6.2 Mẫu thiết kế ADAPTER

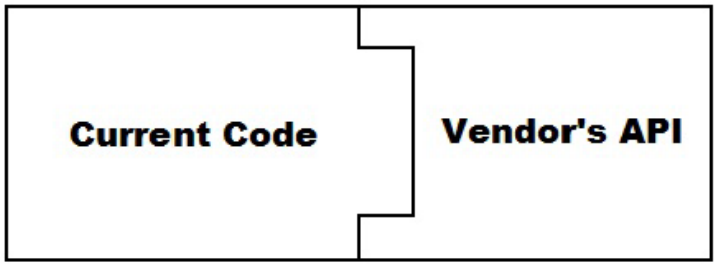
6.2.1 Mẫu thiết kế Adapter

Người phát triển phần mềm Max, làm việc xây dựng website thương mại điện tử. Website cho phép người sử dụng mua và trả tiền trực tuyến. Trang này được tích hợp với cổng trả tiền bên thứ ba, mà theo đó người sử dụng cần trả hóa đơn sử dụng thẻ credit card của họ. Mọi việc tiến triển tốt, cho đến khi quản lý dự án kêu anh ta thay đổi một chút.

Quản lý dự án nói với anh ta là họ phải thay đổi nhà cung cấp cổng trả tiền, và họ cần phải cài đặt lại điều đó.

Vấn đề nảy sinh ở đây là trang đã được đính vào cổng trả tiền Xpay mà sử dụng đối tượng kiểu *Xpay*. Nhà cung cấp mới PayD, chỉ cho phép dùng các đối tượng kiểu *PayD* để xử lý. Max không muốn thay đổi toàn bộ tập khoảng 100 lớp đã tham chiếu đến các đối tượng kiểu *Xpay*. Nó cũng tiềm ẩn rủi ro cho Dự án, mà đã gần đưa vào sử dụng. Anh ta không thể thay đổi công cụ của cổng trả tiền bên thứ ba. Vấn đề xảy ra vì các giao diện không tương thích giữa hai phần code. Để tiến trình vẫn tiếp tục làm việc, Max cần phải tìm cách làm cho code tương thích với các API được cung cấp bởi cổng trả tiền mới

Code hiện tại với API của *Xpay*



Bây giờ giao diện của code hiện tại không tương thích với API của cổng trả tiền mới

6.2.2 Mẫu Adapter giải cứu

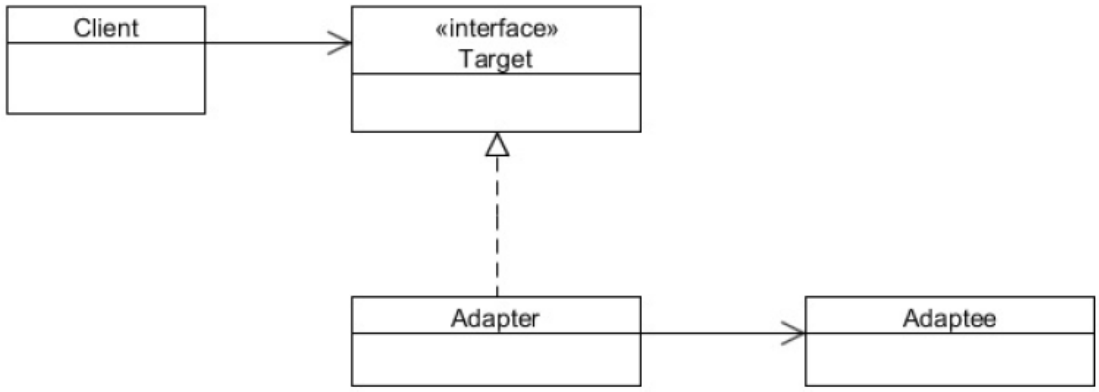
Cái mà Max cần ở đây là mẫu Adapter, mà có thể nằm giữa code và API của nhà cung cấp mới, và có thể cho phép tiến trình tiếp tục làm việc. Nhưng trước khi đưa ra giải pháp, chúng ta sẽ xem Adapter là gì và nó làm việc như thế nào.

Đôi khi, có kịch bản hai đối tượng không khớp nhau, nhưng chúng cần lắp ghép để hoàn thành công việc. Tình huống này có thể phát sinh khi chúng ta tìm cách tích hợp code đã có với code mới, hoặc khi thay đổi API của bên thứ ba. Nó xảy ra vì các giao diện không tương thích của hai đối tượng mà không khớp với nhau.

Mẫu Adapter là mẫu cấu trúc, nó cho phép bạn lắp một đối tượng hay một lớp đặt vào một đối tượng hay một lớp khác. Nó chuyển đổi giao diện của một lớp sang giao diện khác mà client mong đợi. Nó cho phép các lớp làm việc với nhau, mặc dù có các giao diện không tương thích. Nó cho phép xác lập giao diện giữa các đối tượng và các lớp mà không thay đổi trực tiếp các đối tượng và các lớp.

Bạn có thể nghĩ là Adapter giống như bộ chuyển ổ trên thực tế mà được sử dụng để kết nối giữa hai thiết bị mà không nối trực tiếp được với nhau. Một Adapter nằm giữa các thiết bị này để cho phép luồng vào từ một thiết bị chảy sang thiết bị khác ở dạng mà nó muốn, nếu không sẽ không nối hai thiết bị đó với nhau được.

Một Adapter sử dụng kết hợp để lưu đối tượng mà nó hỗ trợ thích nghi, và khi phương thức của Adapter được triệu gọi, nó chuyển các lời gọi này thành những điều mà đối tượng được thích nghi có thể hiểu và truyền các lời gọi này đến các đối tượng được thích nghi. Code mà gọi Adapter không cần biết là nó không tương tác với kiểu đối tượng mà nó nghĩ, mà thay vào đó là các đối tượng được thích nghi.



Giải pháp cho bài toán

Giao diện Xpay được trông như sau:

```
package com.javacodegeeks.patterns.adapterpattern.xpay;

public interface Xpay {

        public String getCreditCardNo();
        public String getCustomerName();
        public String getCardExpMonth();
        public String getCardExpYear();
        public Short getCardCVVNo();
        public Double getAmount();

        public void setCreditCardNo(String creditCardNo);
        public void setCustomerName(String customerName);
        public void setCardExpMonth(String cardExpMonth);
        public void setCardExpYear(String cardExpYear);
        public void setCardCVVNo(Short cardCVVNo);
        public void setAmount(Double amount);

}
```

Nó chứa tập các phương thức *set* và *get* các thông tin về thẻ và tên chủ tài khoản. Giao diện Xpay này đã được cài đặt trong code mà được sử dụng để tạo đối tượng kiểu này và đưa đối tượng cho các API của cổng trả tiền.

Lớp sau đây xác định cài đặt cho giao diện Xpay:

```
package com.javacodegeeks.patterns.adapterpattern.site;

import com.javacodegeeks.patterns.adapterpattern.xpay.Xpay;

public class XpayImpl implements Xpay{

        private String creditCardNo;
        private String customerName;
        private String cardExpMonth;
        private String cardExpYear;
        private Short cardCVVNo;
        private Double amount;

        @Override
        public String getCreditCardNo() {
                return creditCardNo;
        }

        @Override
        public String getCustomerName() {
                return customerName;
        }

        @Override
        public String getCardExpMonth() {
                return cardExpMonth;
        }

        @Override
        public String getCardExpYear() {
                return cardExpYear;
        }
```

Nhưng Xpay đã được tạo bởi hầu hết các phần khác của code, thật sư là khó và rủi ro khi thay đổi toàn bộ tập các lớp này.

Chúng ta cần một cách khác, mà có thể thực hiện yêu cầu của nhà cung cấp mới để xử lý trả tiền mà cũng làm cho code hiện thời không hoặc thay đổi ít. Cách này được cung cấp bởi mẫu Adapter.

Chúng ta sẽ tạo ra Adapter mà sẽ là kiểu *PayD* và bao bọc đối tượng kiểu *Xpay*.

```
        @Override
        public Short getCardCVVNo() {
                return cardCVVNo;
        }

        @Override
        public Double getAmount() {
                return amount;
        }

        @Override
        public void setCreditCardNo(String creditCardNo) {
                this.creditCardNo = creditCardNo;
        }

        @Override
        public void setCustomerName(String customerName) {
                this.customerName = customerName;
        }

        @Override
        public void setCardExpMonth(String cardExpMonth) {
                this.cardExpMonth = cardExpMonth;
        }

        @Override
        public void setCardExpYear(String cardExpYear) {
                this.cardExpYear = cardExpYear;
        }

        @Override
        public void setCardCVVNo(Short cardCVVNo) {
                this.cardCVVNo = cardCVVNo;
        }

        @Override
        public void setAmount(Double amount) {
                this.amount = amount;
        }

}
```

Giao diện của nhà cung cấp mới trông như sau:

```
package com.javacodegeeks.patterns.adapterpattern.payd;

public interface PayD {

        public String getCustCardNo();
        public String getCardOwnerName();
        public String getCardExpMonthDate();
        public Integer getCVVNo();
        public Double getTotalAmount();

        public void setCustCardNo(String custCardNo);
        public void setCardOwnerName(String cardOwnerName);
        public void setCardExpMonthDate(String cardExpMonthDate);
        public void setCVVNo(Integer cVVNo);
        public void setTotalAmount(Double totalAmount);
}
```

Như bạn đã thấy, giao diện này có tập phương thức khác mà cần được cài đặt trong code. Nhưng Xpay đã được tạo bởi hầu hết các phần khác của code, thật sư là khó và rủi ro khi thay đổi toàn bộ tập các lớp này.

Chúng ta cần một cách khác, mà có thể thực hiện yêu cầu của nhà cung cấp mới để xử lý trả tiền mà cũng làm cho code hiện thời không hoặc thay đổi ít. Cách này được cung cấp bởi mẫu Adapter.

Chúng ta sẽ tạo ra Adapter mà sẽ là kiểu *PayD* và bao bọc đối tượng kiểu *Xpay*.

```
package com.javacodegeeks.patterns.adapterpattern.site;

import com.javacodegeeks.patterns.adapterpattern.payd.PayD;
import com.javacodegeeks.patterns.adapterpattern.xpay.Xpay;

public class XpayToPayDAdapter implements PayD{

        private String custCardNo;
        private String cardOwnerName;
        private String cardExpMonthDate;
        private Integer cVVNo;
        private Double totalAmount;

        private final Xpay xpay;

        public XpayToPayDAdapter(Xpay xpay){
                this.xpay = xpay;
                setProp();
        }

        @Override
        public String getCustCardNo() {
                return custCardNo;
        }

        @Override
        public String getCardOwnerName() {
                return cardOwnerName;
        }
```

```
        @Override
        public String getCardExpMonthDate() {
                return cardExpMonthDate;
        }

        @Override
        public Integer getCVVNo() {
                return cVVNo;
        }

        @Override
        public Double getTotalAmount() {
                return totalAmount;
        }

        @Override
        public void setCustCardNo(String custCardNo) {
                this.custCardNo = custCardNo;
        }

        @Override
        public void setCardOwnerName(String cardOwnerName) {
                this.cardOwnerName = cardOwnerName;
        }
```

```
        @Override
        public void setCardExpMonthDate(String cardExpMonthDate) {
                this.cardExpMonthDate = cardExpMonthDate;
        }

        @Override
        public void setCVVNo(Integer cVVNo) {
                this.cVVNo = cVVNo;
        }

        @Override
        public void setTotalAmount(Double totalAmount) {
                this.totalAmount = totalAmount;
        }

        private void setProp(){
                setCardOwnerName(this.xpay.getCustomerName());
                setCustCardNo(this.xpay.getCreditCardNo());
                setCardExpMonthDate(this.xpay.getCardExpMonth()+"/"+this.xpay. ↩
                    getCardExpYear());
                setCVVNo(this.xpay.getCardCVVNo().intValue());
                setTotalAmount(this.xpay.getAmount());
        }

}
```

Trong đoạn code trên đây, chúng ta tạo Adapter (*XpayToPayDAdapter*). Adapter cài đặt giao diện PayD, nó được yêu cầu mô phỏng đối tượng kiểu PayD. Adapter sử dụng kết hợp đối tượng để giữ đối tượng mà nó hỗ trợ thích nghi là đối tượng Xpay. Đối tượng này được truyền cho Adapter thông qua hàm tạo.

Bây giờ, lưu ý rằng ta có hai kiểu giao diện không tương thích, mà ta cần khớp với nhau sử dụng Adapter để làm cho code hoạt động. Các giao diện này có tập phương thức khác nhau.

Nhưng mục tiêu chính của các giao diện này rất giống nhau, tức là cung cấp thông tin khách hàng và thẻ tín dụng để chuyển cho cổng trả tiền.

Phương thức *setProp ()* của lớp trên đây được sử dụng để thiết lập các tính chất của Xpay vào đối tượng PayD. Chúng ta thiết lập các phương thức mà làm việc tương tự nhau trong hai giao diện. Tuy nhiên, chỉ có phương thức duy nhất trong giao diện PayD để thiết lập tháng và năm của thẻ tín dụng, trái ngược với hai phương thức trong Xpay. Chúng ta hợp kết quả của hai phương thức của đối tượng Xpay

```
(this.xpay.getCardExpMonth()+"/"+this.xpay.getCardExpYear())
```

và thiết lập nó vào phương thức

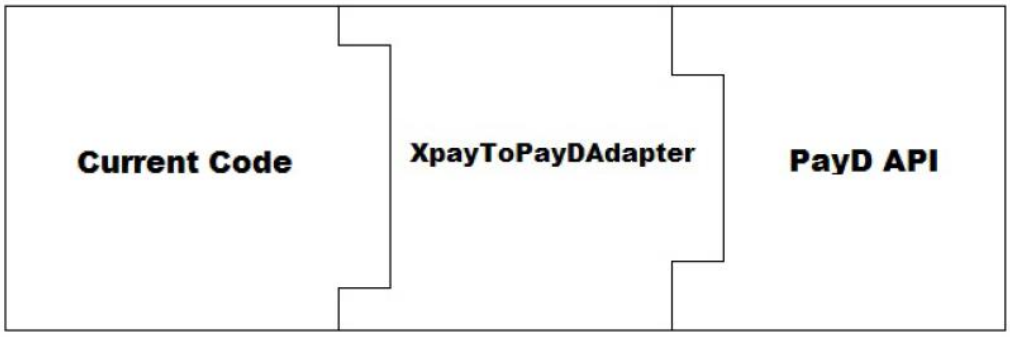
```
setCardExpMonthDate()
```

Bây giờ chúng ta kiểm tra đoạn code trên giải quyết bài toán của Max như thế nào

```
package com.javacodegeeks.patterns.adapterpattern.site;

import com.javacodegeeks.patterns.adapterpattern.payd.PayD;
import com.javacodegeeks.patterns.adapterpattern.xpay.Xpay;

public class RunAdapterExample {

        public static void main(String[] args) {

                // Object for Xpay
                Xpay xpay = new XpayImpl();
                xpay.setCreditCardNo("4789565874102365");
                xpay.setCustomerName("Max Warner");
                xpay.setCardExpMonth("09");
                xpay.setCardExpYear("25");
                xpay.setCardCVVNo((short)235);
                xpay.setAmount(2565.23);

                PayD payD = new XpayToPayDAdapter(xpay);
                testPayD(payD);
        }

        private static void testPayD(PayD payD){

                System.out.println(payD.getCardOwnerName());
                System.out.println(payD.getCustCardNo());
                System.out.println(payD.getCardExpMonthDate());
                System.out.println(payD.getCVVNo());
                System.out.println(payD.getTotalAmount());
        }
}
```

Trong lớp trên, trước hết chúng ta tạo đối tượng Xpay và thiết lập các tính chất của nó. Sau đó, ta tạo đối tượng adapter và truyền Xpay cho nó trong hàm tạo, và gán nó cho giao diện PayD. Phương thức tĩnh *TestPayD ()* dùng đối tượng PayD như đối số mà chạy và in ra các phương thức để kiểm tra chạy. Kiểu được truyền cho phương thức *TestPayD ()* là kiểu *PayD*, phương thức sẽ thực hiện không có bất cứ vấn đề gì. Ở bên trên, ta truyền Adapter cho nó, mà sẽ thấy như kiểu *PayD*, nhưng bên trong bao bọc đối tượng kiểu *Xpay*.

Như vậy trong Dự án của Max tất cả những gì chúng ta cần làm là cài đặt API của cổng trả tiền mới vào trong code và truyền Adapter đến phương thức của nhà cung cấp này để thanh toán tiền. Chúng ta không cần thay đổi bất cứ cái gì trong code đã có.



6.2.1 Khi nào sử dụng mẫu Adapter

Mẫu Adapter cần được sử dụng khi:

- Có lớp đã tồn tại, và giao diện của nó không khớp với cái ta cần
- Bạn muốn tạo một lớp tái sử dụng mà hợp tác với các lớp không liên quan và không biết trước và rằng các lớp này không cần thiết phải có giao diện tương thích.
- Có một số lớp con đã tồn tại để sử dụng, nhưng nó không thể thích nghi giao diện của chúng bằng cách tạo lớp con cho mỗi lớp. Đối tượng Adapter có thể thích nghi giao diện cho các lớp cha.