



SOFTWARE ENGINEERING

System Modelling

Dr. Rathachai Chawuthai

Department of Computer Engineering
Faculty of Engineering

King Mongkut's Institute of Technology Ladkrabang

Agenda

- Use Case Diagram
- Collaboration Diagram
- Sequence Diagram
- Class Diagram

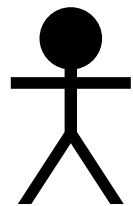
Overview



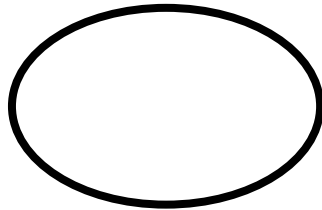
Why?



All Diagrams



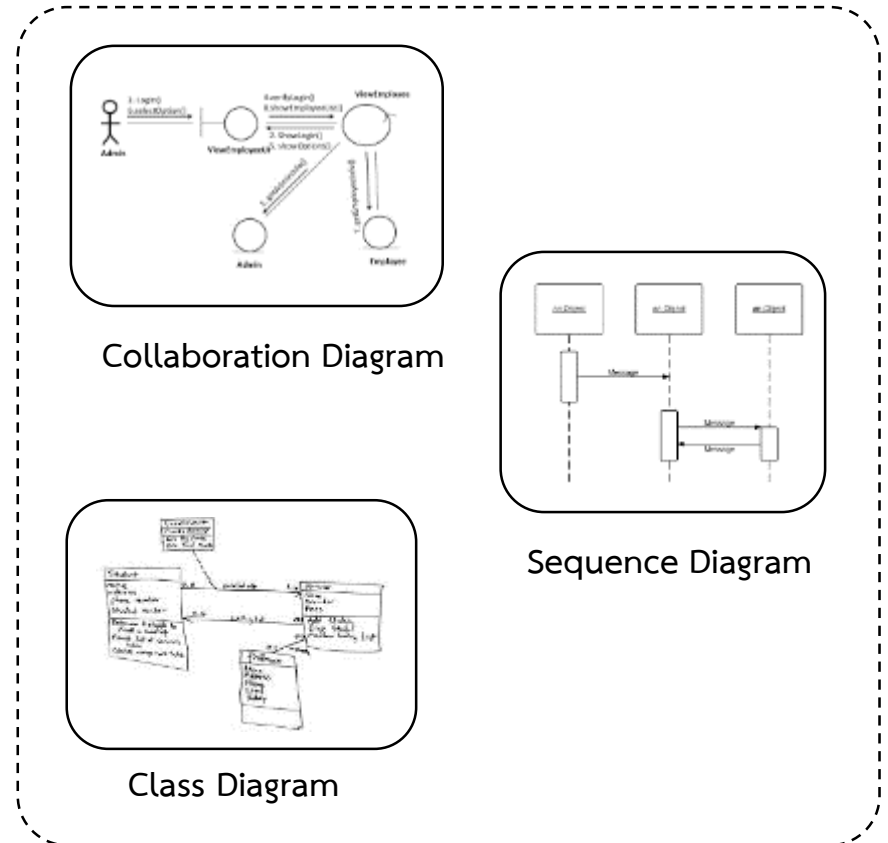
User



Use Case
Diagram



Use Case
Specification



Collaboration Diagram

Sequence Diagram

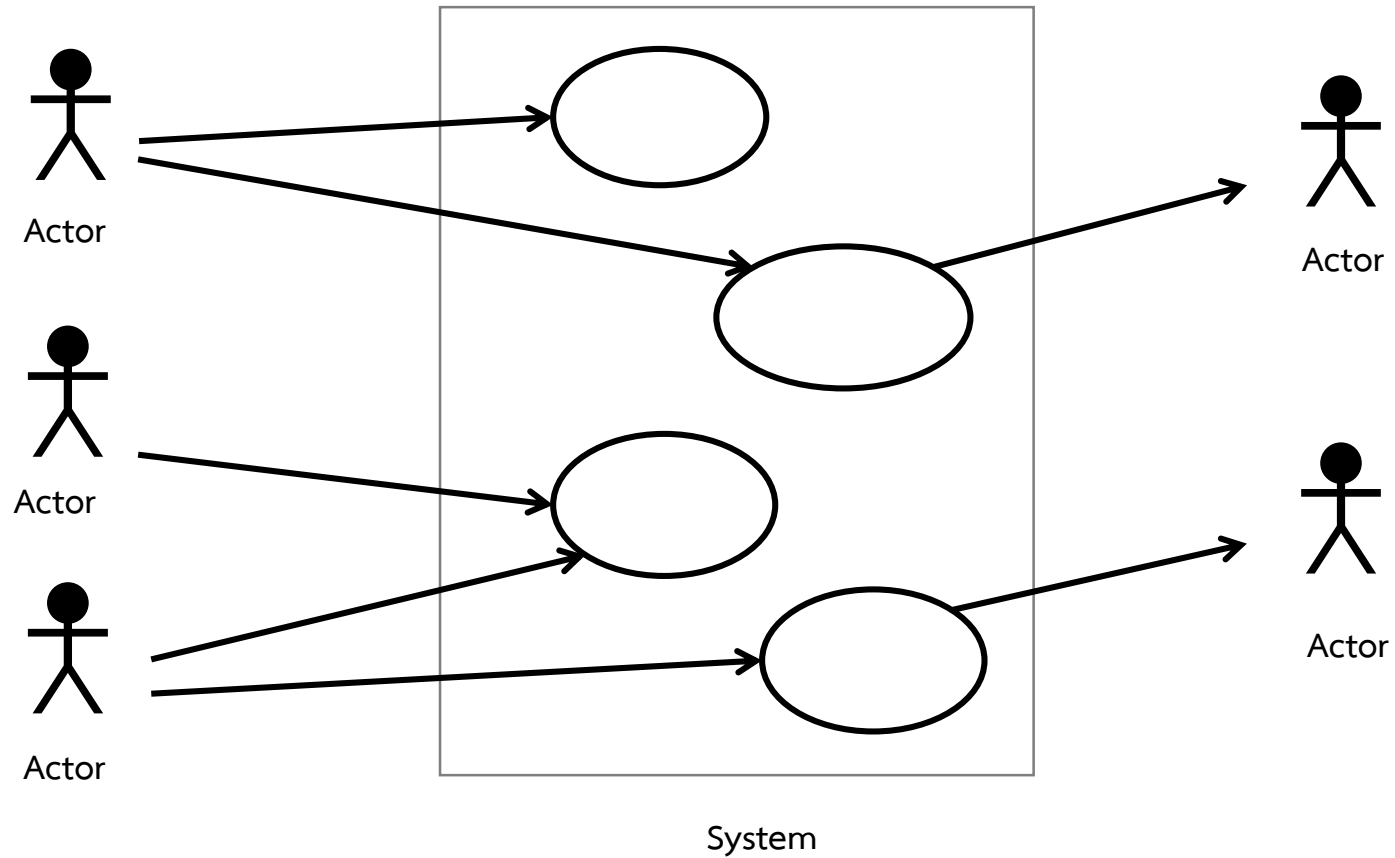
Class Diagram

Use Case Realization

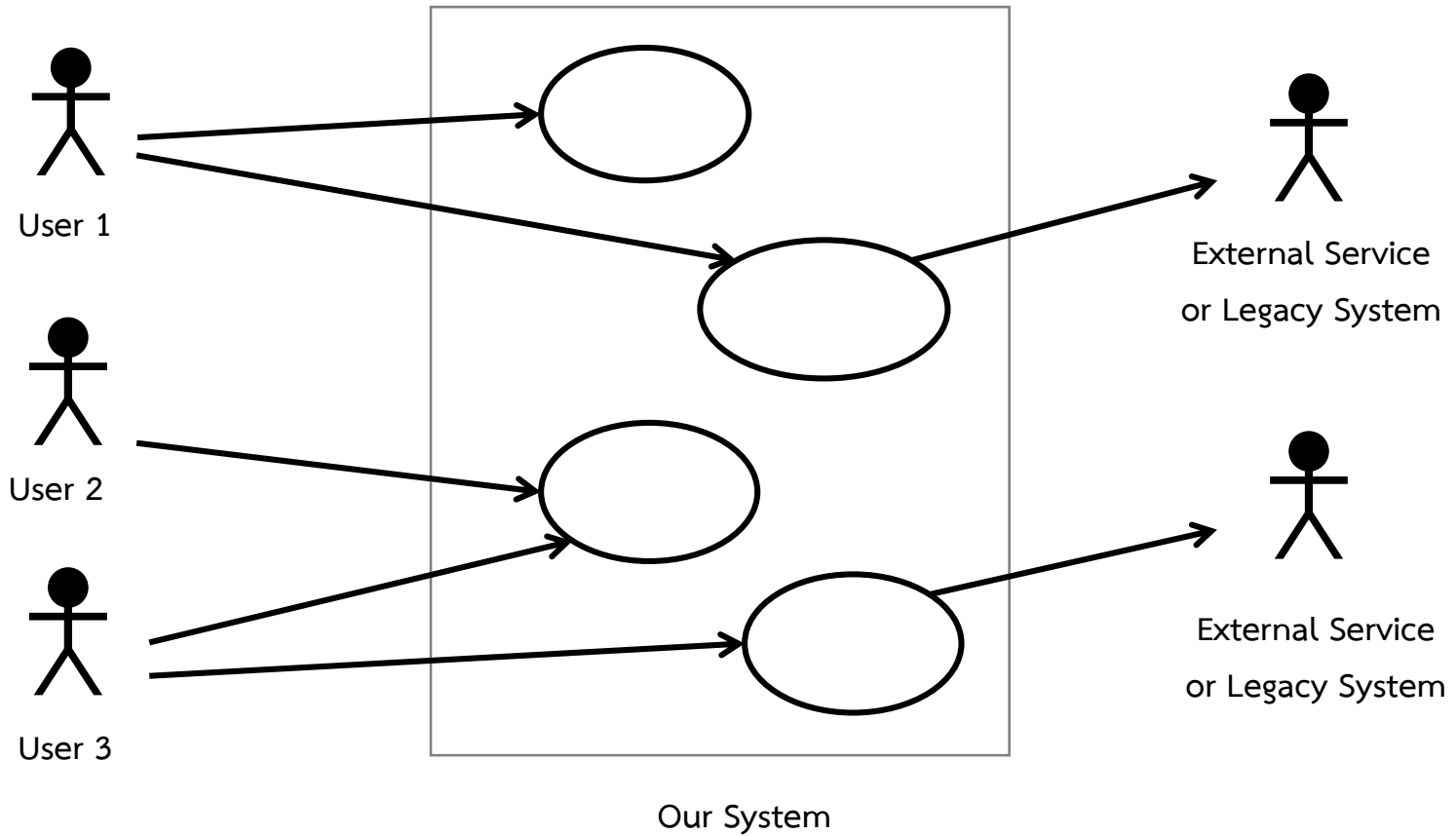
Use Case Diagram



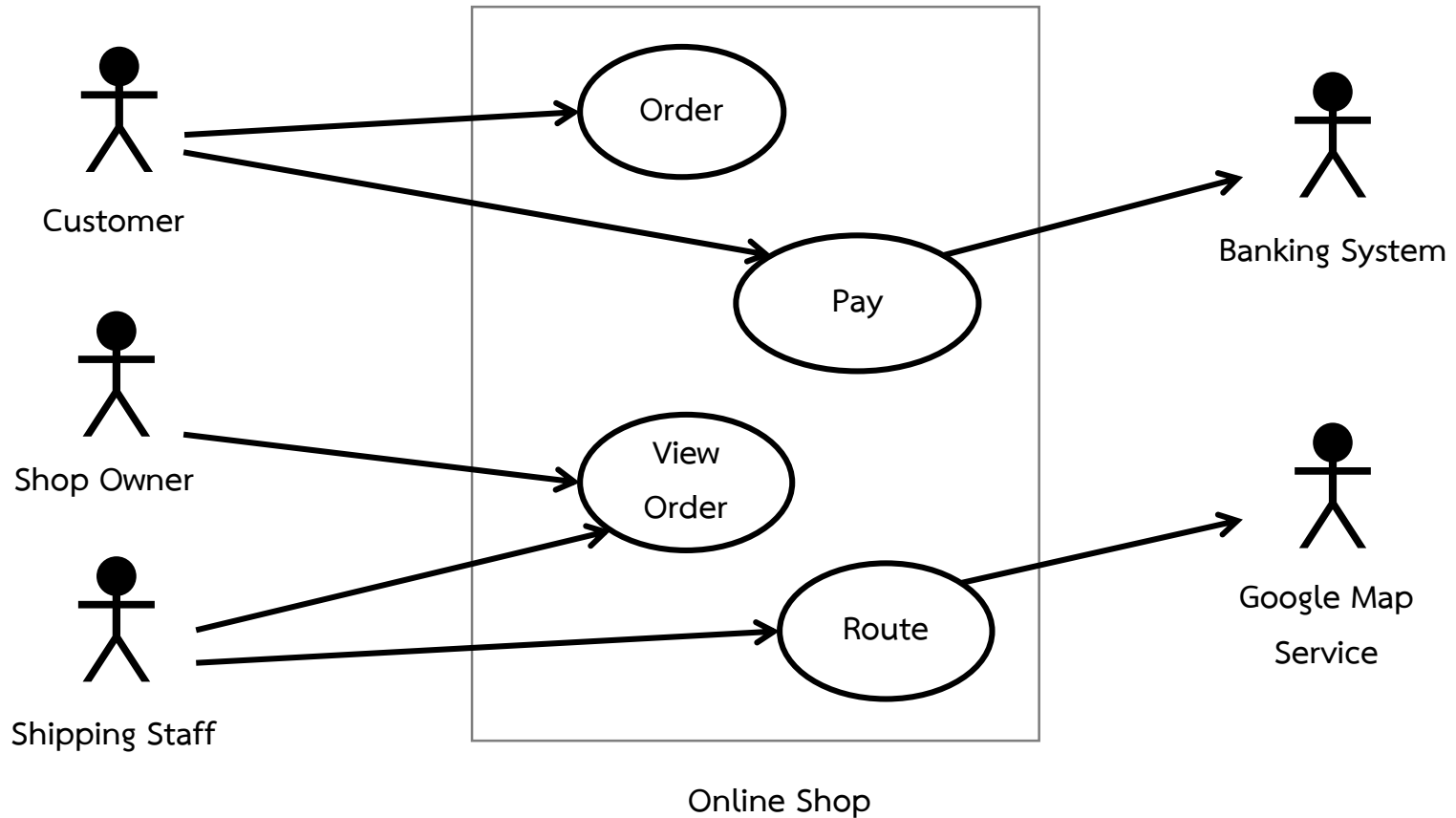
Use Case Diagram



Use Case Diagram



Use Case Diagram : Online Shop



Use-Case Flow of Events

- มี 1 Basic Flow

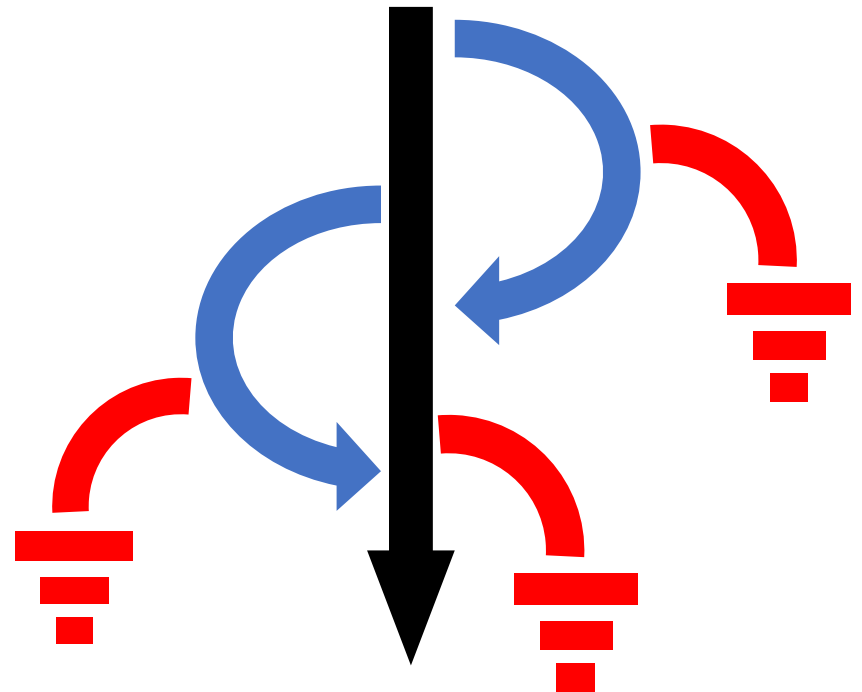


- มี Alternative Flow ได้หลายอัน

a) Regular Variance
เส้นทางอื่นที่ทำได้



a) Exceptional Flows
สำหรับ Error



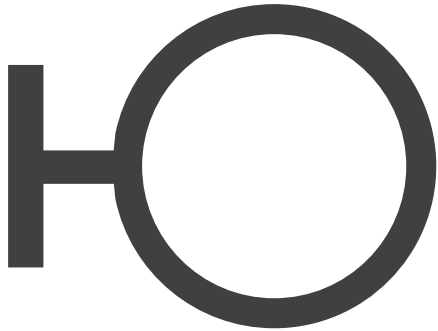
Flow of Event : Pay

- Basic Flow
 - ลูกค้ายืนยันรายการสินค้าเพื่อจ่ายเงิน
 - Online Shop แจ้ง Banking System ว่าลูกค้า ID นี้ จะจ่ายเงินเท่านี้ สำหรับ Order ID นี้
 - ลูกค้าชำระเงินผ่าน Banking System
 - Banking System แจ้ง Online Shop ว่าการชำระเงินครบถ้วนถูกต้อง
 - ลูกค้าได้รับ Message ยืนยันว่าชำระเงินถูกต้อง
- Alternative Flows
 - ระบบ Banking ล่ม
 - ลูกค้าจ่ายเงินไม่ครบ
 - ลูกค้าปิดหน้าเว็บการซื้อ

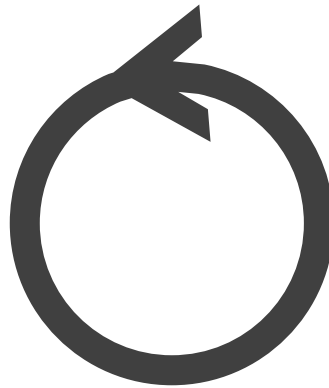
Collaboration Diagram



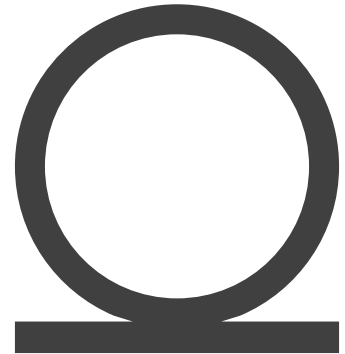
Elements



Boundary



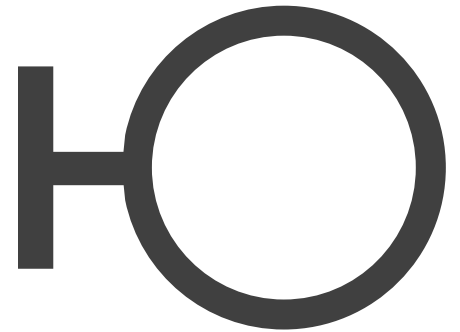
Control



Entity

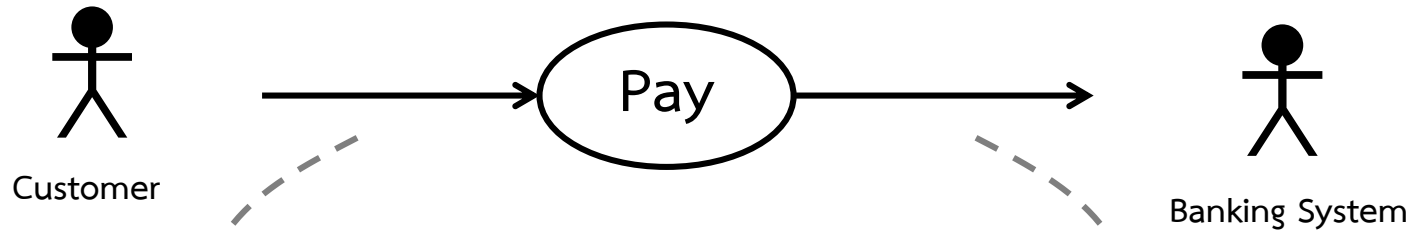
Boundary

- เป็น Interface อยู่ระหว่างระบบที่จะพัฒนากับ users หรือ external systems
- มีดังนี้
 - User interface classes
 - System interface classes
 - Device interface classes
- มี 1 boundary class ต่อ 1 คู่ของ actor กับ use-case



Boundary

Use Case
Model

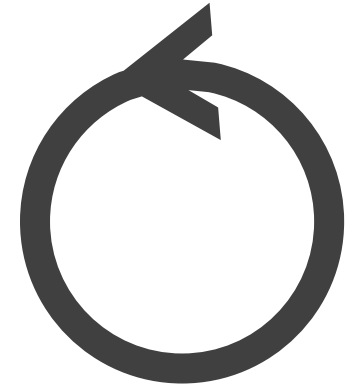


Design
Model

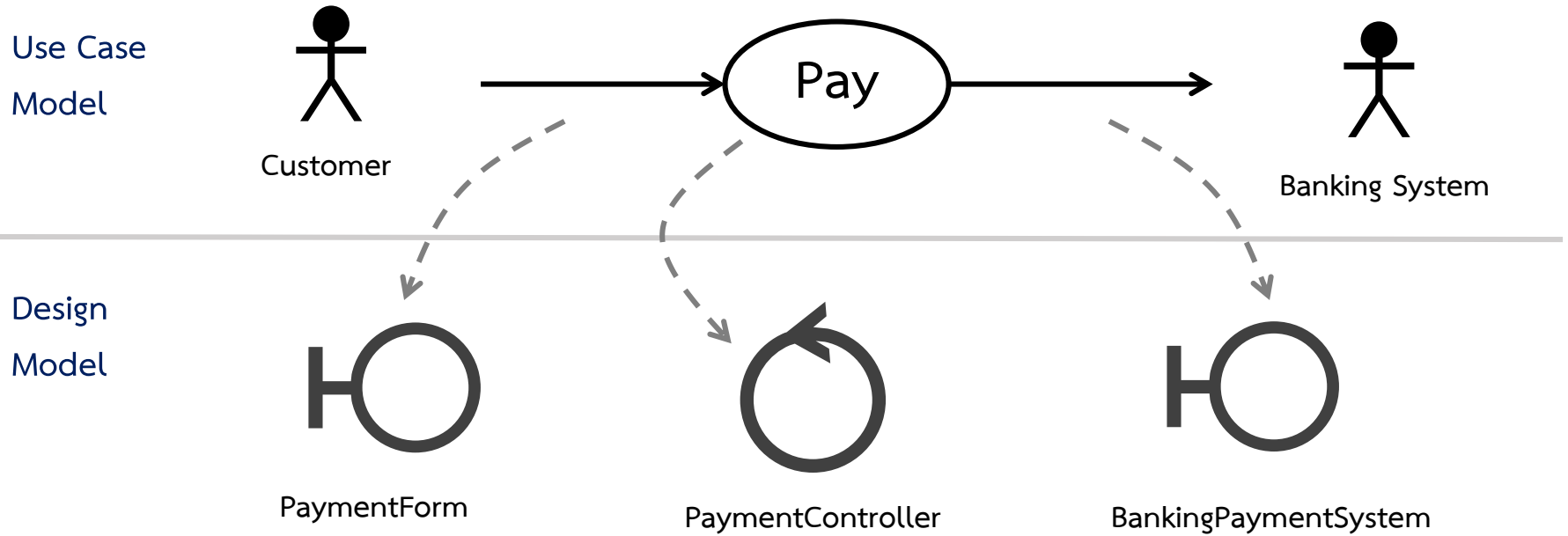


Control

- เป็นส่วนที่ไว้ใช้ควบคุมกระบวนการทั้งหมดของ Use Case นั้น
- ปกติจะมี 1 Control ต่อ 1 Use Case (แต่ถ้า Use Case ซับซ้อนมากจะมีมากกว่า 1 Control ก็ได้)

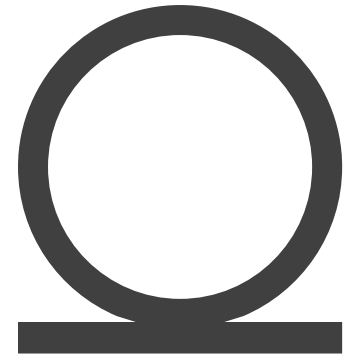


Control



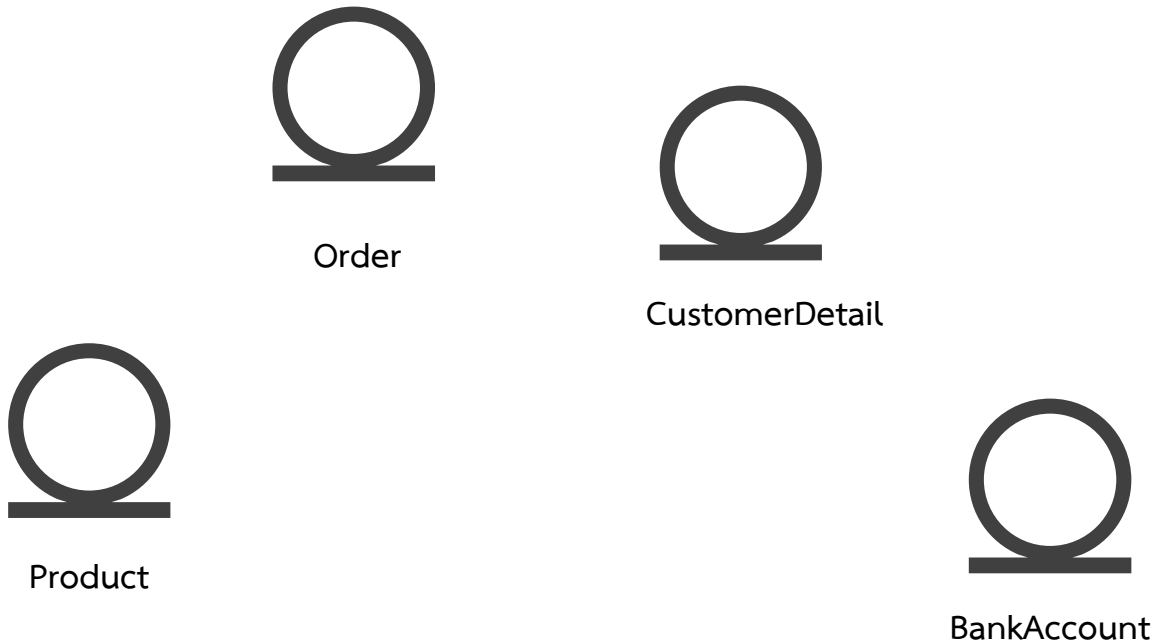
Entity

- เป็น Object แสดงหน่วยข้อมูลต่างๆ ที่ใช้ใน Use Case นี้
- เป็น object ที่ใช้เป็นทั้ง Input, Output, และ ส่วนประกอบอื่นๆ ที่จำเป็น
- ข้อสังเกต
 - เป็นคำนาม
 - ต้องไม่ซ้ำซ้อน
 - ต้องไม่กำกวม
 - ไม่รวม actor
 - เป็น object ไม่ใช่ attributes



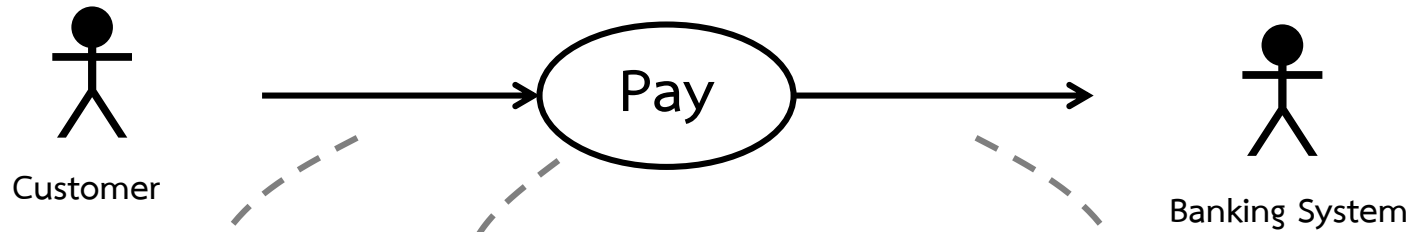
Entity

- ตัวอย่างของ Payment

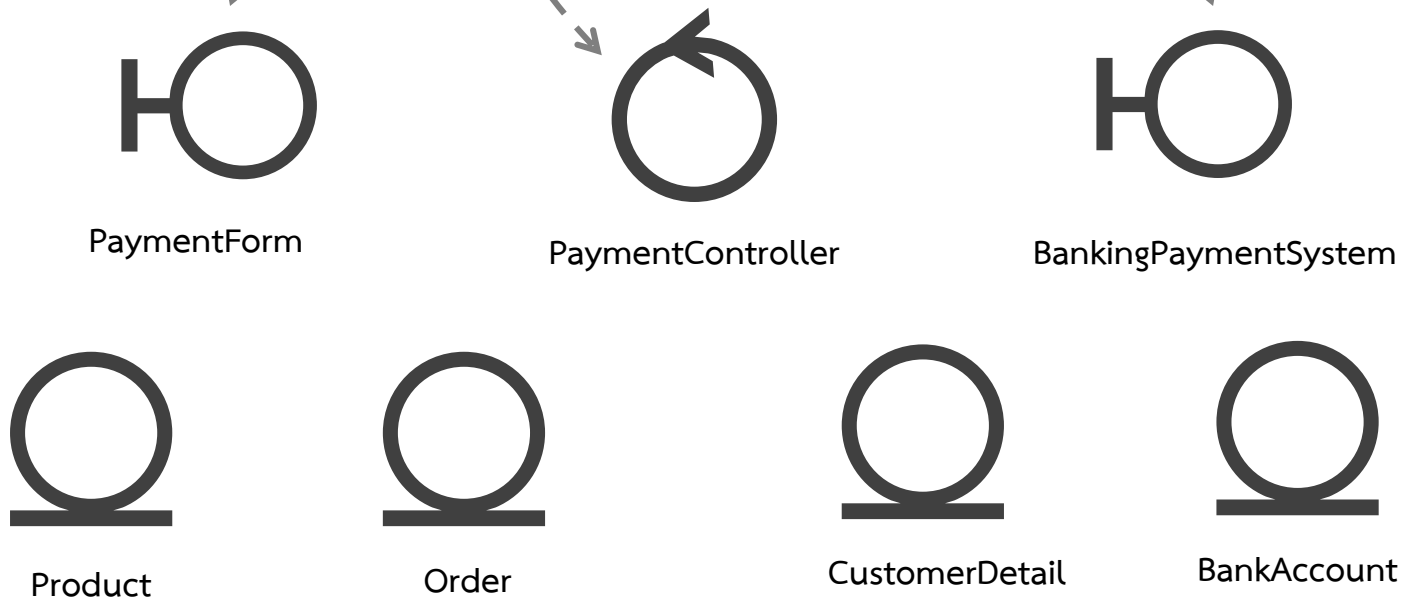


Entity

Use Case
Model



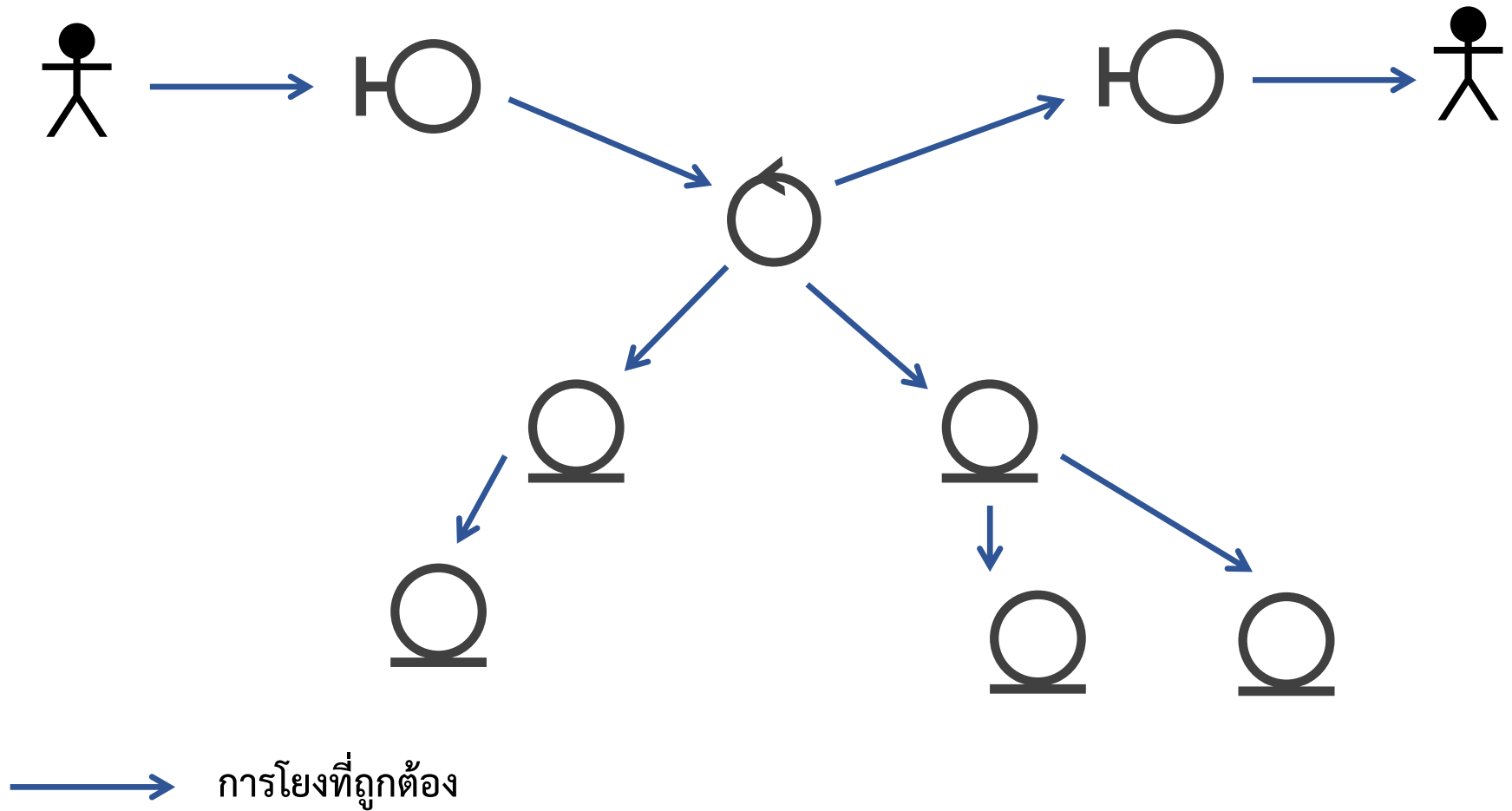
Design
Model



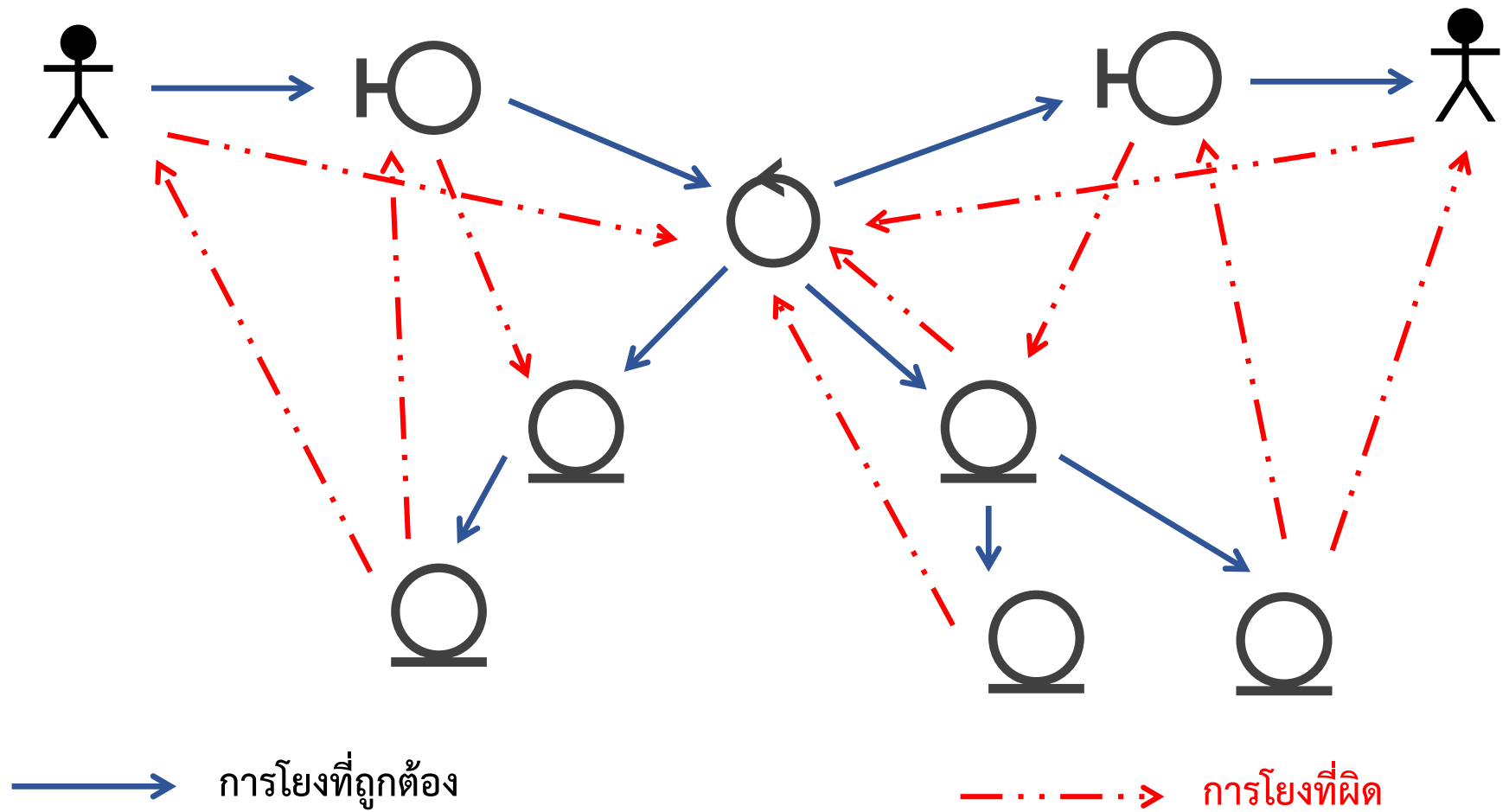
Collaboration Diagram

- แต่ละ element จะมีลูกศรโยงหาความสัมพันธ์กันดังนี้
 - โยงลูกศรระหว่าง actor กับ boundary เท่านั้น
 - โยงลูกศรระหว่าง boundary กับ control เท่านั้น
 - โยงลูกศรระหว่าง control ไปยัง entity อย่างน้อย 1 การเชื่อมโยง
 - โยงลูกศรระหว่าง entity กับ entity ได้
 - ห้ามโยงลูกศรจาก control ไปยัง actor เด็ดขาด
 - ห้ามโยงลูกศรจาก entity ไป control หรือ boundary หรือ actor เด็ดขาด
- การเขียน action กำกับ
 - เขียนหมายเลขลำดับของ action บน (หรือบริเวณ) เส้นที่โยง
- ข้อเสนอแนะ
 - การโยงลูกศรระหว่าง entity ควรทำให้เรียบง่ายที่สุด ไม่ควรให้เกิดการโยงซับซ้อนหรือเกิด loop

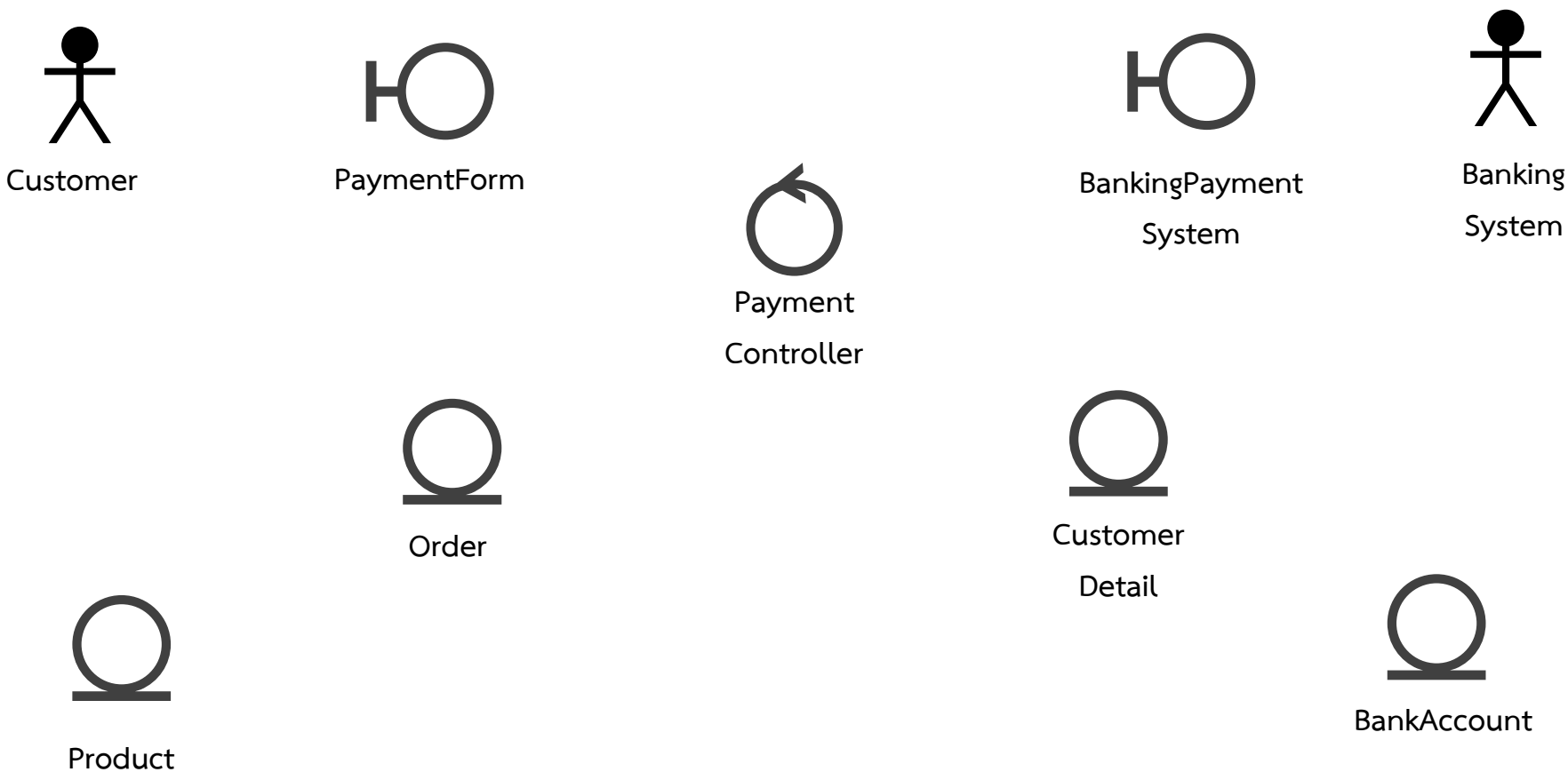
Collaboration Diagram



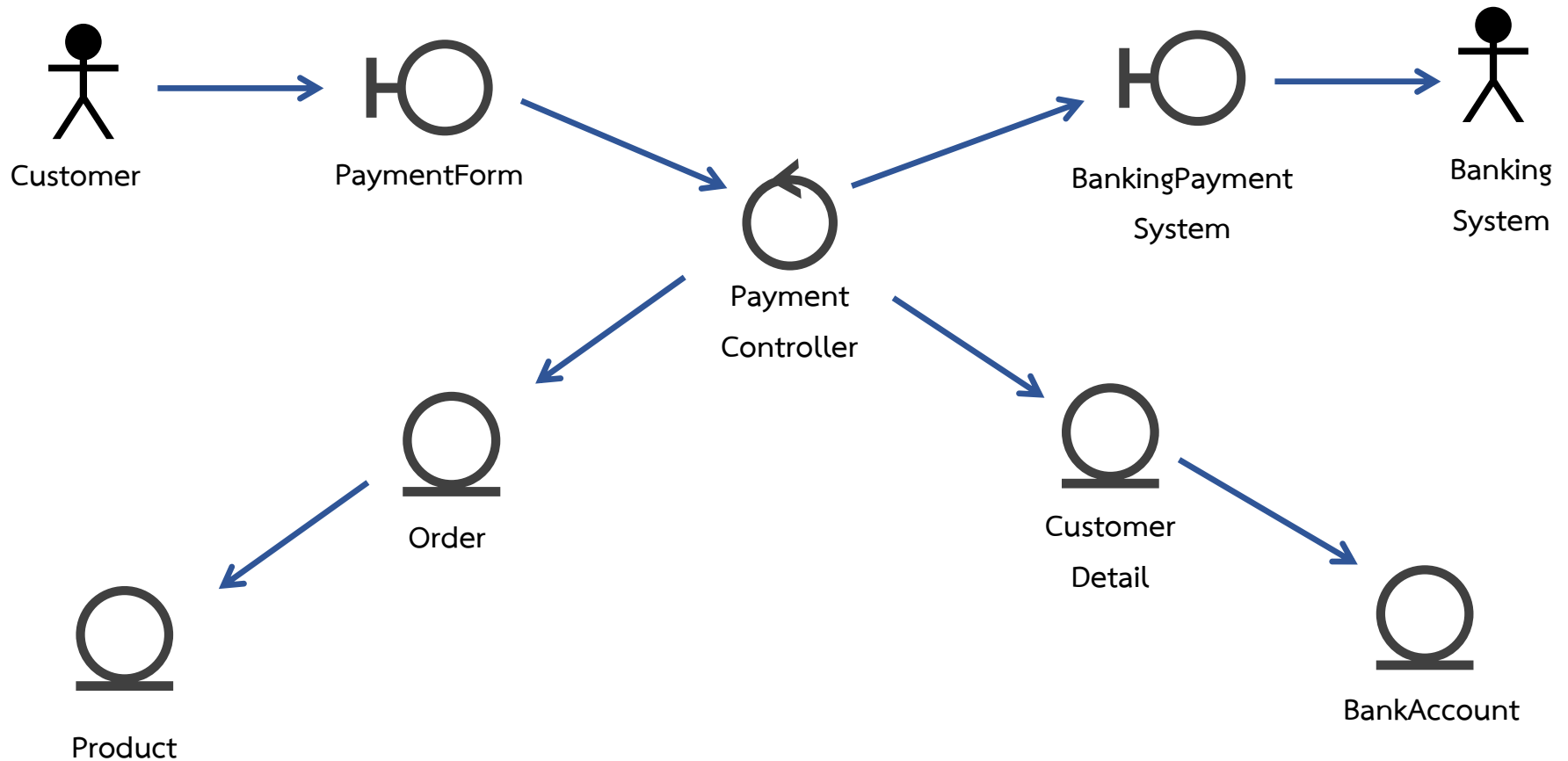
Collaboration Diagram



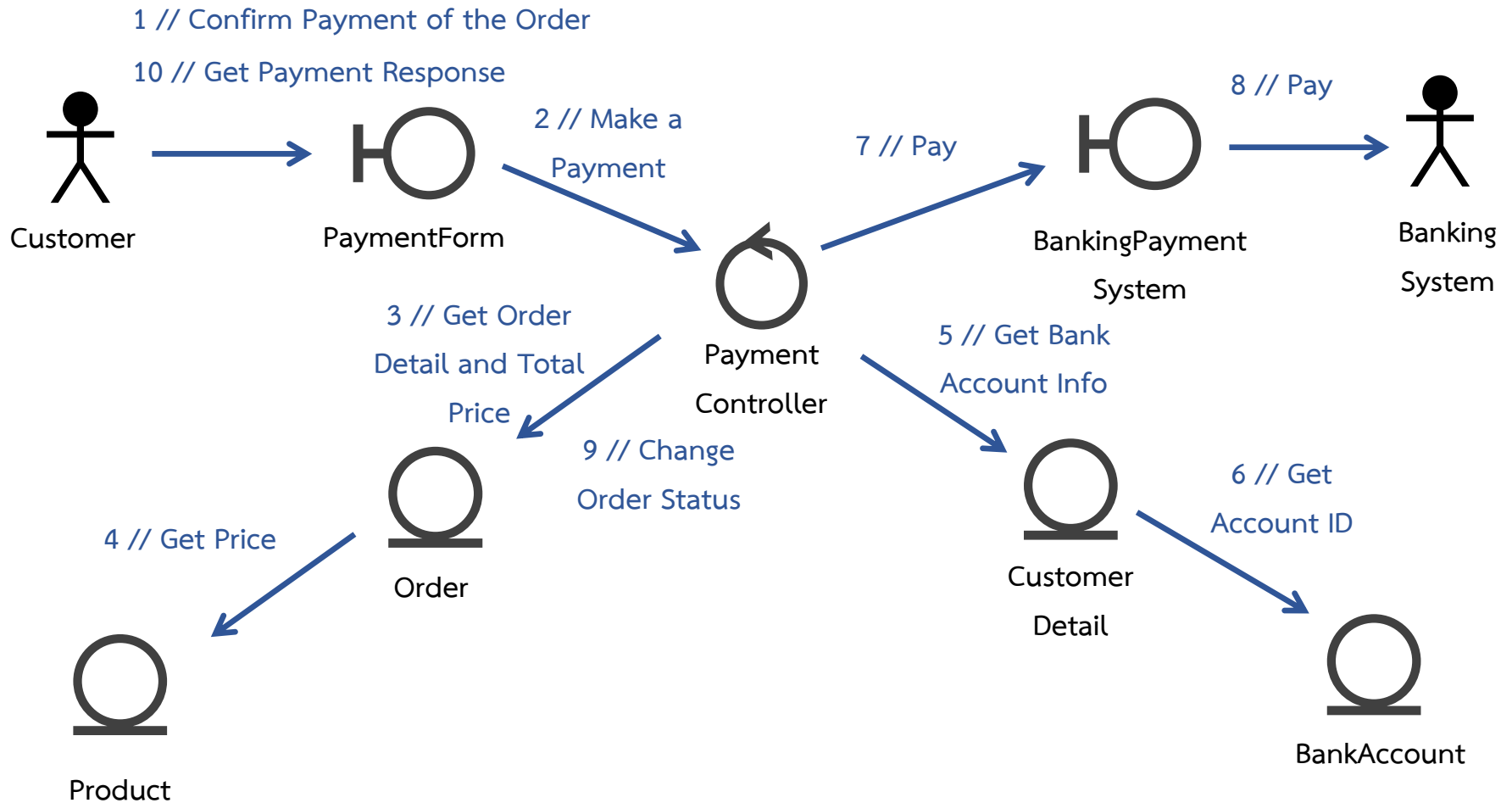
Collaboration Diagram: Payment



Collaboration Diagram: Payment



Collaboration Diagram: Payment



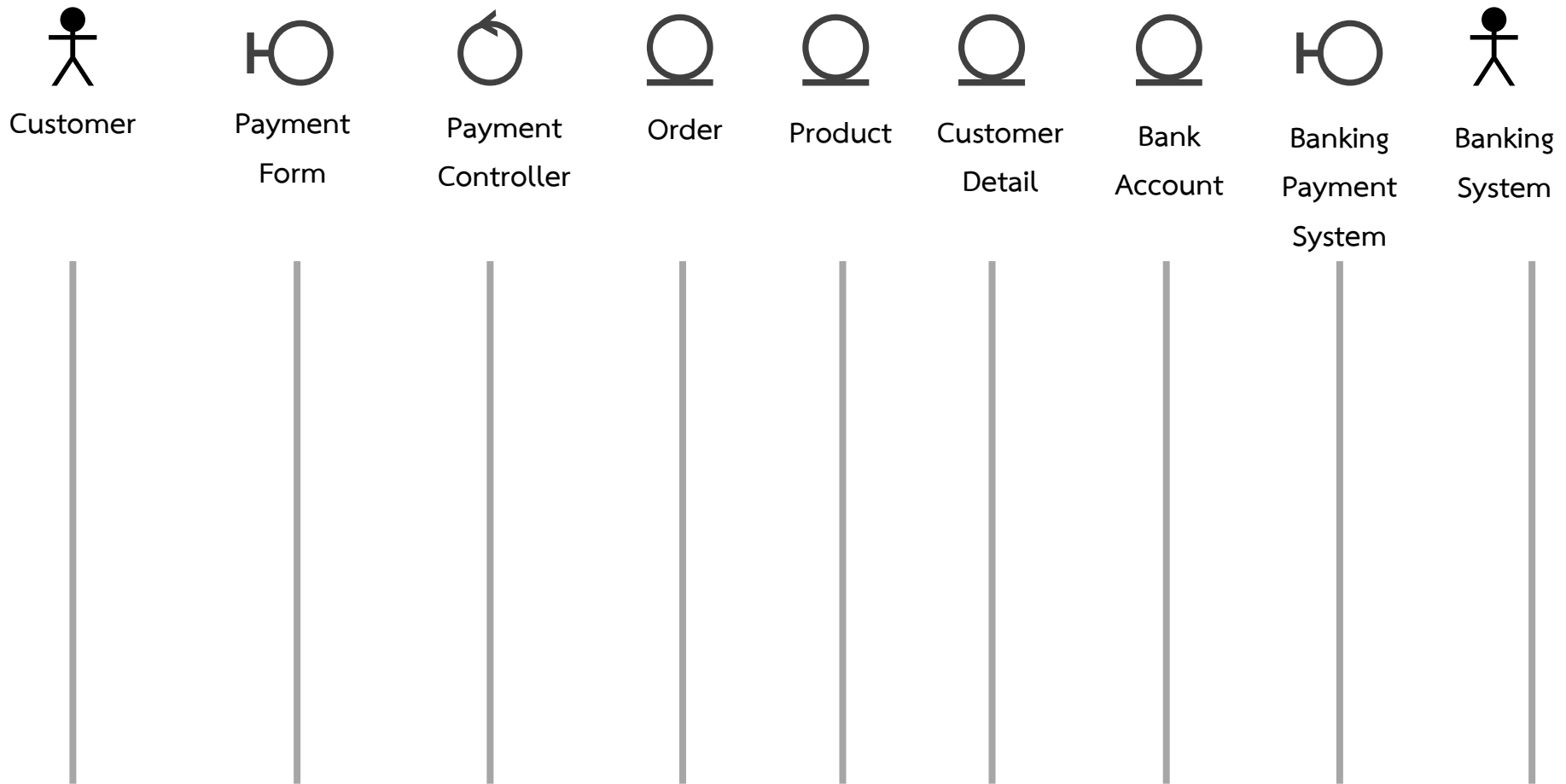
Sequence Diagram



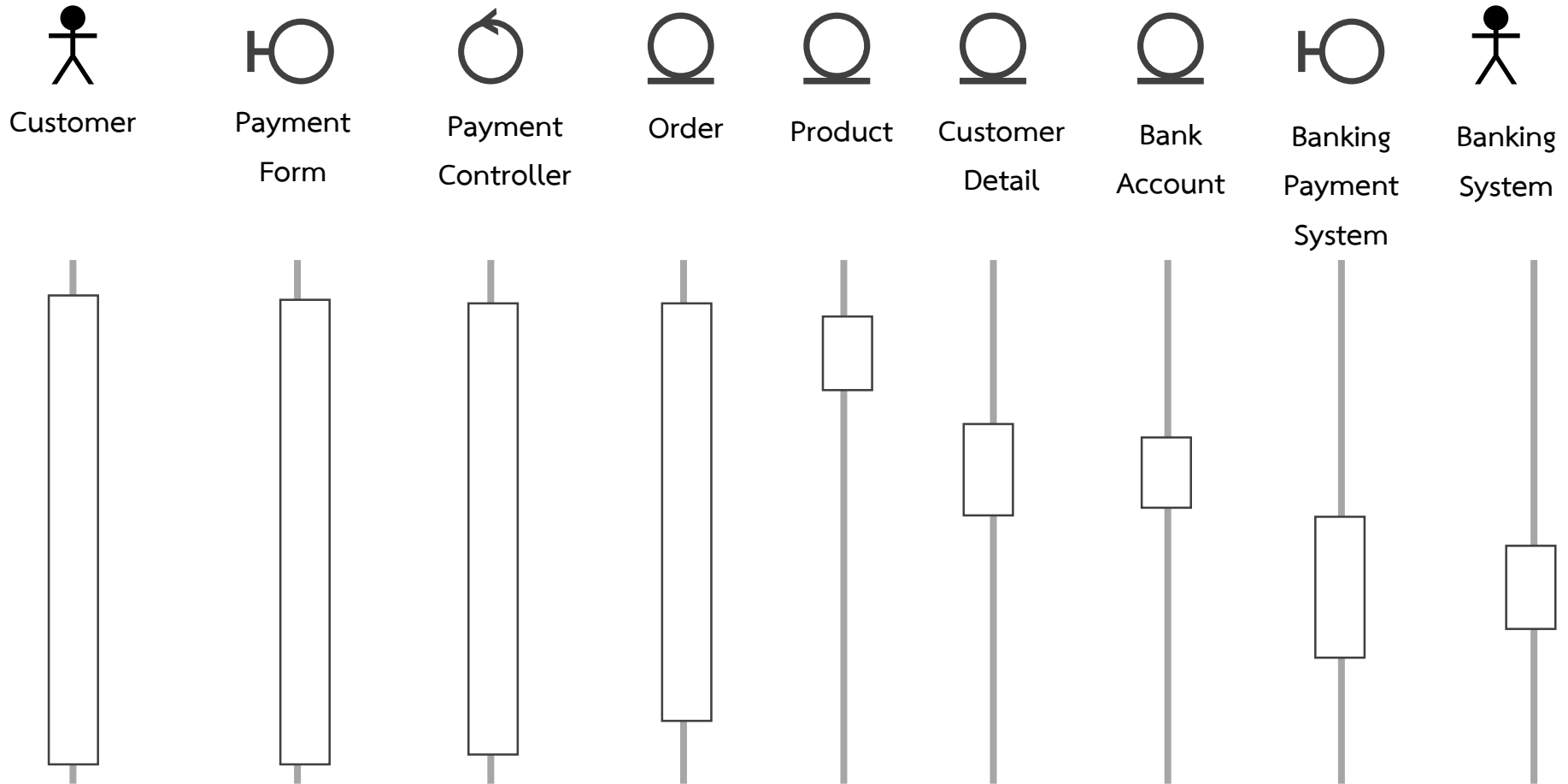
Sequence Diagram

- มี 1 Sequence Diagram ต่อ 1 Use Case
- เขียนจาก Basic Flow เป็นหลัก
- มี alternative flow ใส่ในกล่องเงื่อนไข
- มีทุก elements จาก Collaboration Diagram ของ Use Case นั้น
- มี life time ของแต่ละ element
- มีลูกศรเส้นทึบและข้อความของทุก action ของ collaboration diagram นั้น (สามารถเพิ่มเติมได้ตามความจำเป็น)
- มีลูกศรเส้นประย้อนกลับ เพื่อเป็นค่า return ของ action ได้
- ใส่ค่าให้กับ parameter ของแต่ละ action
เช่น makePayment เป็น makePayment(**OrderID**, **CustomerID**) เป็นต้น

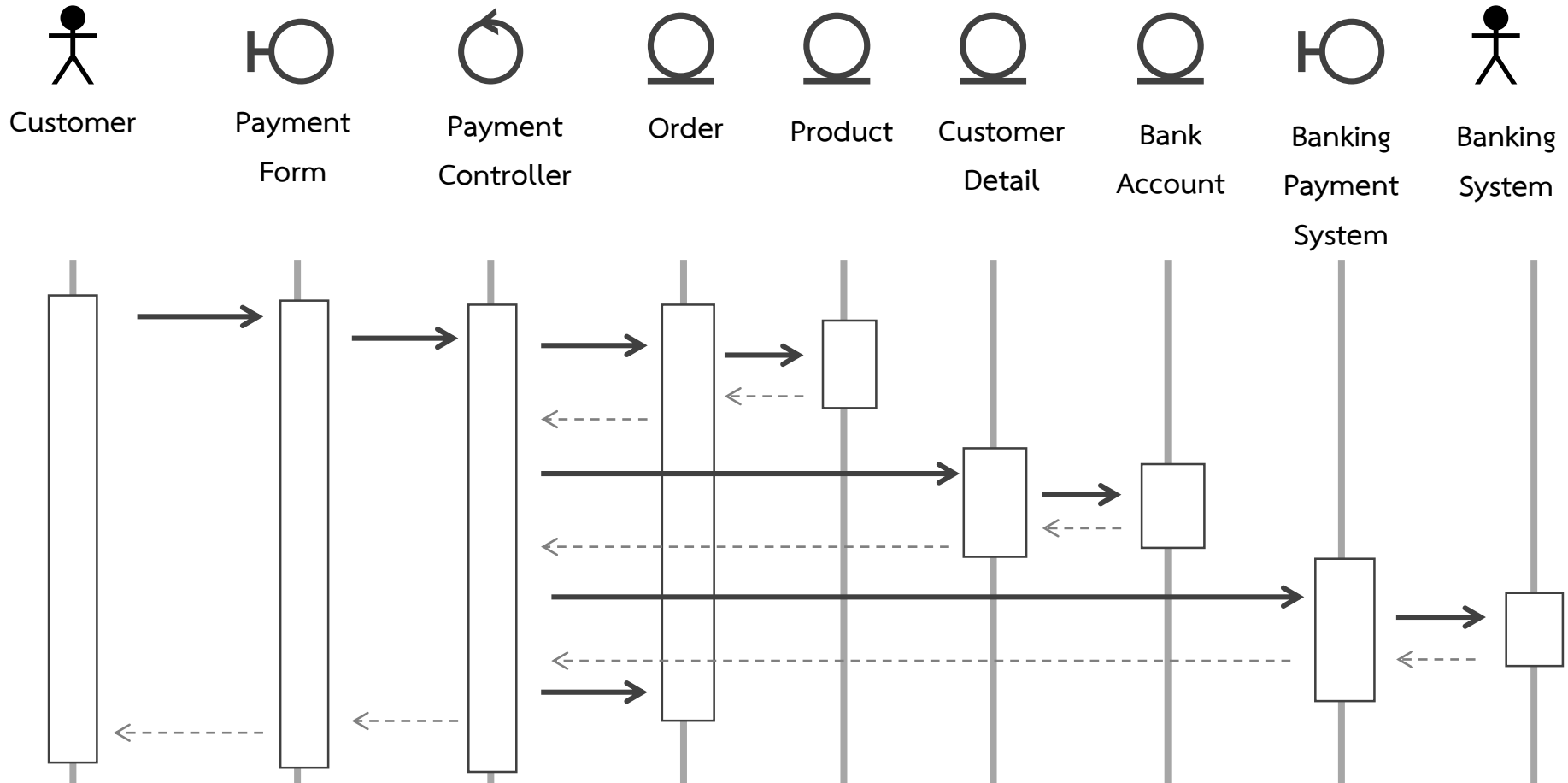
Sequence Diagram: Payment



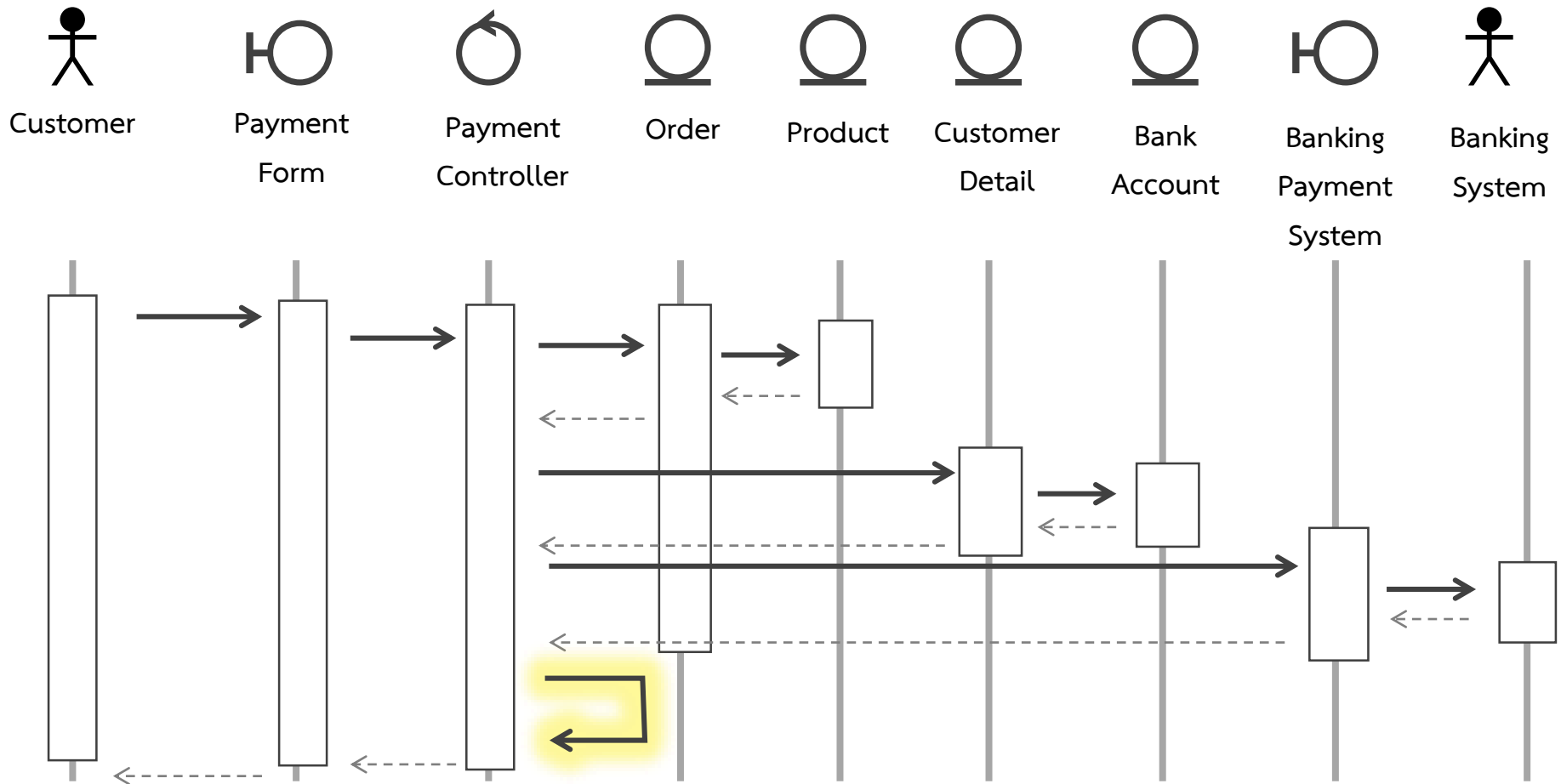
Sequence Diagram: Payment



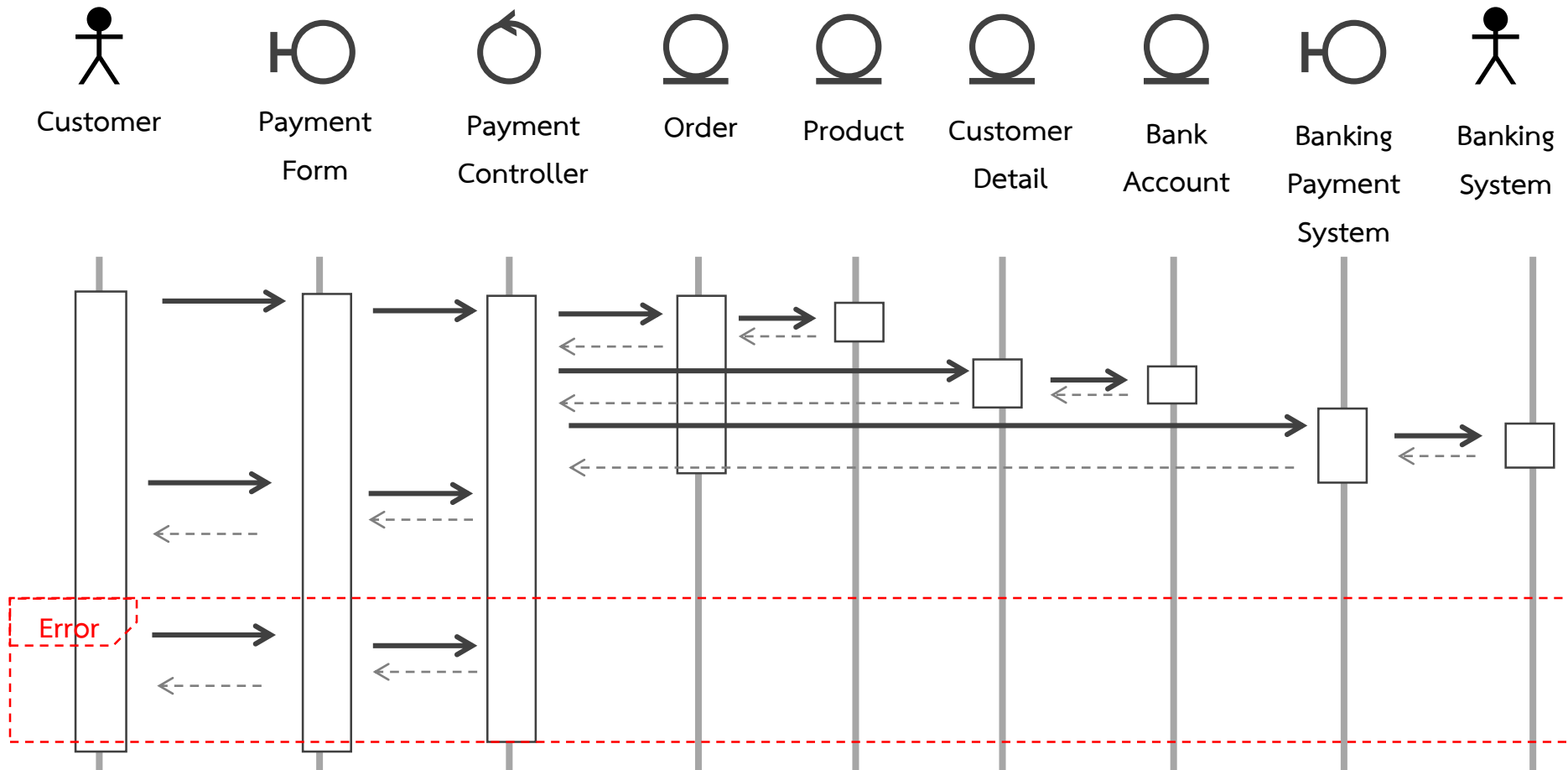
Sequence Diagram: Payment



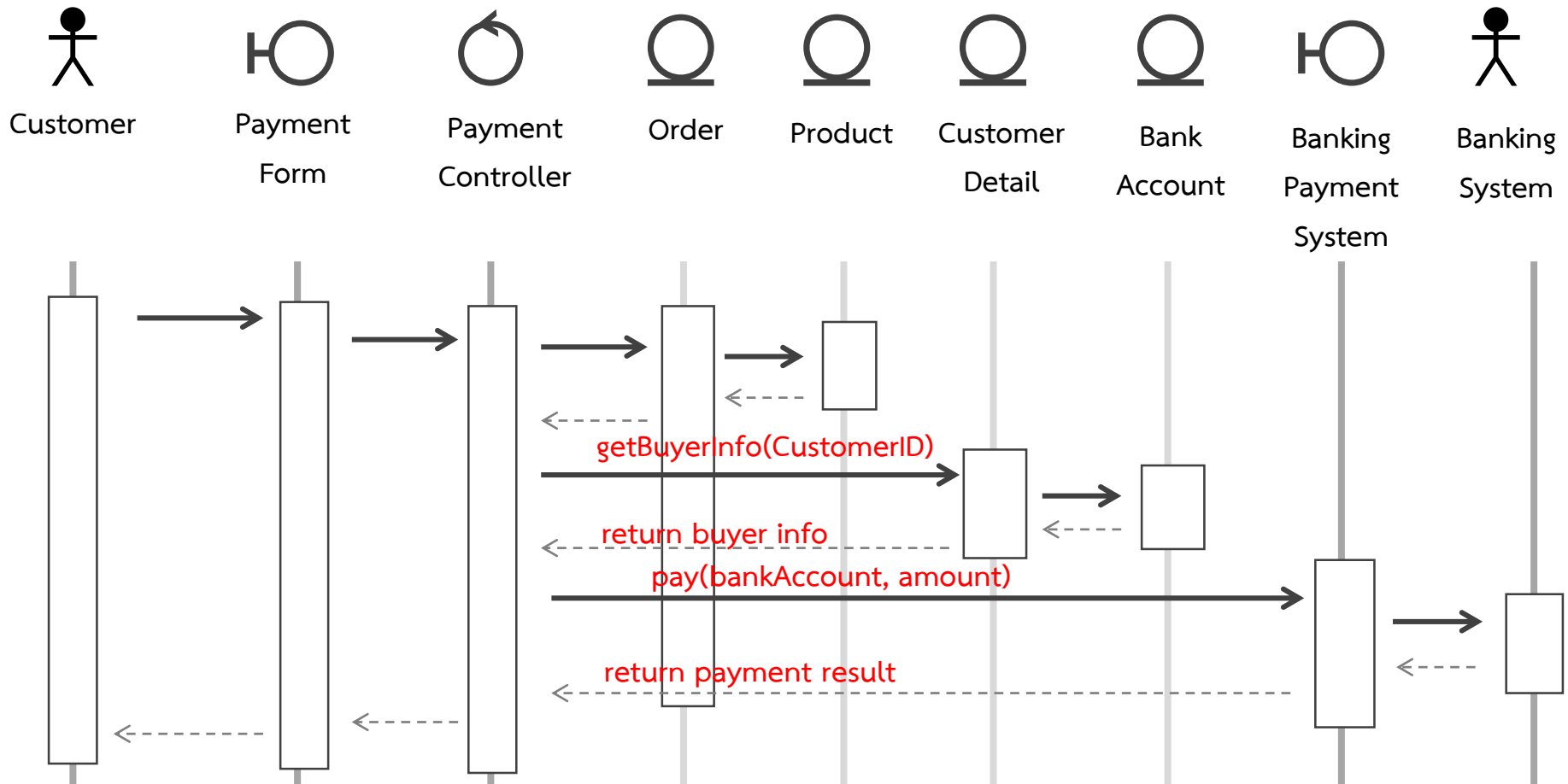
Sequence Diagram: Payment



Sequence Diagram: Payment



Sequence Diagram: Payment



Class Diagram



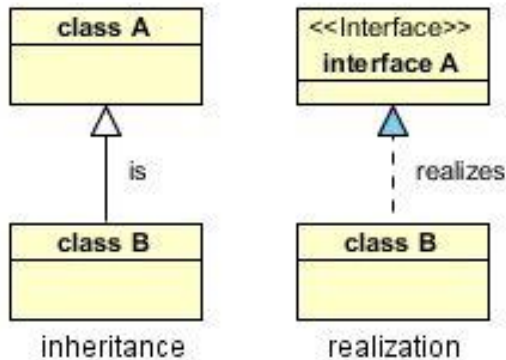
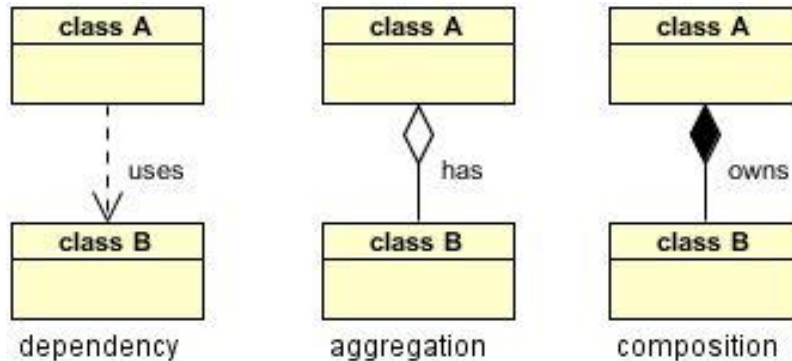
Class

- แต่ละ element ต้องเป็น class

Class Name <<ประเภท>>
<ul style="list-style-type: none">• attribute• attribute• attribute
<ul style="list-style-type: none">• constructor• method• method• method• destructor

Order <<entity>>
<ul style="list-style-type: none">• Product products []
<ul style="list-style-type: none">• Order(orderID)• Product GetProducts()• Float GetAmount()

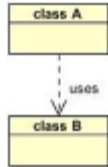
Relationships



1. **Dependency** : class A uses class B
2. **Aggregation** : class A has a class B
3. **Composition** : class A owns a class B
4. **Inheritance** : class B is a Class A (or class A is extended by class B)
5. **Realization** : class B realizes Class A (or class A is realized by class B)

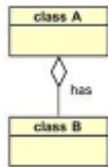
Relationships

Dependency



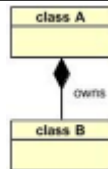
```
public class A {
    public void doSomething(B b) { ... }
```

Aggregation



```
public class A {
    private B _b;
    public void setB(B b) { _b = b; }
```

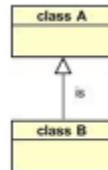
Composition



```
public class A {
    private B _b = new B();
}

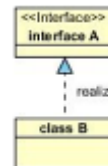
public class A {
    private B _b;
    public A() {
        _b = new B();
    }
```

Inheritance



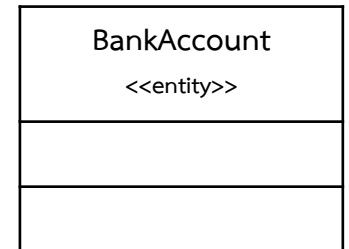
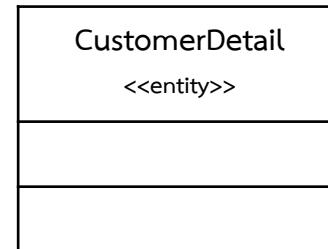
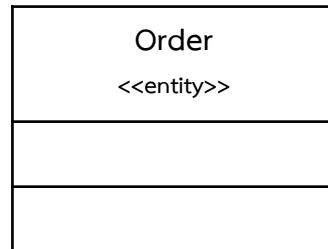
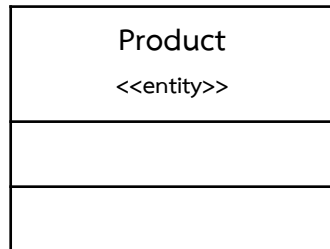
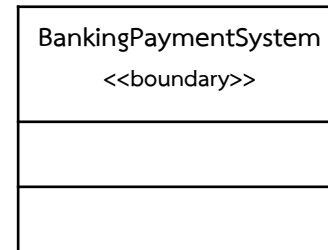
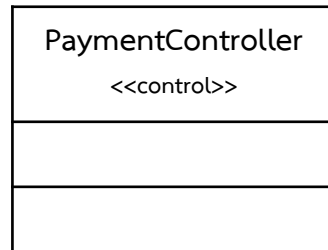
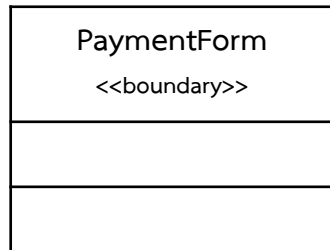
```
public class A { ... }
public class B extends A { ....}
```

Realization

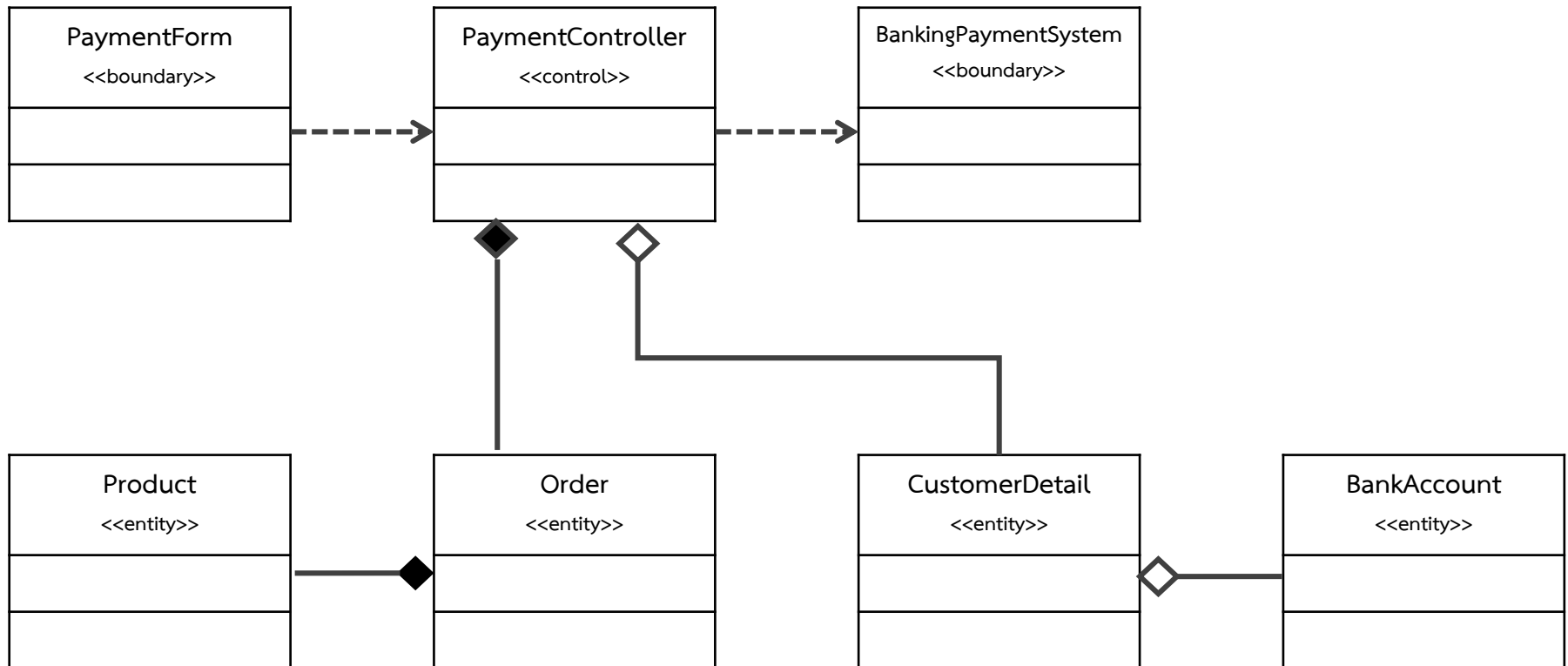


```
public interface A { ... }
public class B implements A { ....}
```

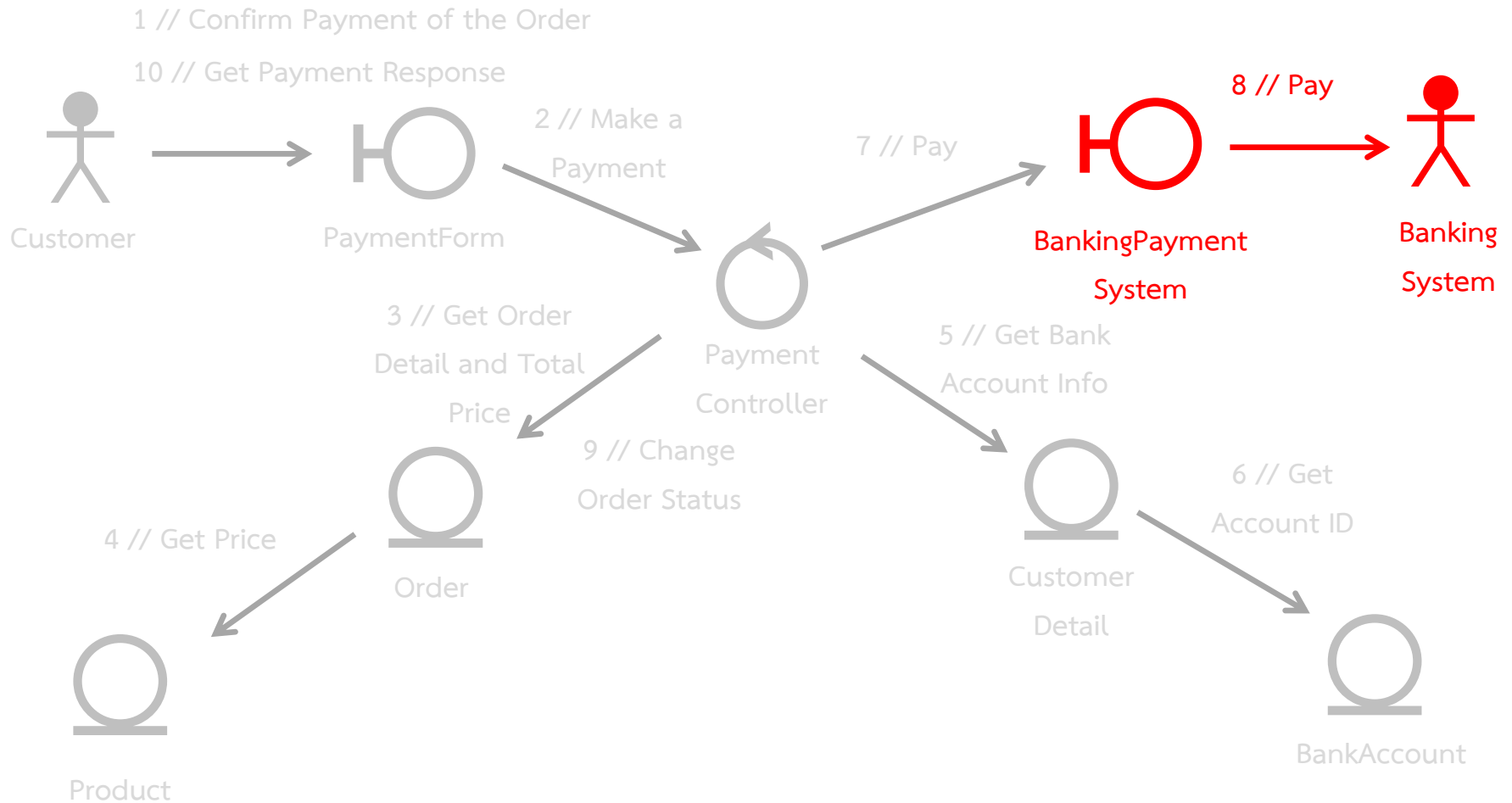
Class Diagram: Payment



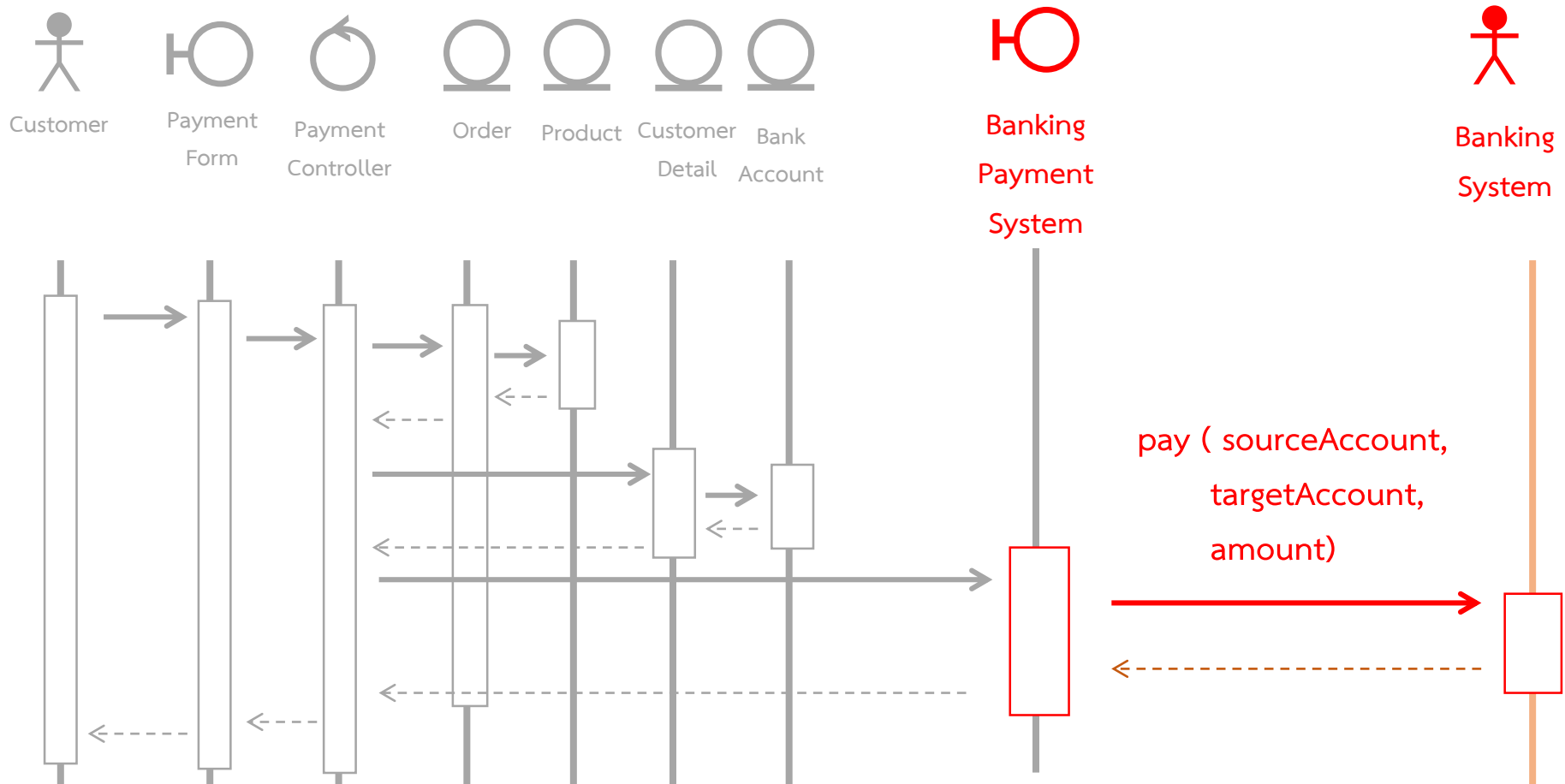
Class Diagram: Payment



Collaboration Diagram: Payment

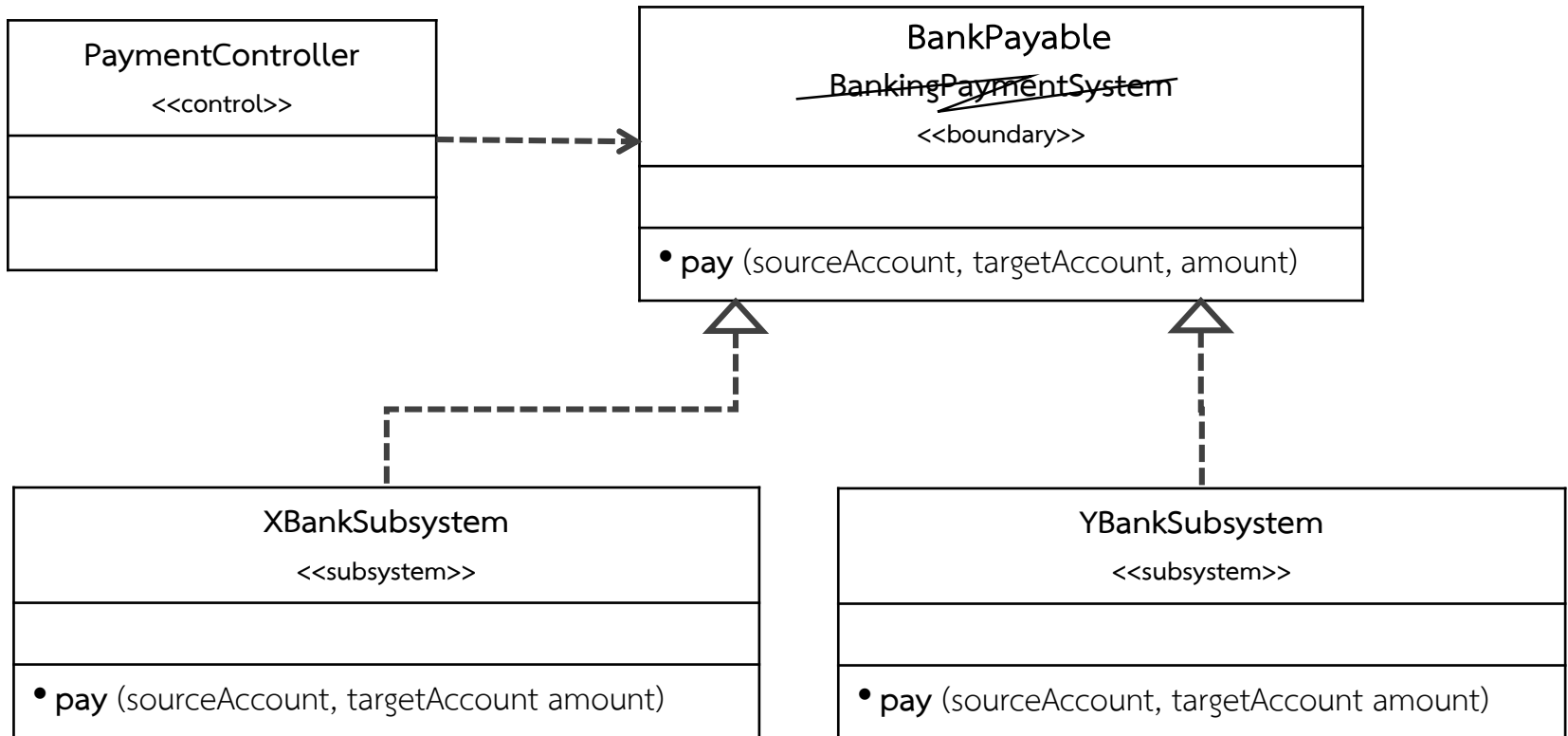


Sequence Diagram: Payment



Class Diagram: Payment

Class Realization



Example Code for Banking Payment

```
class PaymentController {
    public void run() {
        BankPayable payment = (BankPayable)beanfactory.getBean("BankService");
        payment.pay(customerBankAccountID, shopBankAccountID, totalPrice);
    }
}
```

Configuration file "spring.xml"

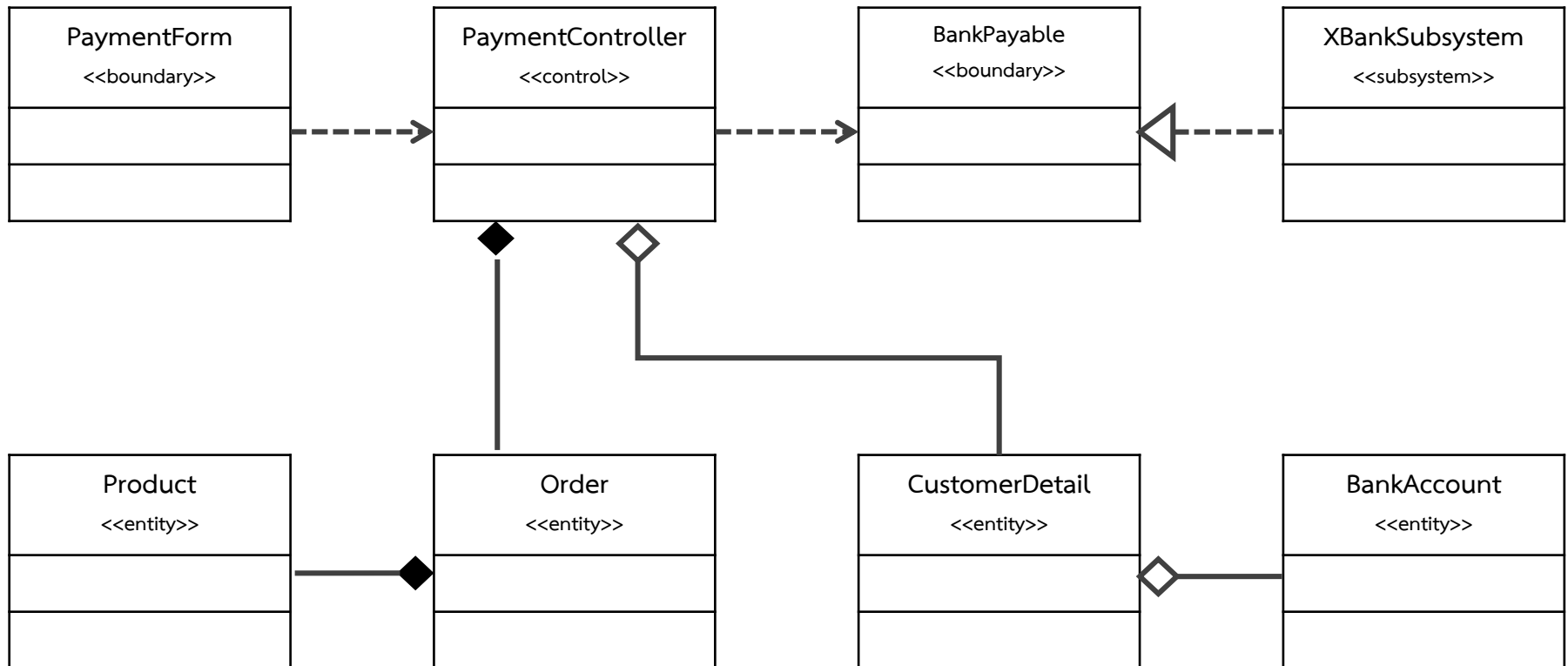
```
<bean id="BankingService" class="XBankSubsystem"></bean>
```

```
interface BankPayable {
    bool pay ( sourceAccount, targetAccount, amount);
}
```

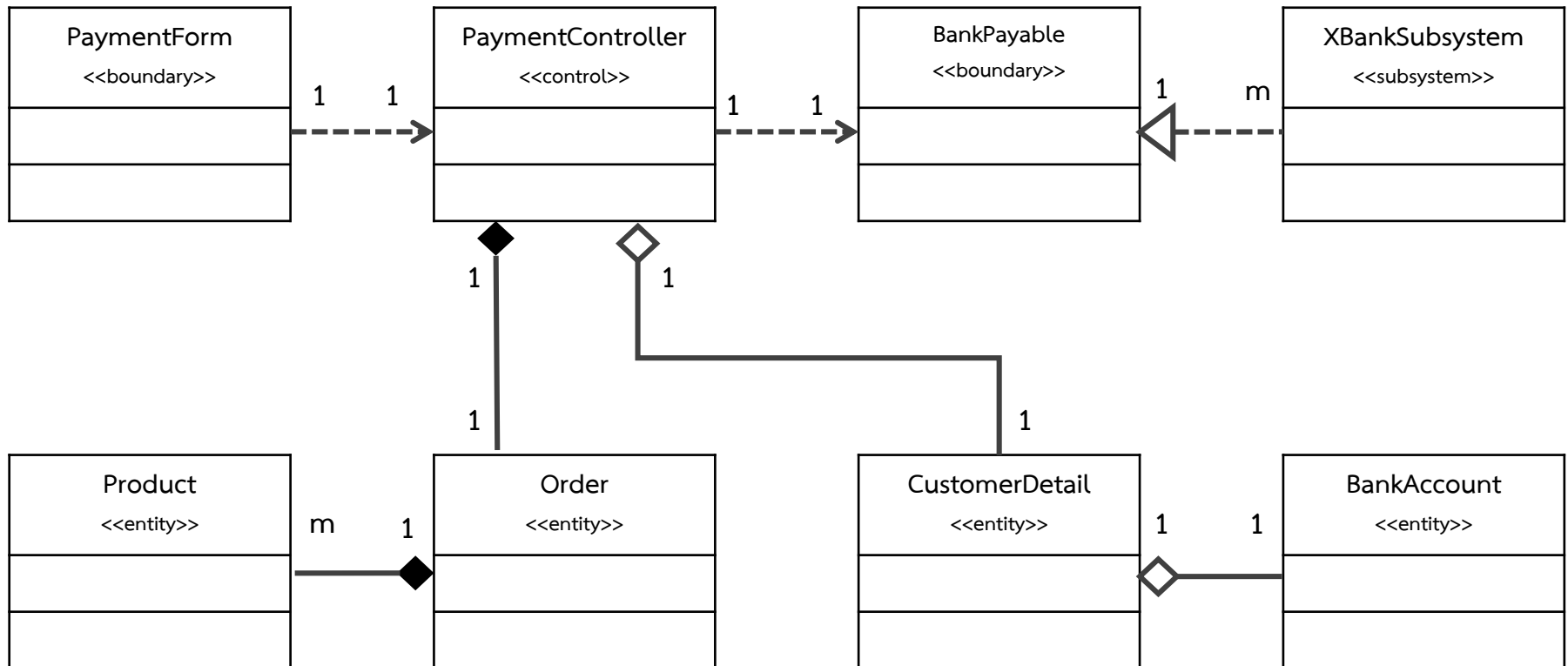
```
Class XBankSubsystem implements BankPayable {
    bool pay ( sourceAccount, targetAccount, amount){
        httpRequest("https://XBANK.com/pay?s=" + sourceAccount + "&t=" + targetAccount + "&p=" + amount );
    }
}
```

```
Class YBankSubsystem implements BankPayable {
    bool pay ( sourceAccount, targetAccount, amount){
        httpRequest("https://YBANK.com/transfer/" + sourceAccount + "/" + targetAccount + "/" + amount );
    }
}
```

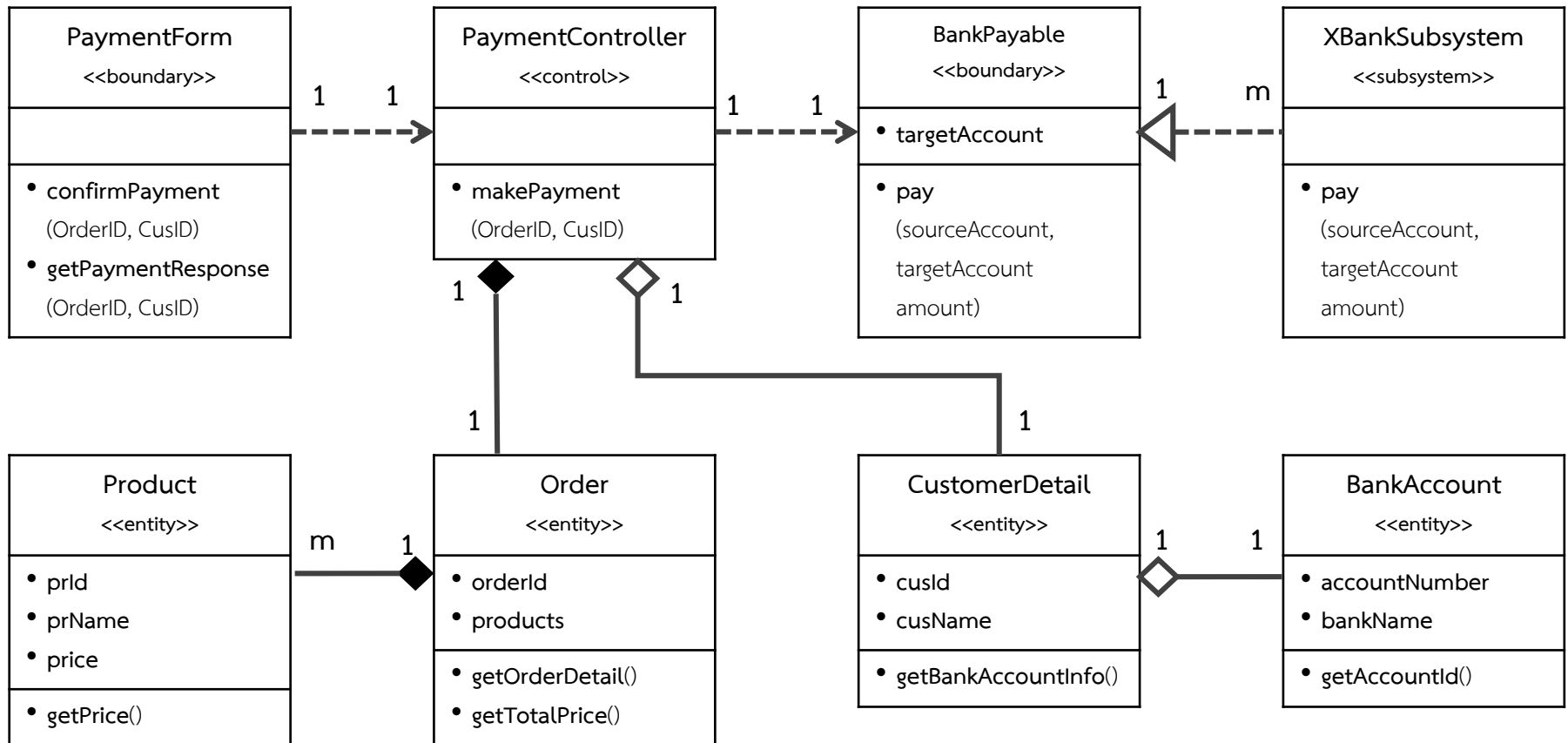
Class Diagram: Payment



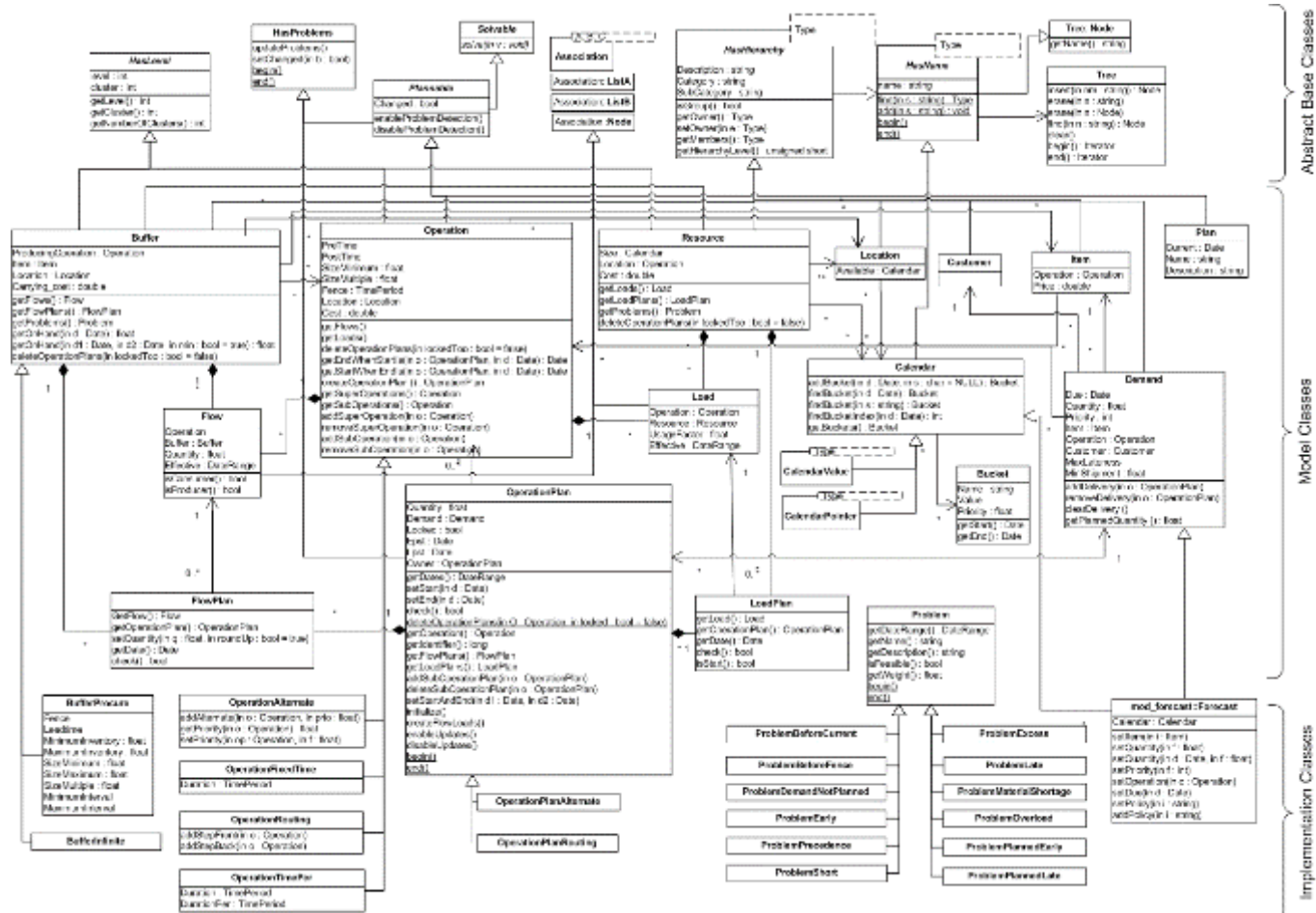
Class Diagram: Payment



Class Diagram: Payment



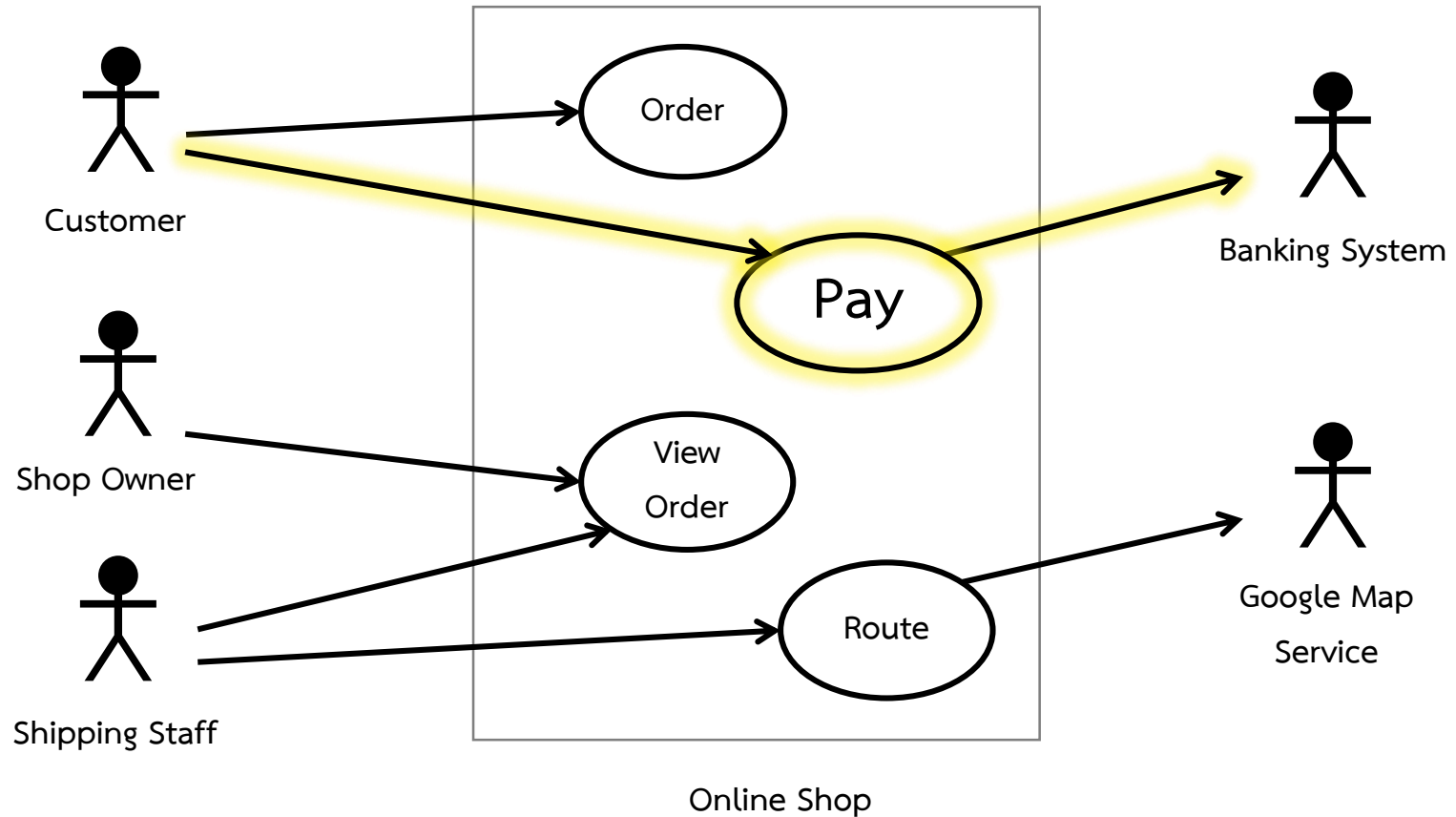
Class Diagram of a Complex System



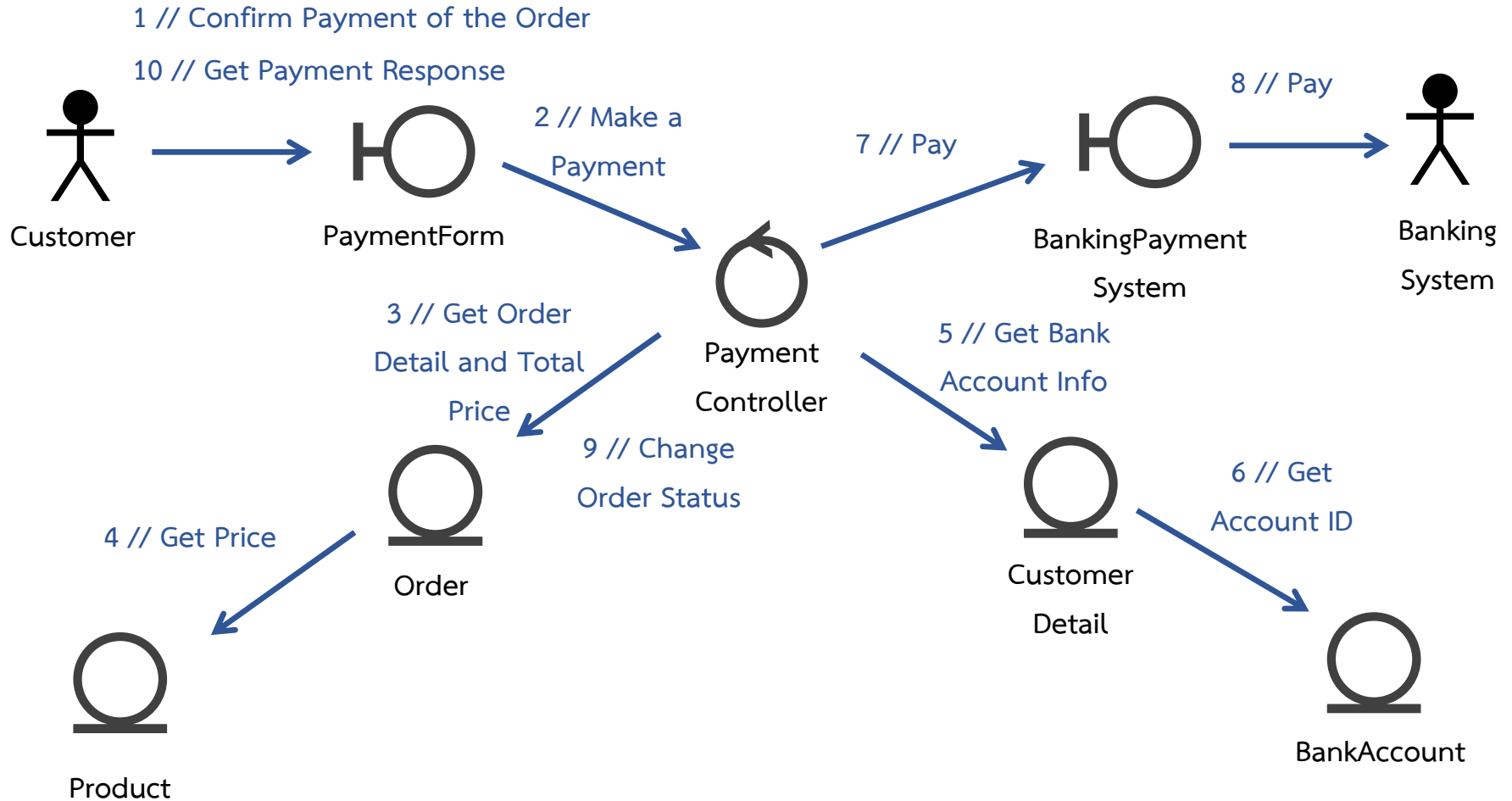
Summary



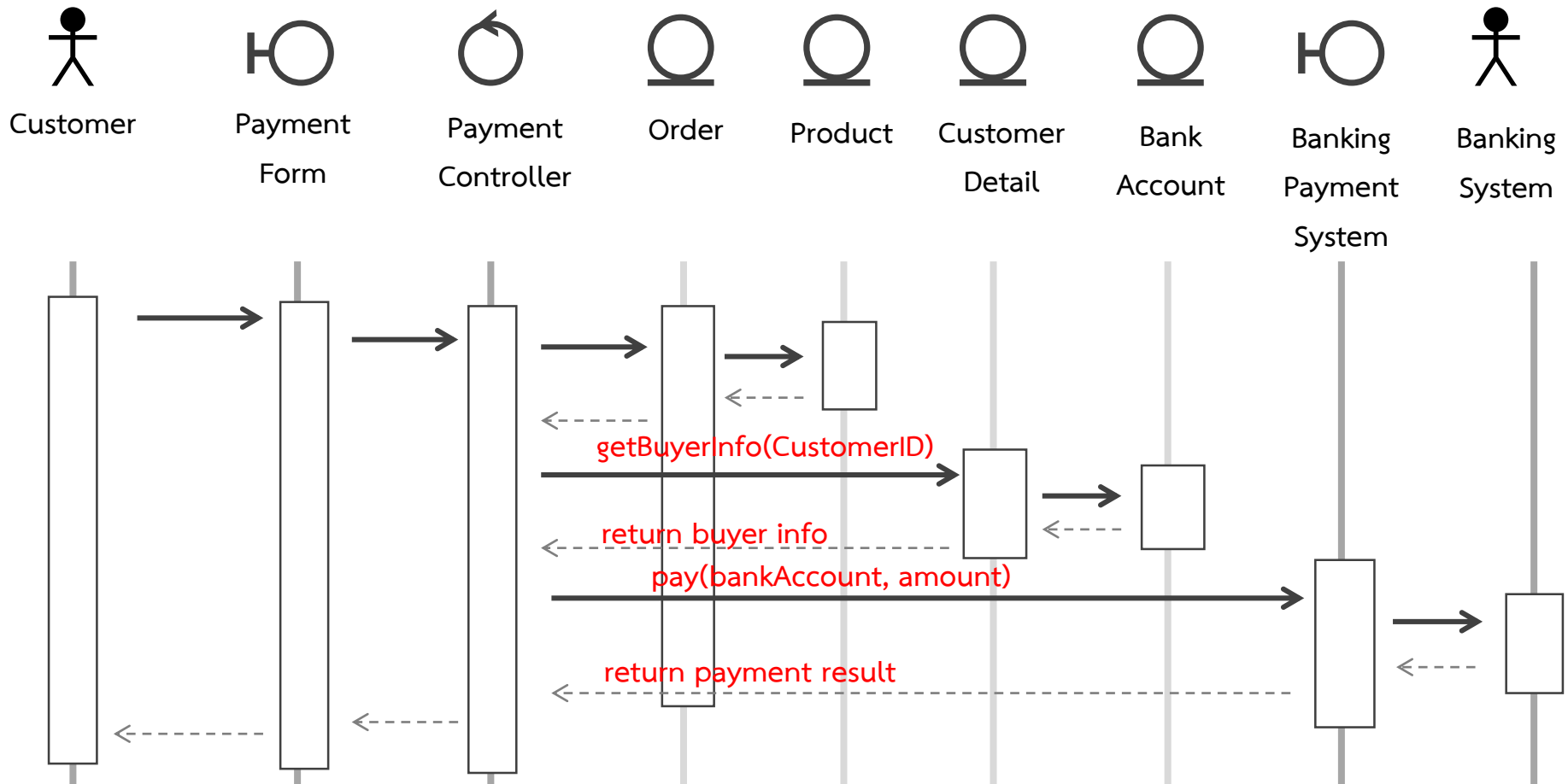
Use Case Diagram



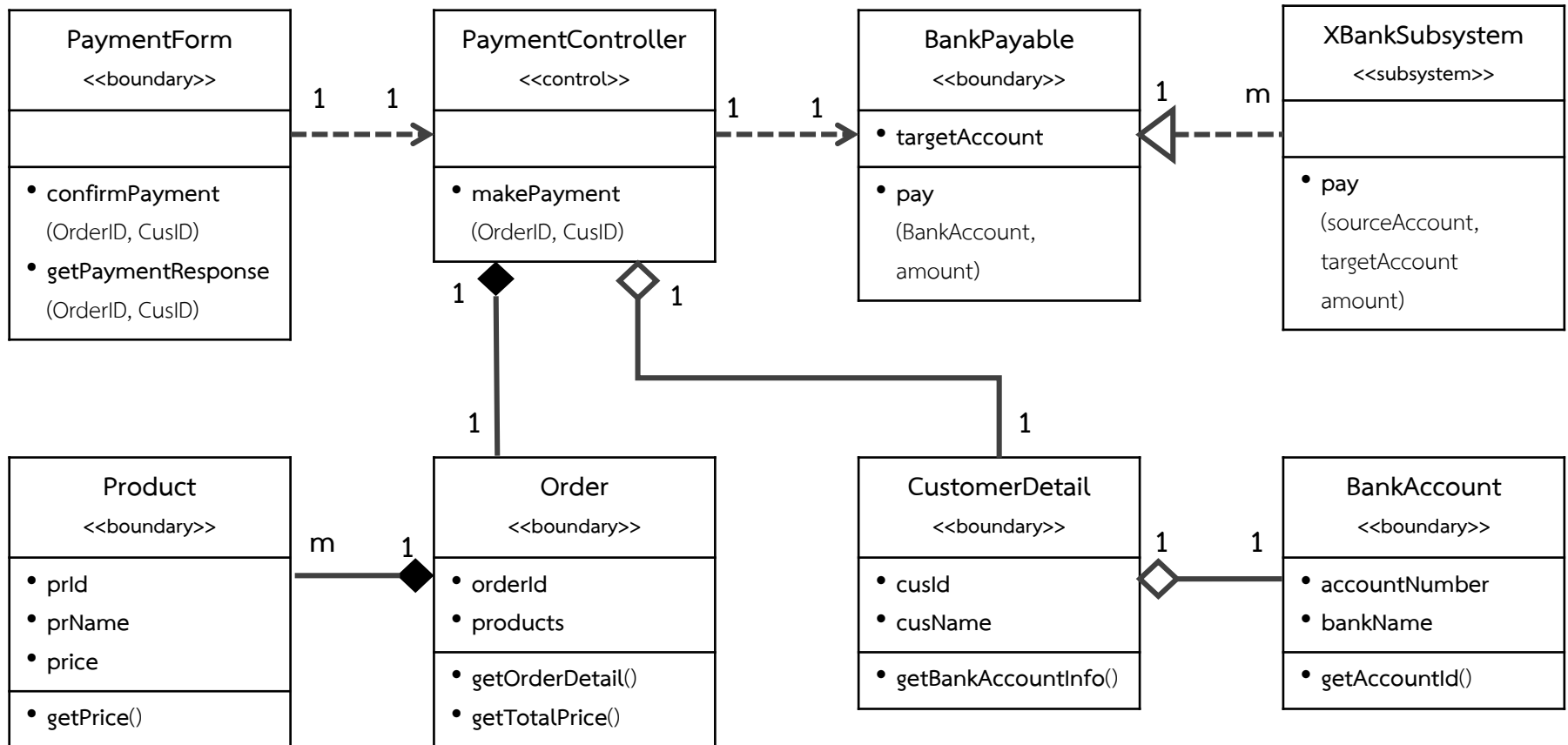
Collaboration Diagram: Payment



Sequence Diagram: Payment



Class Diagram: Payment



“

When you have a blueprint
doesn't mean
you can't add another room.

”

Tom Graves