Strategy pattern

The payment system for a flight booking is as follows in the system

In the “FlightOrder” class the methods

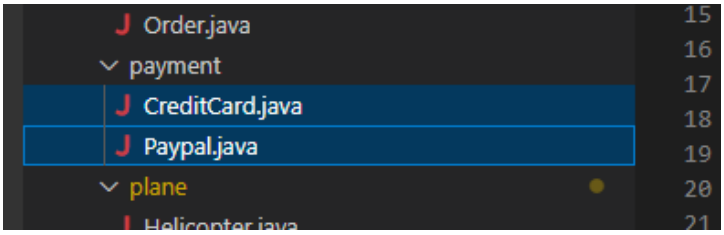
```
processOrderWithCreditCardDetail
processOrderWithCreditCard
payWithCreditCard
processOrderWithPayPal
payWithPayPal
```

are used to process a payment either with Cred card or paypal

```
public boolean processOrderWithCreditCardDetail(String number, Date expirationDate, String cvv) throws IllegalStateException {
    CreditCard creditCard = new CreditCard(number, expirationDate, cvv);
    return processOrderWithCreditCard(creditCard);
}

public boolean processOrderWithCreditCard(CreditCard creditCard) throws IllegalStateException {
    if (isClosed()) {
        // Payment is already proceeded
        return true;
    }
    // validate payment information
    if (!cardIsPresentAndValid(creditCard)) {
        throw new IllegalStateException(s: "Payment information is not set or not valid.");
    }
    boolean isPaid = payWithCreditCard(creditCard, this.getPrice());
    if (isPaid) {
        this.setClosed();
    }
    return isPaid;
}

private boolean cardIsPresentAndValid(CreditCard card) {
    return card != null && card.isValid();
}
```

```
public boolean processOrderWithPayPal(String email, String password) throws IllegalStateException {
    if (isClosed()) {
        // Payment is already proceeded
        return true;
    }
    // validate payment information
    if (email == null || password == null || !email.equals(Paypal.DATA_BASE.get(password))) {
        throw new IllegalStateException(s: "Payment information is not set or not valid.");
    }
    boolean isPaid = payWithPayPal(email, password, this.getPrice());
    if (isPaid) {
        this.setClosed();
    }
    return isPaid;
}

public boolean payWithCreditCard(CreditCard card, double amount) throws IllegalStateException {
    if (cardIsPresentAndValid(card)) {
        System.out.println("Paying " + getPrice() + " using Credit Card.");
        double remainingAmount = card.getAmount() - getPrice();
        if (remainingAmount < 0) {
            System.out.printf(format: "Card limit reached - Balance: %f%n", remainingAmount);
            throw new IllegalStateException(s: "Card limit reached");
        }
        card.setAmount(remainingAmount);
        return true;
    } else {
        return false;
    }
}

public boolean payWithPayPal(String email, String password, double amount) throws IllegalStateException {
    if (email.equals(Paypal.DATA_BASE.get(password))) {
        System.out.println("Paying " + getPrice() + " using PayPal.");
        return true;
    } else {
        return false;
    }
}
```

Where the creditcard and paypal classes are defined outside



–credit card

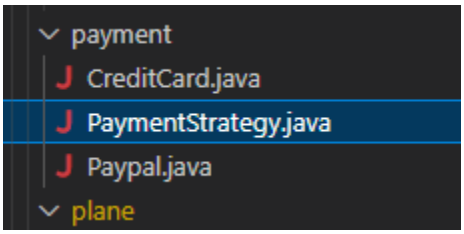
```java
package flight.reservation.payment;

import java.util.Date;

/**
 * Dummy credit card class.
 */
public class CreditCard {
    private double amount;
    private String number;
    private Date date;
    private String cvv;
    private boolean valid;

    public CreditCard(String number, Date date, String cvv) {
        this.amount = 100000;
        this.number = number;
        this.date = date;
        this.cvv = cvv;
        this.setValid();
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }

    public boolean isValid() {
        return valid;
    }

    public void setValid() {
        // Dummy validation
        this.valid = number.length() > 0 && date.getTime() > System.currentTimeMillis() && !cvv.equals("000");
    }
}
```

-Paypal

```java
package flight.reservation.payment;

import java.util.HashMap;
import java.util.Map;

public class Paypal {
    public static final Map<String, String> DATA_BASE = new HashMap<>();

    static {
        DATA_BASE.put("amanda1985", "amanda@ya.com");
        DATA_BASE.put("qwerty", "john@amazon.eu");
    }
}
```

A payment strategy could be implemented as follows

Adding a new interface for payment




```java
package flight.reservation.payment;

public interface PaymentStrategy {
    public void Pay(int amount);
}
```

Now implement this interface in all payment classes

And the payments could be reduced to as follows

```java
public boolean processOrderWithCreditCardDetail(String number, Date expirationDate, String cvv) throws Illega
    CreditCard creditCard = new CreditCard(number, expirationDate, cvv);
    return processOrderWithCreditCard(creditCard);
}

public boolean processOrderWithCreditCard(CreditCard creditCard) throws IllegalStateException {
    if (isClosed()) {
        // Payment is already proceeded
        return true;
    }
    // validate payment information
    if (!cardIsPresentAndValid(creditCard)) {
        throw new IllegalStateException(s: "Payment information is not set or not valid.");
    }
    Pay(creditCard);
    return this.isClosed();
}

private boolean cardIsPresentAndValid(CreditCard card) {
    return card != null && card.isValid();
}

public boolean processOrderWithPayPal(String email, String password) throws IllegalStateException {
    if (isClosed()) {
        // Payment is already proceeded
        return true;
    }
    // validate payment information
    if (email == null || password == null || !email.equals(Paypal.DATA_BASE.get(password))) {
        throw new IllegalStateException(s: "Payment information is not set or not valid.");
    }
    Pay(new Paypal(email, password));
    return this.isClosed();
}


public void Pay(PaymentStrategy paymentMethod){
    boolean isPaid=paymentMethod.Pay(getPrice());
    if (isPaid) {
        this.setClosed();
    }
```