"	API URL for server-to-server operations.

Method

1. sha256(message)

```
public static String sha256(String message)  {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash = digest.digest(message.getBytes("UTF-8"));
    StringBuilder hexString = new StringBuilder();

    for (byte b : hash) {
        String hex = Integer.toHexString(0xff & b);
        if (hex.length() == 1) hexString.append('0');
        hexString.append(hex);
    }

    return hexString.toString();
}
```

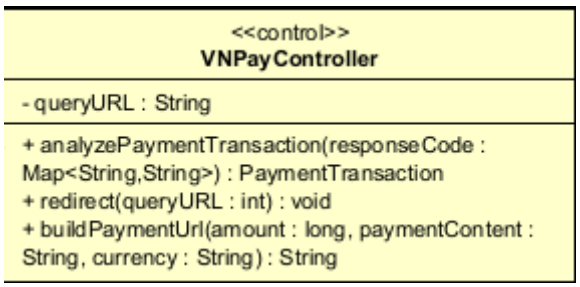
2. hmacSha512(key, data)

```
public static String hmacSha512(String key, String data){
   Mac hmac = Mac.getInstance("HmacSHA512");
   SecretKeySpec secretKey = new SecretKeySpec(key.getBytes("UTF-8"),
"HmacSHA512");
   hmac.init(secretKey);
   byte[] hashBytes = hmac.doFinal(data.getBytes("UTF-8"));

   StringBuilder result = new StringBuilder();
   for (byte b : hashBytes) {
      String hex = Integer.toHexString(0xff & b);
      if (hex.length() == 1) result.append('0');
      result.append(hex);
   }
   return result.toString();
}
```

3. BuildConfig()

```
public void buildConfig(long amount, String paymentContent, String currency)
this.amount = (int) amount * 100;
this.orderInfor = paymentContent; this.currency = currency;
this.createDate = getDate();
this.expireDate = new Date(this.createDate.getTime() + 15 * 60 * 1000);  phút

getIPAdress()
setLocale();
getTxnRef()
setReturnURL();
setPaymentUrl();

}
```

Design for class “VNPayController”



	Name	Return Type	Description
1	buildPaymentUrl	String	Generates a secure VNPay payment URL based on the specified payment amount. This method constructs the URL

			required to redirect the user to VNPay's payment gateway, embedding necessary parameters such as the transaction ID, amount, return URL, and any cryptographic signatures for verification.
2	analyzePaymentTransaction	PaymentTransaction	Parses and analyzes the response parameters returned from VNPay after a transaction attempt. This method extracts relevant information such as the transaction ID, status code, and result message, then maps them into a PaymentTransaction object.
3	redirect	void	Send url to VNPay.

Table 2. Example of operation design

Parameter

Name	Default Value	Description
amount	Long	The total amount to be paid in VND.
responseCode	Map<String,String>	A map containing key-value pairs returned by VNPay after the payment attempt.
queryURL	String	URL including transaction information

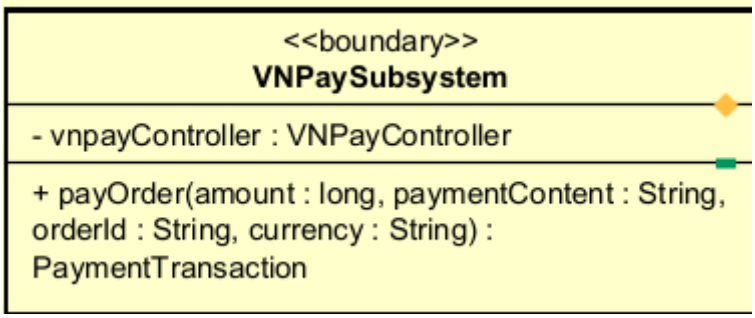
Method

```
public class VNPayController {

    private final String vnp_TmnCode = "ABC123"; // Merchant code
    private final String vnp_HashSecret = "SECRET_KEY"; // Secure secret key
    private final String vnp_PayUrl =
"https://sandbox.vnpayment.vn/paymentv2/vpcpay.html";
    private final String vnp_ReturnUrl = "https://yourdomain.com/payment-return";
    public String buildPaymentUrl(long amount) {
        try {
            String vnp_Version = "2.1.0";
            String vnp_Command = "pay";
            String vnp_OrderInfo = "Payment for order";
            String vnp_TxnRef = UUID.randomUUID().toString();
            String vnp_Locale = "vn";
            String vnp_CurrCode = "VND";
            String vnp_IpAddr = "127.0.0.1";
```

```
        String vnp_CreateDate = new
java.text.SimpleDateFormat("yyyyMMddHHmmss").format(new Date());

        Map<String, String> vnp_Params = new TreeMap<>();
        vnp_Params.put("vnp_Version", vnp_Version);
        vnp_Params.put("vnp_Command", vnp_Command);
        vnp_Params.put("vnp_TmnCode", vnp_TmnCode);
        vnp_Params.put("vnp_Amount", String.valueOf(amount * 100)); // multiply to
smallest unit
        vnp_Params.put("vnp_CurrCode", vnp_CurrCode);
        vnp_Params.put("vnp_TxnRef", vnp_TxnRef);
        vnp_Params.put("vnp_OrderInfo", vnp_OrderInfo);
        vnp_Params.put("vnp_ReturnUrl", vnp_ReturnUrl);
        vnp_Params.put("vnp_IpAddr", vnp_IpAddr);
        vnp_Params.put("vnp_CreateDate", vnp_CreateDate);
        vnp_Params.put("vnp_Locale", vnp_Locale);

        StringBuilder hashData = new StringBuilder();
        StringBuilder query = new StringBuilder();
        for (Map.Entry<String, String> entry : vnp_Params.entrySet()) {
            hashData.append(entry.getKey()).append('=').append(entry.getValue());
            query.append(URLEncoder.encode(entry.getKey(),
StandardCharsets.US_ASCII))
                .append('=')
                .append(URLEncoder.encode(entry.getValue(),
StandardCharsets.US_ASCII));
            hashData.append('&');
            query.append('&');
        }

        // Remove the last &
        hashData.setLength(hashData.length() - 1);
        query.setLength(query.length() - 1);

        String vnp_SecureHash = hmacSHA512(vnp_HashSecret, hashData.toString());
        query.append("&vnp_SecureHash=").append(vnp_SecureHash);

        return vnp_PayUrl + "?" + query.toString();
    } catch (Exception e) {
        throw new RuntimeException("Error generating payment URL", e);
    }
  }
  public PaymentTransaction analyzePaymentTransaction(Map<String, String>
responseParams) {
    String receivedSecureHash = responseParams.remove("vnp_SecureHash");
    Map<String, String> sortedParams = new TreeMap<>(responseParams);
    StringBuilder hashData = new StringBuilder();
    for (Map.Entry<String, String> entry : sortedParams.entrySet()) {
        hashData.append(entry.getKey()).append('=').append(entry.getValue()).append('&');
    }
```

```
        hashData.setLength(hashData.length() - 1); // remove last '&'

        String calculatedHash = hmacSHA512(vnp_HashSecret, hashData.toString());
        if (!calculatedHash.equals(receivedSecureHash)) {
            throw new SecurityException("Invalid VNPay signature.");
        }

        PaymentTransaction transaction = new PaymentTransaction();
        transaction.setTransactionId(responseParams.get("vnp_TxnRef"));
        transaction.setAmount(Long.parseLong(responseParams.get("vnp_Amount")) / 100);
        transaction.setStatus(responseParams.get("vnp_ResponseCode").equals("00") ?
"SUCCESS" : "FAILED");
        transaction.setMessage("VNPay Response: " +
responseParams.get("vnp_ResponseCode"));

        return transaction;
    }

    private String hmacSHA512(String key, String data) throws Exception {
        SecretKeySpec secretKeySpec = new
SecretKeySpec(key.getBytes(StandardCharsets.UTF_8), "HmacSHA512");
        Mac mac = Mac.getInstance("HmacSHA512");
        mac.init(secretKeySpec);
        byte[] bytes = mac.doFinal(data.getBytes(StandardCharsets.UTF_8));
        StringBuilder hash = new StringBuilder();
        for (byte b : bytes) {
            hash.append(String.format("%02x", b));
        }
        return hash.toString();
    }
}
```

Design for class “VNPaySubsystem”



	Name	Data Type	Default value	Description
1	vnpayController	VNPayController		An internal controller used to handle low-level

				operations such as generating the payment URL and analyzing transaction responses.

Table 1. Example of attribute design

	Name	Return Type	Description
1	payOrder	PaymentTransaction	Initiates a payment request to VNPay by using the given payment details and configuration.

Table 2. Example of operation design

Parameter

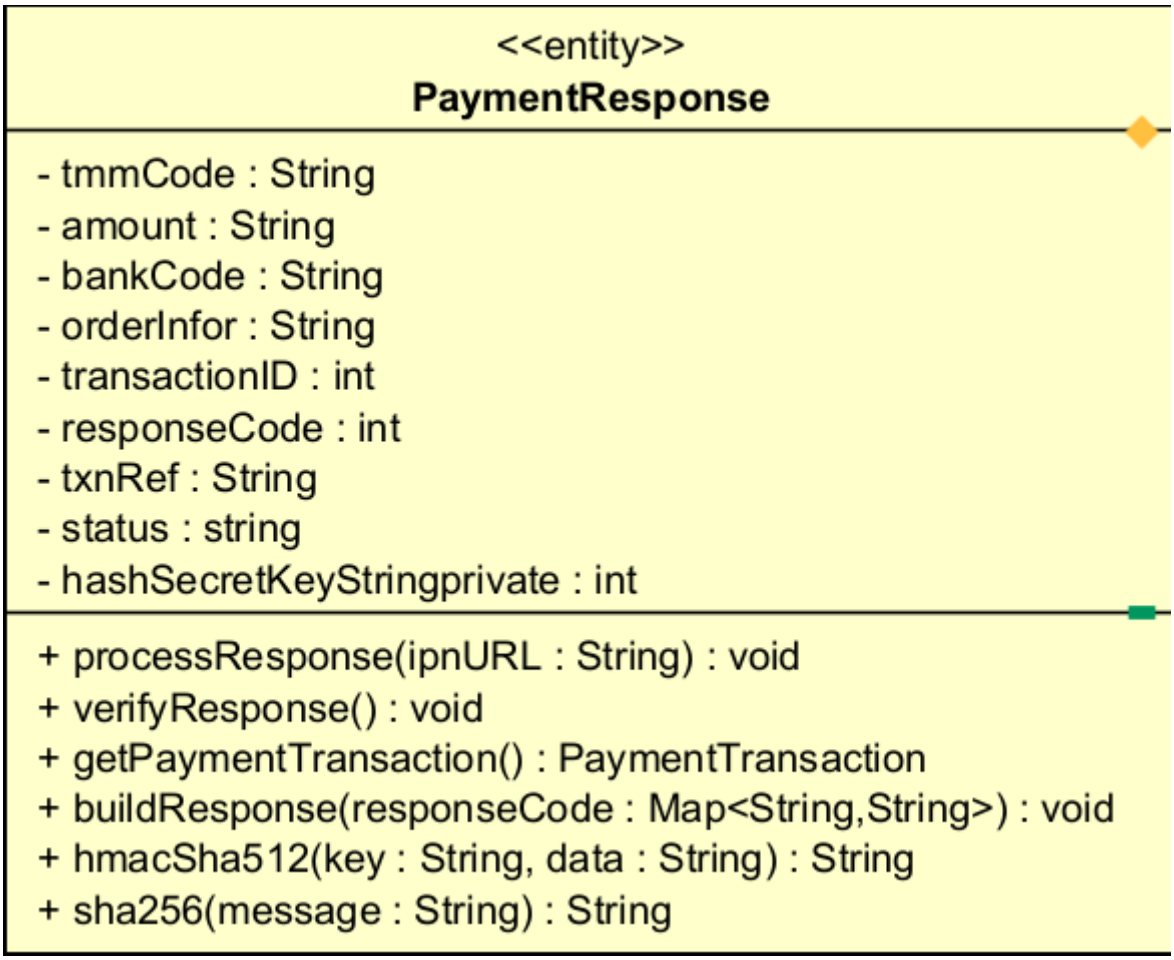
Name	Default Value	Description
amount	long	The total payment amount in Vietnamese Dong
paymentContent	String	A textual description or label for the payment, which will be shown on the VNPay interface and receipts.
orderID	String	A unique identifier for the order or transaction, used to track the payment
paymentConfig	PaymentConfig	Configuration object containing VNPay-specific parameters

Method

```
public class VNPaySubsystem {

    private VNPayController vnpayController;

    public VNPaySubsystem() {
        this.vnpayController = new VNPayController();
    }
    public PaymentTransaction payOrder(long amount, String paymentContent, String
orderID, PaymentConfig paymentConfig) {
        try {
            paymentConfig.buildConfig(amount, paymentContent, "VND");  // or get from
method args
            String paymentUrl = vnpayController.buildPaymentUrl(amount);
```

```
            System.out.println("Redirect user to: " + paymentUrl);
            Map<String, String> dummyResponse = new HashMap<>();
            dummyResponse.put("vnp_TmnCode", paymentConfig.getVnpTmnCode());
            dummyResponse.put("vnp_Amount", String.valueOf(amount * 100));
            dummyResponse.put("vnp_TxnRef", orderID);
            dummyResponse.put("vnp_ResponseCode", "00");
            dummyResponse.put("vnp_TransactionNo", "10000001");
            return vnpayController.analyzePaymentTransaction(dummyResponse);
        } catch (Exception e) {
            PaymentTransaction failed = new PaymentTransaction();
            failed.setTransactionId(orderID);
            failed.setAmount(amount);
            failed.setStatus("FAILED");
            failed.setMessage("Error processing payment: " + e.getMessage());
            return failed;
        }
    }
}
```

Design for class “PaymentResponse”



	Name	Data Type	Default value	Description
1	tmmCode	String		Merchant terminal code assigned by VNPay to identify the seller or store.
2	amount	String		Transaction amount in the smallest currency unit (e.g., 10000 for 100 VND).
3	bankCode	String		Bank code used by the customer to make the payment (optional).
4	orderInfor	String		Description of the order shown on the VNPay payment page and invoice.
5	transactionID	int		VNPay's transaction ID used to uniquely identify the payment.
6	responseCode	int		Response code from VNPay indicating transaction result (e.g., '00' for success).
7	txnRef	String		Merchant-generated order reference ID to track and match the transaction.
8	status	String		Overall status of the transaction, such as 'SUCCESS', 'FAILED', or 'REVERSED'.
9	hashSecretKeyStringprivate	int		Likely a typo; possibly intended as 'hashSecretKey: String'. Refers to the secret key for validating VNPay's response signature.

Table 1. Example of attribute design

	Name	Return Type	Description
1	processResponse	void	ipnURL: String — Processes VNPay's IPN URL and extracts response data.
2	verifyResponse	void	Validates the integrity and signature of the response.
3	getPaymentTransaction	PaymentTransaction	Returns a populated PaymentTransaction object based on response.
4	buildResponse	void	responseCode: Map<String, String> — Builds internal state from VNPay response parameters.

5	hmacSha512	String	key: String, data: String — Generates an HMAC SHA512 hash.
6	sha256	String	message: String — Computes SHA256 hash of the input string.

Table 2. Example of operation design

Parameter

Name	Default Value	Description
ipnURL	"https://yourdomain.com/vnpay_ipn"	The Instant Payment Notification URL received from VNPay for processing.
responseCode	{}	Map<String, String> containing VNPay response parameters such as transaction ID, amount, response code, etc.
key	"SECRET_KEY"	Merchant's secret key used to generate or verify HMAC.
data	"vnp_Amount=100000&vnp_TxnRef=123"	Raw data string used as input to generate HMAC.
message	"Input string"	Text to be hashed using the SHA-256 algorithm.
amount	100000	Payment amount passed to buildConfig method.
paymentContent	"Payment for order #123"	Description or purpose of the payment.
currency	"VND"	Currency code used for the transaction.

Method

```
public class PaymentResponse {
    private String tmmCode;
    private String amount;
    private String bankCode;
    private String orderInfor;
    private int transactionID;
    private int responseCode;
    private String txnRef;
    private String status;
```

```
    private String hashSecretKey;
    public PaymentResponse(String hashSecretKey) {
        this.hashSecretKey = hashSecretKey;
    }
    public void processResponse(String ipnURL) {
        // Dummy example - parse query string manually (in real code, use proper URL
parser)
        Map<String, String> params = new TreeMap<>();
        String[] parts = ipnURL.split("\\?");
        if (parts.length > 1) {
            String[] pairs = parts[1].split("&");
            for (String pair : pairs) {
                String[] kv = pair.split("=");
                if (kv.length == 2) {
                    params.put(kv[0], kv[1]);
                }
            }
        }
        buildResponse(params);
    }
    public void verifyResponse() {
        if (this.hashSecretKey == null) throw new RuntimeException("Secret key not set.");

        Map<String, String> data = new TreeMap<>();
        data.put("vnp_TmnCode", tmmCode);
        data.put("vnp_Amount", amount);
        data.put("vnp_BankCode", bankCode);
        data.put("vnp_TxnRef", txnRef);
        data.put("vnp_ResponseCode", String.valueOf(responseCode));

        StringBuilder hashData = new StringBuilder();
        for (Map.Entry<String, String> entry : data.entrySet()) {
            hashData.append(entry.getKey()).append('=').append(entry.getValue()).append('&');
        }
        hashData.setLength(hashData.length() - 1);

        String hash = hmacSha512(hashSecretKey, hashData.toString());
        // Normally compare hash to received vnp_SecureHash (not available here)
        System.out.println("Computed hash: " + hash);
    }
    public PaymentTransaction getPaymentTransaction() {
        PaymentTransaction transaction = new PaymentTransaction();
        transaction.setTransactionId(txnRef);
        transaction.setAmount(Long.parseLong(amount) / 100);
        transaction.setStatus(responseCode == 0 ? "SUCCESS" : "FAILED");
        transaction.setMessage("Processed VNPay transaction with code " + responseCode);
        return transaction;
    }
    public void buildResponse(Map<String, String> responseCodeMap) {
        this.tmmCode = responseCodeMap.get("vnp_TmnCode");
```

```
        this.amount = responseCodeMap.get("vnp_Amount");
        this.bankCode = responseCodeMap.get("vnp_BankCode");
        this.orderInfor = responseCodeMap.get("vnp_OrderInfo");
        this.txnRef = responseCodeMap.get("vnp_TxnRef");

        try {
            this.responseCode =
Integer.parseInt(responseCodeMap.getOrDefault("vnp_ResponseCode", "99"));
            this.transactionID =
Integer.parseInt(responseCodeMap.getOrDefault("vnp_TransactionNo", "0"));
        } catch (NumberFormatException e) {
            this.responseCode = 99;
            this.transactionID = 0;
        }

        this.status = (this.responseCode == 0) ? "SUCCESS" : "FAILED";
    }
    public String hmacSha512(String key, String data) {
        try {
            Mac hmac = Mac.getInstance("HmacSHA512");
            SecretKeySpec secretKey = new
SecretKeySpec(key.getBytes(StandardCharsets.UTF_8), "HmacSHA512");
            hmac.init(secretKey);
            byte[] hash = hmac.doFinal(data.getBytes(StandardCharsets.UTF_8));

            StringBuilder sb = new StringBuilder();
            for (byte b : hash) {
                sb.append(String.format("%02x", b));
            }
            return sb.toString();
        } catch (Exception ex) {
            throw new RuntimeException("Error while generating HMAC SHA512", ex);
        }
    }

    public String sha256(String message) {
        try {
            java.security.MessageDigest md = java.security.MessageDigest.getInstance("SHA-
256");
            byte[] hash = md.digest(message.getBytes(StandardCharsets.UTF_8));
            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                hexString.append(String.format("%02x", b));
            }
            return hexString.toString();
        } catch (Exception ex) {
            throw new RuntimeException("Error while generating SHA-256 hash", ex);
        }
    }
}
```

Exception

Exception	When It Occurs
NotEnoughBalanceException	When user doesn’t have enough balance.
SuspiciousException	When the transaction is flagged as suspicious (e.g. multiple fast transactions).
AccountLockedException	When user's account is locked due to policy or suspicious behavior.
ExceededPasswordAttemptsException	When the user entered the wrong password too many times.
UnknownPaymentException	When the system cannot recognize or find the payment request.
OrderNotFound	When the provided order ID does not exist.
OrderAlreadyConfirmed	When the order was already confirmed previously.
InvalidVerification	When verification information (OTP, signature) is invalid.
ExceededDailyLimitException	When the transaction exceeds the daily allowed limit.
BankMaintenanceException	When the bank’s system is under maintenance.
InsufficientBalanceException	When the selected bank account doesn’t have enough funds.
TransactionCancelledException	When the transaction was manually or automatically cancelled.
TransactionTimeoutException	When the payment gateway does not respond in time.
ExceededVerificationAttemptsException	When too many invalid verification attempts have been made.

```
public class PaymentException extends PaymentException {
    public PaymentException() {
        super("PaymentException occurred.");
    }
}

public class NotEnoughBalanceException extends PaymentException {
    public NotEnoughBalanceException() {
        super("NotEnoughBalanceException occurred.");
    }
}

public class SuspiciousException extends PaymentException {
    public SuspiciousException() {
        super("SuspiciousException occurred.");
    }
```

```
}

public class AccountLockedException extends PaymentException {
    public AccountLockedException() {
        super("AccountLockedException occurred.");
    }
}

public class ExceededPasswordAttemptsException extends PaymentException {
    public ExceededPasswordAttemptsException() {
        super("ExceededPasswordAttemptsException occurred.");
    }
}

public class UnknownPaymentException extends PaymentException {
    public UnknownPaymentException() {
        super("UnknownPaymentException occurred.");
    }
}

public class OrderNotFound extends PaymentException {
    public OrderNotFound() {
        super("OrderNotFound occurred.");
    }
}

public class OrderAlreadyConfirmed extends PaymentException {
    public OrderAlreadyConfirmed() {
        super("OrderAlreadyConfirmed occurred.");
    }
}

public class InvalidVerification extends PaymentException {
    public InvalidVerification() {
        super("InvalidVerification occurred.");
    }
}

public class ExceededDailyLimitException extends PaymentException {
    public ExceededDailyLimitException() {
        super("ExceededDailyLimitException occurred.");
    }
}

public class BankMaintenanceException extends PaymentException {
    public BankMaintenanceException() {
        super("BankMaintenanceException occurred.");
    }
}
```

```
public class InsufficientBalanceException extends PaymentException {
    public InsufficientBalanceException() {
        super("InsufficientBalanceException occurred.");
    }
}

public class TransactionCancelledException extends PaymentException {
    public TransactionCancelledException() {
        super("TransactionCancelledException occurred.");
    }
}

public class TransactionTimeoutException extends PaymentException {
    public TransactionTimeoutException() {
        super("TransactionTimeoutException occurred.");
    }
}

public class ExceededVerificationAttemptsException extends PaymentException {
    public ExceededVerificationAttemptsException() {
        super("ExceededVerificationAttemptsException occurred.");
    }
}
```