

Design Pattern

STRATEGY

กลุ่มที่ 42

WHAT IS STRATEGY?

เป็น Design Pattern ที่มีวัตถุประสงค์เพื่อช่วยในการจัดการได้ด

และทำให้โปรแกรมมีความยืดหยุ่น ง่ายต่อการเพิ่มหรือ

เปลี่ยนแปลงพฤติกรรมต่างๆ ในโปรแกรมซึ่งทำได้โดยไม่ต้อง^{เปลี่ยนแปลง}
แก้ไขใน class หลักหลายครั้ง

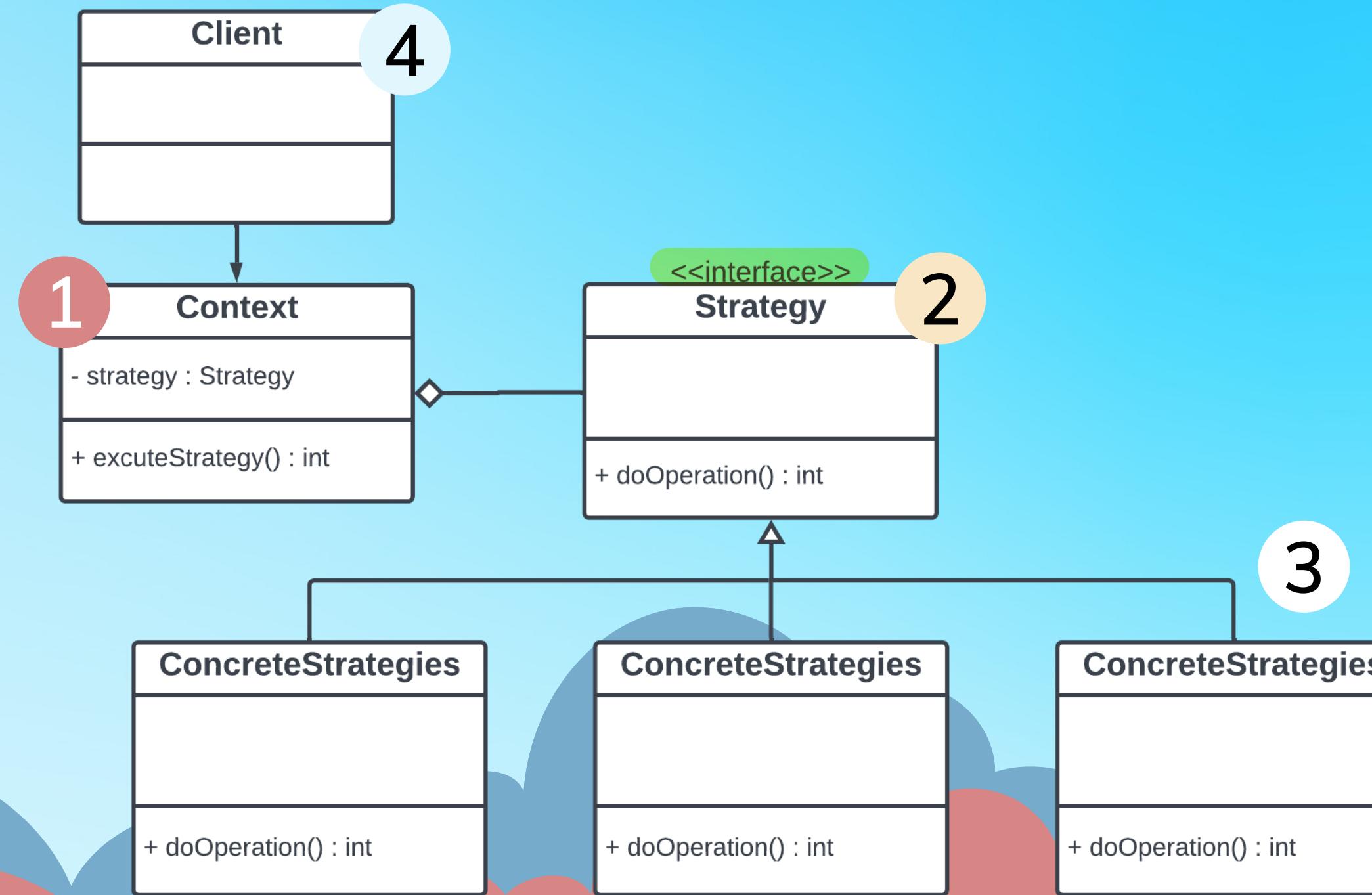


WHAT IS STRATEGY? (2)

รูปแบบนี้มักถูกใช้ในกรณีที่เราต้องการที่จะ “**แยก**” พฤติกรรม (Behavior) ของโปรแกรมออกจาก class หลัก เป็นพฤติกรรมย่อยๆ ให้ผู้ใช้ (client) เป็นคนกำหนดว่าจะสร้าง object เพื่อทำงานในเงื่อนไขหรือพฤติกรรมแบบที่ต้องการ ซึ่งรูปแบบนี้จะง่ายต่อการพัฒนาโปรแกรมเพิ่มเติมในอนาคต



COMPONENTS OF A STRATEGY



1. Context

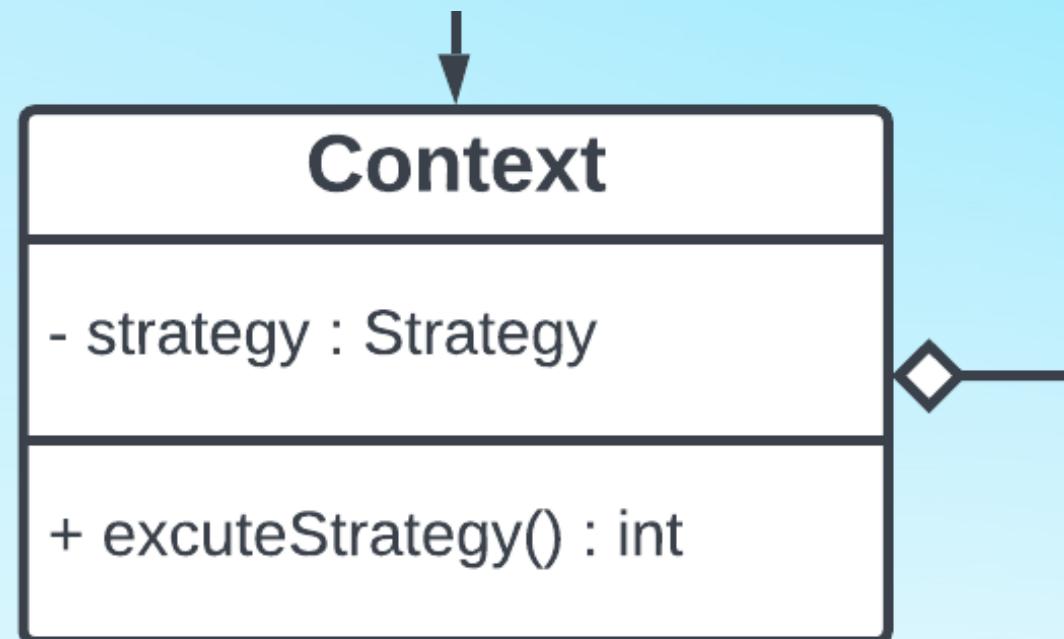
2. Strategy interface

3. Concrete Strategies

4. Client

1. CONTEXT

context คือ คลาสที่เก็บ object ของ strategy เอาไว้ โดย client(ผู้ใช้) จะเรียกใช้ strategy ผ่านทาง context ตามเงื่อนไขหรือบังทึกที่ต้องการ ซึ่ง context ไม่จำเป็น ต้องรู้ว่าใน strategy มีการทำงานอย่างไร



รูปตัวอย่างโค้ดอย่างง่าย

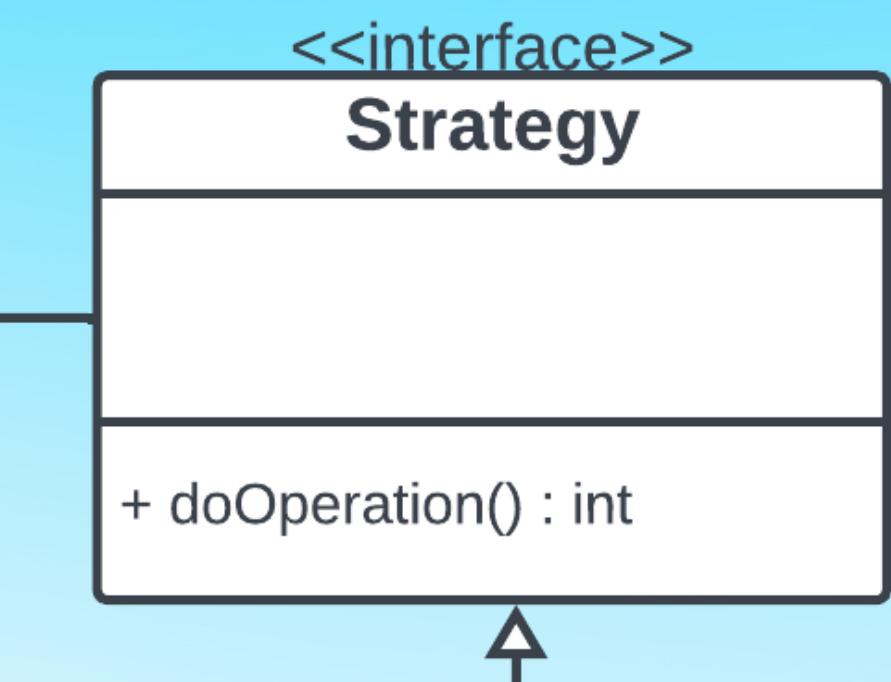
```
public class Context {  
    private Strategy strategy;  
  
    public Context(Strategy strategy){  
        this.strategy = strategy;  
    }  
  
    public int executeStrategy(int num1, int num2){  
        return strategy.doOperation(num1, num2);  
    }  
}
```

2. STRATEGY INTERFACE

เป็นแม่แบบให้กับ Concrete Strategies หรือ strategy ย่อยๆ ซึ่งมี method เพื่อให้ context สามารถเรียกใช้งานได้ และ method นี้จะถูก implement ใน Concrete Strategies

รูปตัวอย่างโค้ดอย่างง่าย

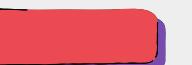
```
public interface Strategy {  
    public int doOperation(int num1, int num2);  
}
```



```
public class OperationAdd implements Strategy{  
    @Override  
    public int doOperation(int num1, int num2) {  
        return num1 + num2;  
    }  
}
```



```
public class OperationSubtract implements Strategy{  
    @Override  
    public int doOperation(int num1, int num2) {  
        return num1 - num2;  
    }  
}
```



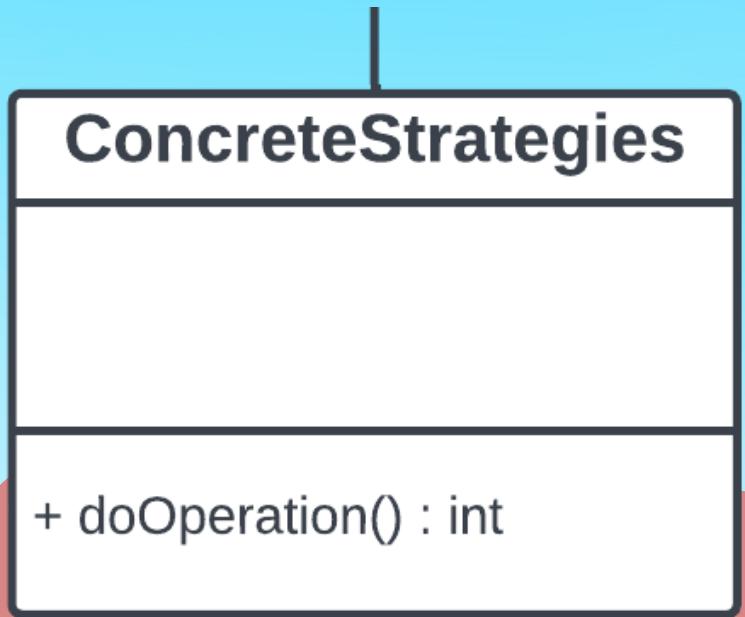
```
public class OperationMultiply implements Strategy{  
    @Override  
    public int doOperation(int num1, int num2) {  
        return num1 * num2;  
    }  
}
```



รูปตัวอย่างโค้ดอย่างง่าย

3.CONCRETE STRATEGIES

คือตัวที่ทำงานจริงๆ ซึ่งส่วนใหญ่มักมี concrete strategy หลายตัว เพื่อให้สามารถทำงานได้หลายแบบแตกต่างกันตามความต้องการหรือบริบท



4.CLIENT

เป็นคลาสที่ส่ง strategy ที่จะใช้ให้กับ context

รูปตัวอย่างโค้ดอย่างง่าย

```
public class StrategyPatternDemo {  
    public static void main(String[] args) {  
        Context context = new Context(new OperationAdd());  
        System.out.println("10 + 5 = " + context.executeStrategy(10, 5));  
  
        context = new Context(new OperationSubstract());  
        System.out.println("10 - 5 = " + context.executeStrategy(10, 5));  
  
        context = new Context(new OperationMultiply());  
        System.out.println("10 * 5 = " + context.executeStrategy(10, 5));  
    }  
}
```

Result

$10 + 5 = 15$

$10 - 5 = 5$

$10 * 5 = 50$

»» เทมาศก์บงานประโภตใจ

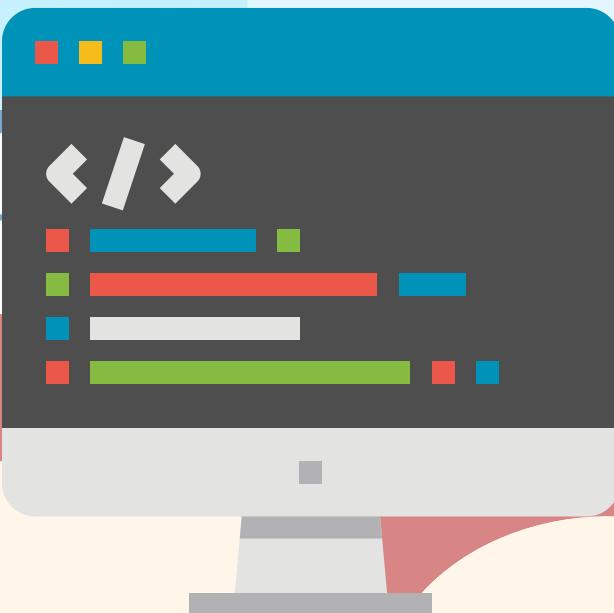
ເໜາະລຳຮັບການໃຊ້ງານໃນສານກາຣົນທີ່ມີກາຣດິດດຳນວນຫຼື
ພຸຕິກຣມທີ່ຕ້ອງການໃໝ່ຢືດຫຍຸ່ນແລລສາມາຣົບປັ້ງແປລງໄດ້ອ່າງ
ຈ່າຍໂດຍໄມ່ຕ້ອງແກ້ໄຂໄດ້ໜັກຂອງໂປຣແກຣມ ເຊັ່ນ

- ກາຣຈັດກາຣກັບບັນຕອນກາຣດຳນວນທີ່ມີຄວາມແຕກຕ່າງກັນ
- ກາຣຈັດກາຣກັບເງື່ອນໄຟທີ່ມີໜາຍຮູປແບບ
- ກາຣທຳໜ້າທີ່ດ້າຍກັນແຕ່ຕ້ອງກາຣພຸຕິກຣມແຕກຕ່າງກັນ



>> ทำไม่ถึงอยู่ใน CATEGORY BEHAVIORAL

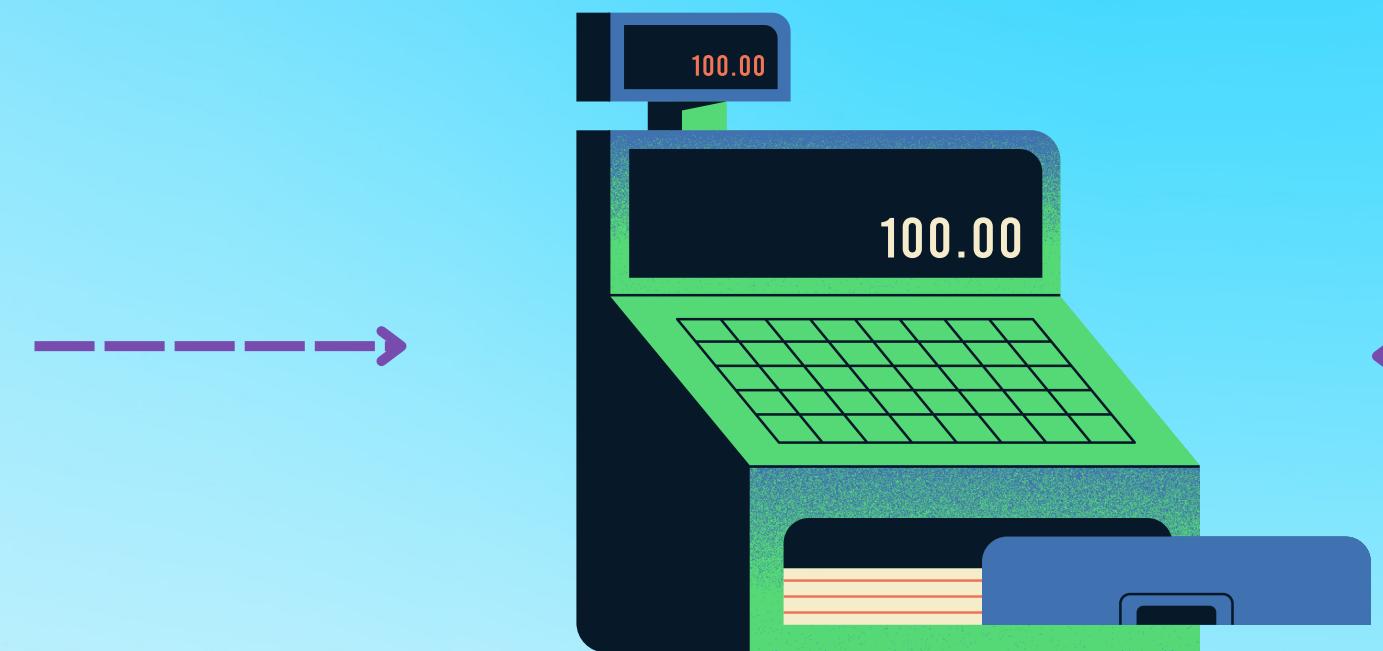
เนื่องจาก Strategy เกี่ยวข้องกับวิธีที่ออบเจกต์และคลาสทำงานร่วมกันและปรับ **พฤติกรรม** ของตนเองตามเงื่อนไขหรือสถานการณ์ต่างๆ โดยที่ไม่ต้องแก้ไขโค้ดในคลาสเหล็ก (Behavioral Design Patterns มีจุดประสงค์ในการจัดการและควบคุมพฤติกรรมและความล้มเหลวของออบเจกต์และคลาสในโปรแกรม)



ณ ร้านขายของชำแห่งหนึ่ง

ภาพการทำงานของระบบ

มีการออกแบบระบบการจ่ายเงินด้วยบอร์ดโดยจ่ายด้วยเงินสด



ຕົວອຍ່າງໂຄດ ສ້າງເນື້ອໃຫ້ Strategy

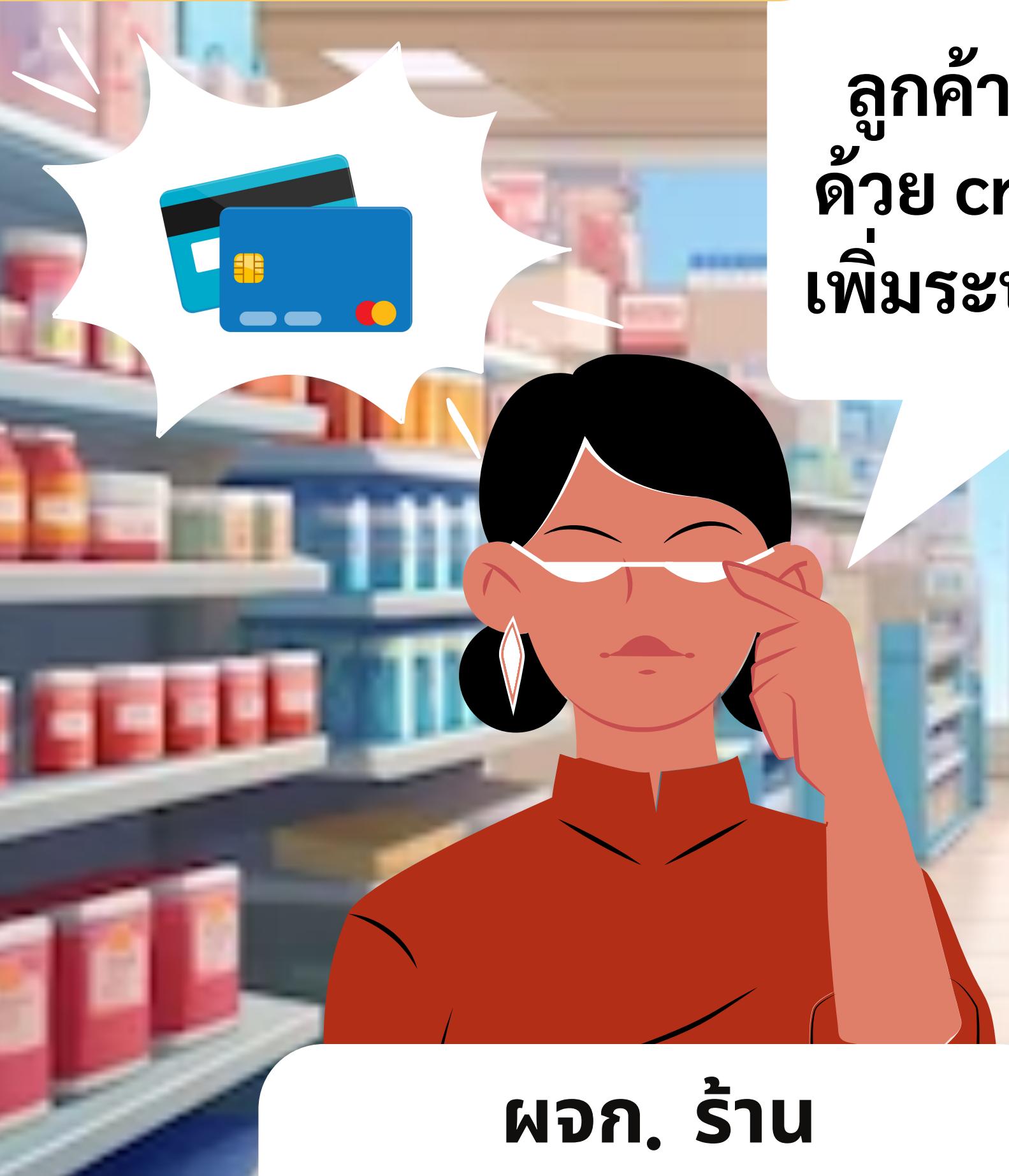
```
class Item {  
  
    private String name;  
    private double price;  
  
    public Item(String name, double price) {  
        this.name = name;  
        this.price = price;  
    }  
  
    public double getPrice() {  
        return price;  
    }  
}
```

```
public class Client{  
    public static void main(String[] args){  
        ShoppingCart cart = new ShoppingCart();  
  
        Item item1 = new Item("ABC1234", 100.50);  
        Item item2 = new Item("ABC2345", 50);  
  
        cart.addItem(item1);  
        cart.addItem(item2);  
  
        cart.pay();  
    }  
}
```

```
import java.util.List;  
import java.util.ArrayList;  
  
public class ShoppingCart {  
    private List<Item> items;  
    private Payment payment;  
  
    public ShoppingCart() {  
        items = new ArrayList<>();  
        payment = new Payment();  
    }  
  
    public void addItem(Item item) {  
        items.add(item);  
    }  
  
    public void removeItem(Item item) {  
        items.remove(item);  
    }  
  
    public double totalPrice() {  
        return items.stream()  
            .mapToDouble(Item::getPrice)  
            .sum();  
    }  
  
    public void pay() {  
        double amount = totalPrice();  
        payment.pay(amount);  
    }  
}
```

```
public class Payment {  
    public void pay(double amount) {  
        System.out.println(amount + " paid with cash");  
    }  
}
```

ณ ร้านขายของชำแห่งหนึ่ง



ผจก. ร้าน

ลูกค้าต้องการจ่าย
ด้วย credit card ไป
เพิ่มระบบมาหน่อยสิ!



โปรแกรมเมอร์

ຕົວອຍ່າງໂຄດ ສໍາເຮັນໃຫ້ Strategy



```
public class Payment {  
    public void pay(double amount) {  
        System.out.println(amount + " paid with cash");  
    }  
}
```

```
public class Payment {  
    private Scanner in = new Scanner(System.in);  
    private String type = "cash";  
    public void setType(String type) {  
        this.type = type;  
    }  
    public void pay(double amount) {  
        if(type.equals("credit_card")) {  
            System.out.println("Enter Card number: ");  
            String number = in.nextLine();  
            System.out.println("Enter expiration date mm/yy: ");  
            String date = in.nextLine();  
            System.out.println("Enter CVV code: ");  
            String cvv = in.nextLine();  
            CreditCard card = new CreditCard(number, date, cvv);  
            if(card.validate()) {  
                card.setAmount(amount);  
                System.out.println("Paid with " + card);  
            } else {  
                System.out.println("Payment Credit Card failed");  
            }  
        } else {  
            System.out.println("paid with cash " + amount);  
        }  
    }  
}
```

“NEW”

วันต่อมา...



เพิ่มระบบ
พร้อมเพย์มาด้วย



ผจก. ร้าน

โปรแกรมเมอร์

ครับผม



ຂາຍໆວັນຕ່ອມາ...



ເພີ່ມ
ຮະບບPaypal
ມາດ້ວຍ

ຮະບບນູ່ນີ້ນັ້ນ
ດ້ວຍນະ

ຜັກ. ຮ້ານ

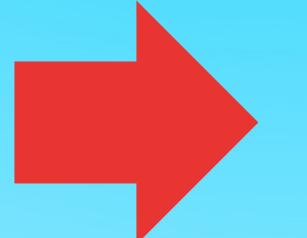
ໂປຣແກຣມເມອ່ວ

ຄຣັບ...

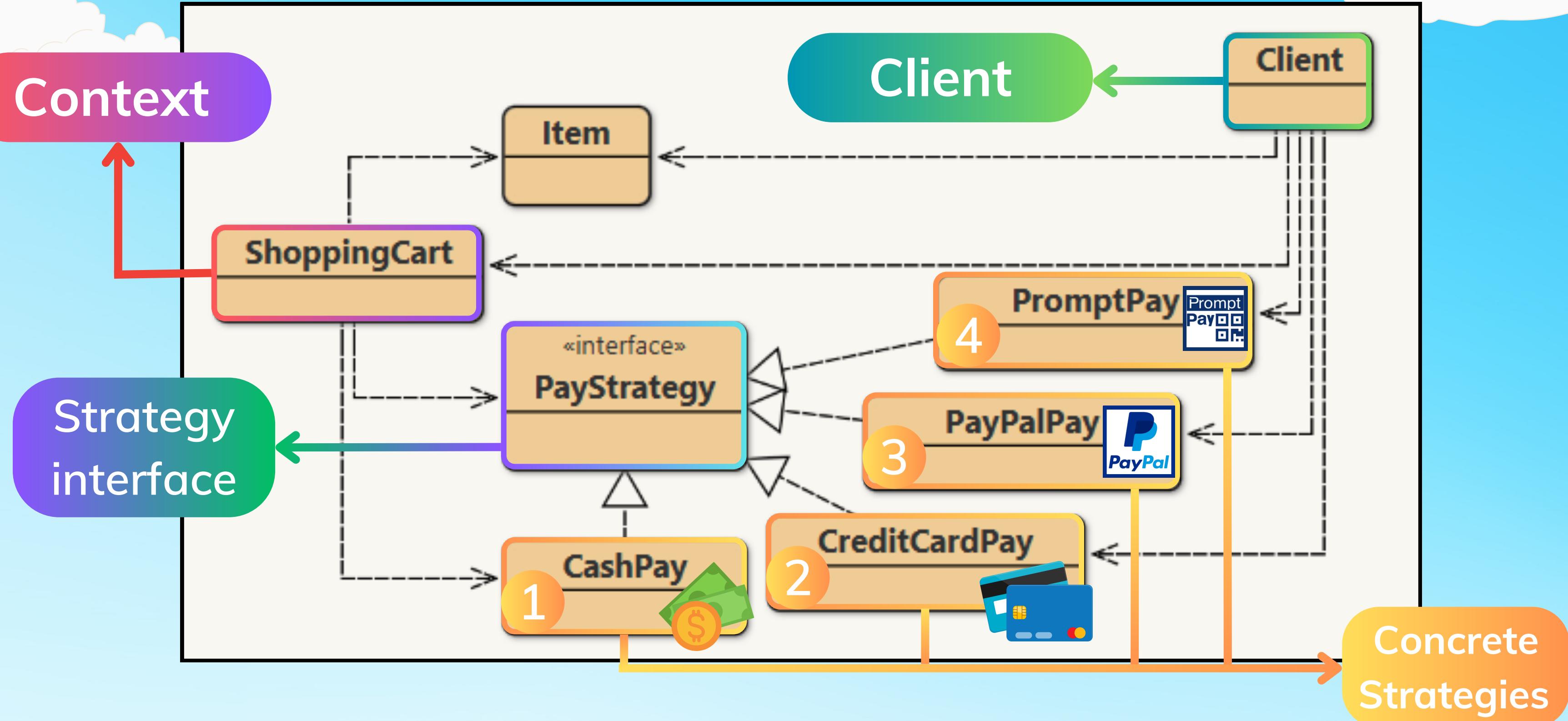
ຊັບເໜື່ອຍາກນະ

ຕັ້ງອຍ່າງໂຄດ ລ້າເຮົາໃນໃຊ້ Strategy

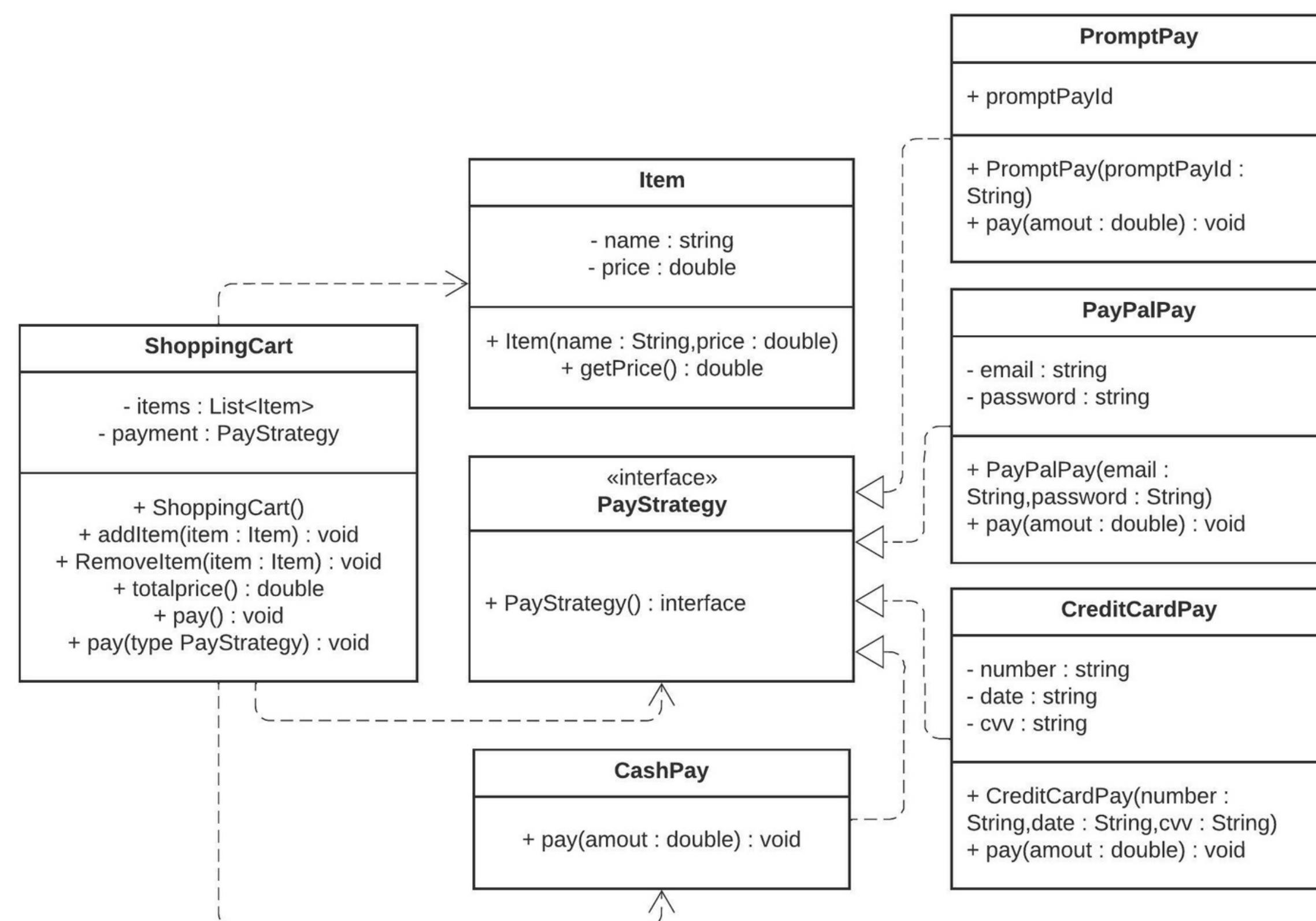
```
public class Payment {  
    private Scanner in = new Scanner(System.in);  
    private String type = "cash";  
    public void setType(String type) {  
        this.type = type;  
    }  
    public void pay(double amount) {  
        if(type.equals("credit_card")) {  
            System.out.println("Enter Card number: ");  
            String number = in.nextLine();  
            System.out.println("Enter expiration date mm/yy: ");  
            String date = in.nextLine();  
            System.out.println("Enter CVV code: ");  
            String cvv = in.nextLine();  
            CreditCard card = new CreditCard(number, date, cvv);  
            if(card.validate()) {  
                card.setAmount(amount);  
                System.out.println("Paid with " + card);  
            } else {  
                System.out.println("Payment Credit Card failed");  
            }  
        } else {  
            System.out.println("paid with cash " + amount);  
        }  
    }  
}
```



CLASS DIAGRAM FORM BLUEJ



CLASS DIAGRAM



ตัวอย่างโค้ด USE STRATEGY

Strategy
interface

Context

```
import java.util.List;
import java.util.ArrayList;

class ShoppingCart {
    private List<Item> items;
    private PayStrategy payment;

    public ShoppingCart() {
        items = new ArrayList<>();
        payment = new CashPay();
    }
    public void addItem(Item item) {
        items.add(item);
    }
    public void removeItem(Item item) {
        items.remove(item);
    }
}
```

```
public double totalPrice() {
    return items.stream()
        .mapToDouble(Item::getPrice)
        .sum();
}
public void pay() {
    double amount = totalPrice();
    payment.pay(amount);
}
public void pay(PayStrategy type) {
    double amount = totalPrice();
    payment = type;
    payment.pay(amount);
}
```

```
public interface PayStrategy {
    void pay(double amount);
}
```



ຕົວອຍ່າງໂຄດ USE STRATEGY

Concrete Strategies

```
public class CashPay implements PayStrategy {  
    @Override  
    public void pay(double amount) {  
        System.out.println("paid with cash amount: " + amount);  
    }  
}
```

```
public class CreditCardPay implements PayStrategy {  
    private String number;  
    private String date;  
    private String cvv;  
  
    public CreditCardPay(String number, String date, String cvv){  
        this.number = number;  
        this.date = date;  
        this.cvv = cvv;  
    }  
  
    public void pay(double amount) {  
        System.out.println("paid with Credit card amount: " + amount);  
    }  
}
```



1



2

```
public class PayPalPay implements PayStrategy {  
    private String email;  
    private String password;  
  
    public PayPalPay(String email, String password){  
        this.email = email;  
        this.password = password;  
    }  
  
    public void pay(double amount) {  
        System.out.println("paid with PayPal amount: " + amount);  
        System.out.println("Email: " + email);  
    }  
}
```



3

```
public class PromptPay implements PayStrategy {  
    String promptPayId;  
  
    public PromptPay(String promptPayId){  
        this.promptPayId = promptPayId;  
    }  
  
    public void pay(double amount) {  
        System.out.println("paid with PromptPay amount: " + amount);  
        System.out.println("PromptPay ID: " + promptPayId);  
    }  
}
```



4



ຕົວຢ່າງໂຄດ

USE STRATEGY

Client

```
public class Client
{
    public static void test()
    {
        ShoppingCart cart = new ShoppingCart();

        Item item1 = new Item("ABC1234", 100.50);
        Item item2 = new Item("ABC2345", 50);

        cart.addItem(item1);
        cart.addItem(item2);

         cart.pay(); //cart.pay(new CashPay());
        System.out.println("=====");
        cart.pay( new CreditCardPay("123456789", "12/24", "999"));
        System.out.println("=====");
        cart.pay( new PayPalPay("client_mail@gmail.com", "clientPassword"));
        System.out.println("=====");
        cart.pay( new PromptPay("123456"));
    }
}
```

Result



paid with cash amount: 150.5



paid with Credit card amount: 150.5



paid with PayPal amount: 150.5



paid with PromptPay amount: 150.5

PromptPay ID: 123456



การการทำงานของระบบ



SUMMARY

Strategy Pattern มีจุดประสงค์หลักในการแยกพฤติกรรมและวิธีการของอุปกรณ์ออกจากโค้ดหลักของโปรแกรม และทำให้สามารถสลับหรือเปลี่ยนแปลงพฤติกรรมโดยไม่ต้องแก้ไขโค้ดหลัก โดย strategy pattern นี้ประกอบไปด้วย

1. Context : ตัวรับการทำงานจาก client ว่าจะใช้ strategy ไหน
2. Strategy interface : เป็นแม่แบบของ Concrete Strategies
3. Concrete Strategies : มีได้หลายอัน ซึ่งจะมีการอิมเพลเม้นต์ Strategy เพื่อรับบุพติกรรมที่แตกต่างกัน
4. Client : เป็นคลาสที่ลั่ง strategy ที่จะใช้ให้กับ context

OUR TEAM

- 1.นายธิติรัตน์ หล่อเจริญ
- 2.นายวีโรจน์ แสงสงข์
- 3.น.ส.สุชาวดี ตุ้มสมบัติ

650710215 (designer)
650710221 (programmer)
650710590 (presenter)



THANK YOU