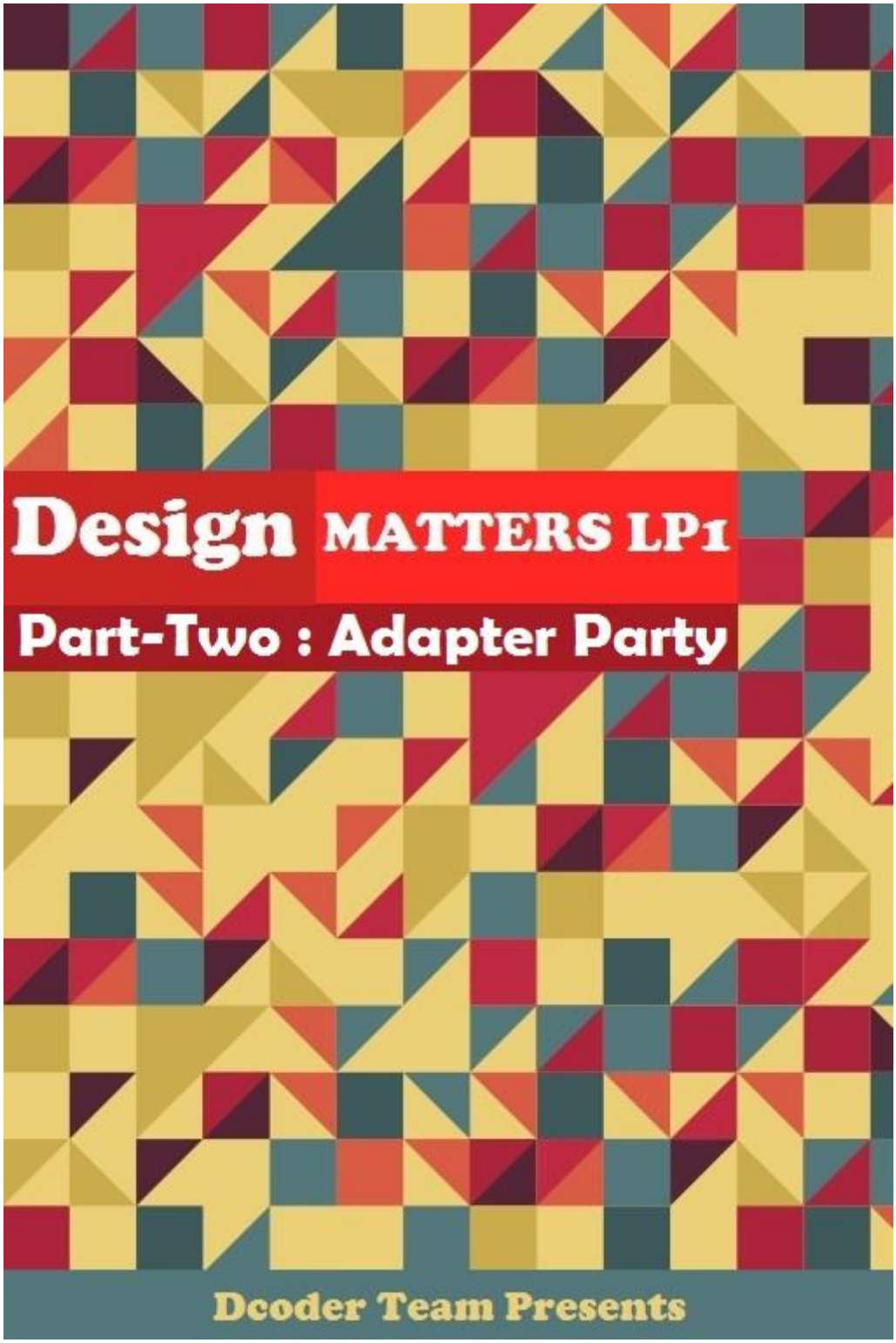

Design MATTERS LP1
Part-Two : Adapter Party
Dcoder Team Presents

Information

Article : Design Matters LP 1

Authors : Mhmv and Joulook

Prerequisite : Object-Oriented Concepts

Published By : Dcoder Team

Follow us on Telegram : https://t.me/de_coder

The Cover of Article is Adapted From “Design Patterns Explained Simply By Alexander Shvets” Cover Page

Part-Two : Adapter Party

A Tales of Disease

مکس یک برنامه نویس بود که در یکی از این شرکت های سرکاریه تجارت الکترونیک کار می کرد شرکتی که مکس در آن کار می کرد یک website داشت که به کاربر ها اجازه می داد عملیات پرداخت و خرید را به صورت Online انجام بدهند. سایت آن ها به یک درگاه پرداخت 3rd Party ، Integrate شده بود. درگاه پرداخت 3rd Party یعنی یک درگاه پرداختی که توسط یک Vendor دیگه ای ساخته شده و ما داریم ازش برای انجام Transaction های مالی استفاده می کنیم. بنابر این کاربر ها می توانند صورت حساب های خودشان را از طریق سایت آن ها به واسطه کارت اعتباریشان پرداخت کنند. همه چیز به خوبی و خوشی داشت پیش می رفت تا این که مدیر مکس خواب هایی برای پروژه شان دید و او را برای تغییرات در پروژه صدا زد و مکس می دانست که اتفاقی که قرار است بیافتد اصلا خوش آیند نیست

مدیر به او گفت که برنامه دارند تا Vendor ای که درگاه پرداخت از آن گرفته بودند را عوض کنند و او باید این تغییرات را در کد اعمال کند

مشکل از آنجایی به وجود می آمد که Site به درگاه پرداختِ Xpay ، Attach شده بود که Object ای از تایپِ Xpay دریافت می کرد. نام Vendor جدید PayD بود که فقط اجازه Process روی Object هایی از نوعِ PayD را می داد . مکس نمی خواست که تمام 100 کلاسی را که به Object ای از تایپِ Xpay ، Reference شده بود را تغییر بدهد . این کار همچنین باعث افزایش ریسک روی پروژه می شد که در حال اجرا بود و کاربر ها از آن در حال استفاده بودند. همچنین او که نمی تواند ابزار های درگاه پرداخت 3rd Party را تغییر دهد به تعبیری می توان مشکل او را با دو شکل زیر توصیف کرد

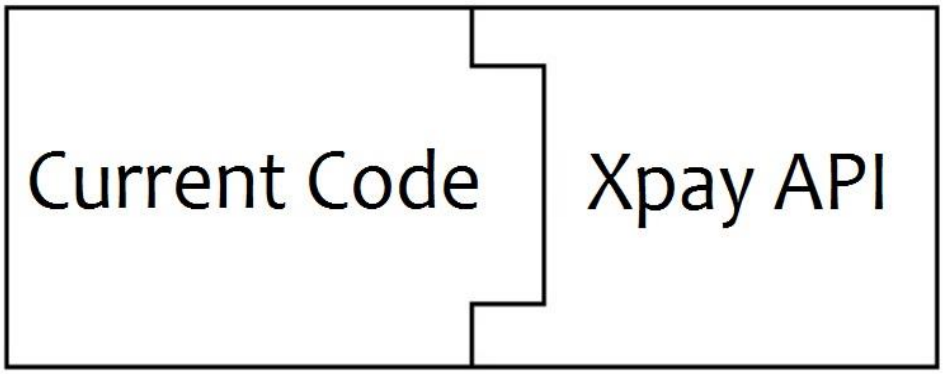


Figure 2.1: screenshot

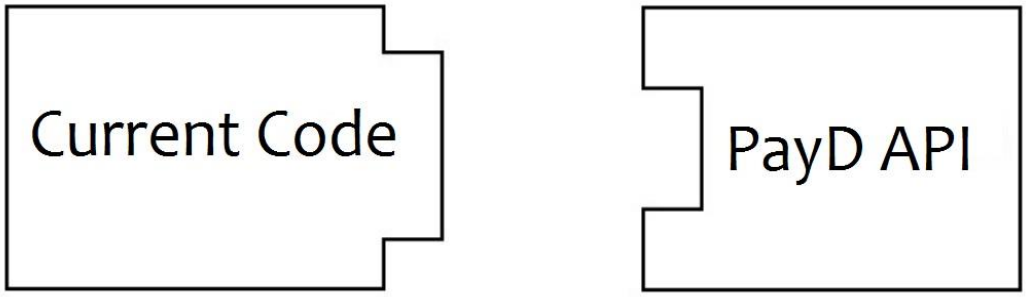


Figure 2.2: screenshot

Adapter Party Season

خب همانطور که مشاهده می کنید ما به مشکلی برخوردیم که پیدا کردن راه حل آن کار راحتی نیست اگر که با Design Pattern ها آشنا نباشیم . در این جلسه قصد داریم که یکی از Design Pattern های پر کاربرد که در دسته Structural قرار دارد را به شما معرفی کنیم ، Adapter Pattern که در بعضی کتاب ها از آن به اسم Wrapper Pattern یاد می شود

در حالت کلی یک Adapter یک Interface را(Adaptee's) به Interface دیگر (Target) تطبیق میدهد و در نتیجه یک Abstraction یکپارچه از Interface های مختلف بدست می آید

یک Class Adapter این کار را به وسیله Inheriting به طور Private از یک Adaptee Class انجام می دهد و بحث Multi-Inheritance را پیش می آورد این بحث و همچنین Object Adapter Pattern را جلوتر توضیح خواهیم داد اما ابتدا بگذارید به مفهوم Adapter بپردازیم

شما می توانید به Adapter همان دیدی را داشته باشید که در دنیای واقعی به یک آداپتور دارید. در دنیای واقعی از آداپتور برای وصل کردن دو قطعه مختلف الکتریکی بهم که نمی توان به طور مستقیم بهم وصلشان کرد استفاده می شود . آداپتور بین این دو قطعه قرار میگیرد و جریان را از یکی از قطعه ها دریافت می کند و به فرمی که برای قطعه دیگر قابل استفاده باشد در می آورد و به آن تحویل می دهد که این کار بدون استفاده از آداپتور بخاطر وجود Incompatible Interface هایی که دارند ممکن نیست

پترنِ Adapter به شما اجازه می دهد که یک Object و یا Class بتواند چیزی را که Object و یا Class نیاز دارد را Expose کند . کاری که می کند این است که Interface یک کلاس را به آن Interface ای که Client نیاز دارد Convert کند در اصل به کلاس ها اجازه می دهد که بتوانند با یکدیگر کار کنند در صورتی که بخاطر Incompatible Interface هایی که دارند نمی توانستند و این امکان را به ما می دهد که Interface بین Object ها و Class ها را بتوانیم Fix کنیم بدون ایجاد تغییر در Class ها و Object ها به طور مستقیم

Object Adapter Pattern and Believe me... It's Easy to Learn!

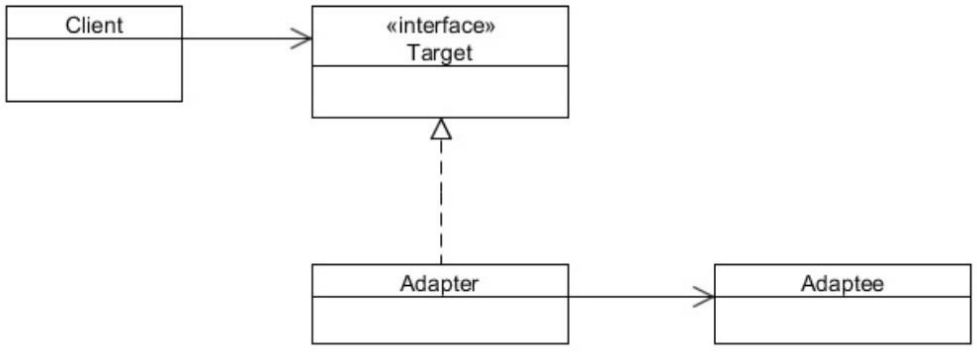


Figure 2.3: screenshot

ابتدا به Class Diagram فوق توجه کنید

یک Adapter برای Store کردن یک Object که نیاز دارد به حالت جدید Adapt شود از Composition استفاده می کند و زمانی که یک Method از Adapter ، Call می شود به نوعی آن Call ها را تبدیل می کند به چیزی که آن Adapted Object ها می توانند متوجه شوند و آن Calling ها را Pass کند به Object های Adapt شده . Code ای که Adapter را Call میکند هیچ وقت نیاز نیست بداند که آن Object ای که دارد باهاش کار میکند دقیقا همان چیزی نیست که فکر می کند باشد در حقیقت Object های Adapt شده هستند

خب می توانیم بحث Adapter را ادامه بدهیم اما ابتدا بهتر است با توجه به داغ بودن تنور همینجا مسئله مکس را حل کنیم و سپس بحث را ادامه دهیم زیرا برای حل مسئله Max نیاز به استفاده از Object Adapter Pattern داریم

Max Knows About Patterns !! piece of cake.

اگر بخواهیم که به صورت شماتیک نشان دهیم که Max چه کاری را برای حل مسئله باید انجام دهد چیزی شبیه به شکل زیر می شود

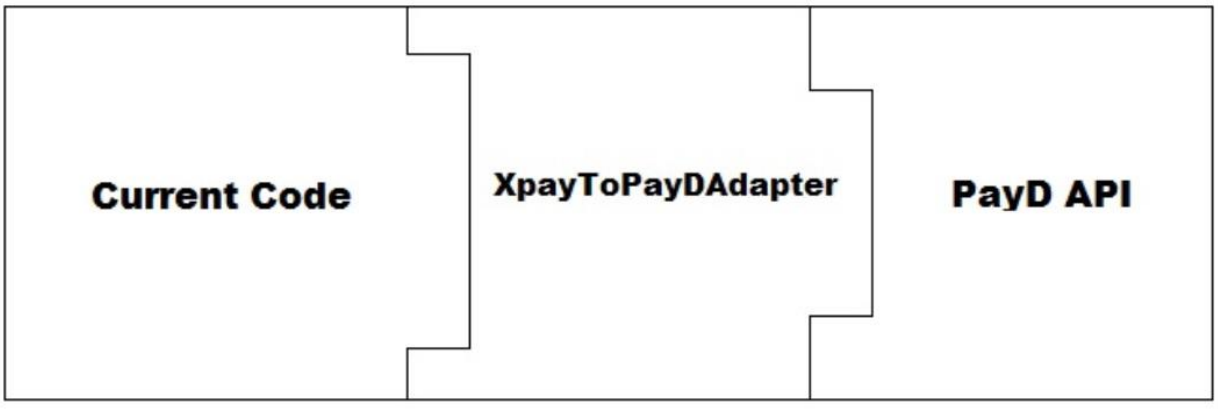


Figure 2.4: screenshot

در اینجا Adaptee همان Xpay هست که باید Object اش Adapted شود و Target Interface همان PayD می باشد که باید در Adapter ، Implement شود

کد Interface 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);
}
```

کد Implement کردن Interface 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;
}
@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;
}
}
```

کد مربوط به Interface 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);
}
```

کد مربوط به Adaptor بین Xpay و PayD

```
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());
}
}
```

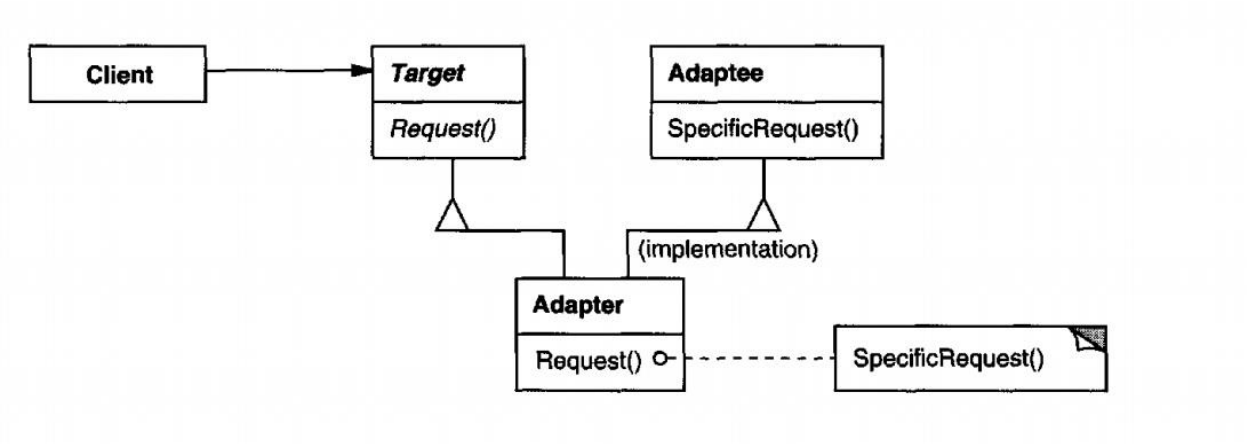
حال میخواهیم Adapter نوشته شده را تست کنیم

```
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());
}
}
```

و به این صورت Max توانست بدون تغییر Class ها و Object ها تغییراتی را که مدیر میخواست را اعمال کند

Class Adapter Pattern And Multi-Inheritance Disease

در شکل زیر Class Diagram مربوط به این نوع Pattern را مشاهده می کنید



یک Class Adapter متکی است بر Multiple Inheritance برای این که بتواند یک Interface را به یک Interface دیگر Adapt کند . برای پیاده سازی یک Class Adapter به این صورت عمل می کنیم که یک Adapter به صورت Publicly از Target ، Inherit می کند و به صورت Privately از Adaptee ، Inherit میکند و در نتیجه Adapter یک SubType از Target می شود اما نه از Adaptee

بحثی که پیش می آید این است که در زبان هایی مثل Java که بنا به دلایل درستی از Multi Inheritance جلوگیری شده است نمی توانیم از این Design Pattern استفاده کنیم . برخی منابع اشاره به این دارند که می توانیم با استفاده از Interface این تکنیک را پیاده سازی کنیم . اما برخی منابع هم با توجه به رفرنس اصلی این مبحث GoF یا همان Gang of Four یا همان کتابی که به عنوان اولین رفرنس در انتهای این Article معرفی شده است می گویند که بنا به تعریف دقیق Adapter باید از چند Base Class ، Inherit شده باشد و نه Interface

When To Use Adapter Pattern

1. زمانی که شما میخواهید از یک Existing Class استفاده کنید و Interface آن با کاری که می خواهید انجام بدهید Match نیست
2. زمانی که شما میخواهید یک Reusable Class بسازید که با کلاس های غیر قابل دیدن و بی مرتبط در حال Cooperate کردن است که این کلاس ها لزوما دارای Compatible Interface نیستند

Where You Are Now ?

بحث Design Pattern Adapter به همینجا ختم نمی شود بحث های تخصصی تر نسبت به این که در چه مواقعی از این Pattern باید استفاده شود و یا این که چه عواقب ای ممکن است داشته باشد و...

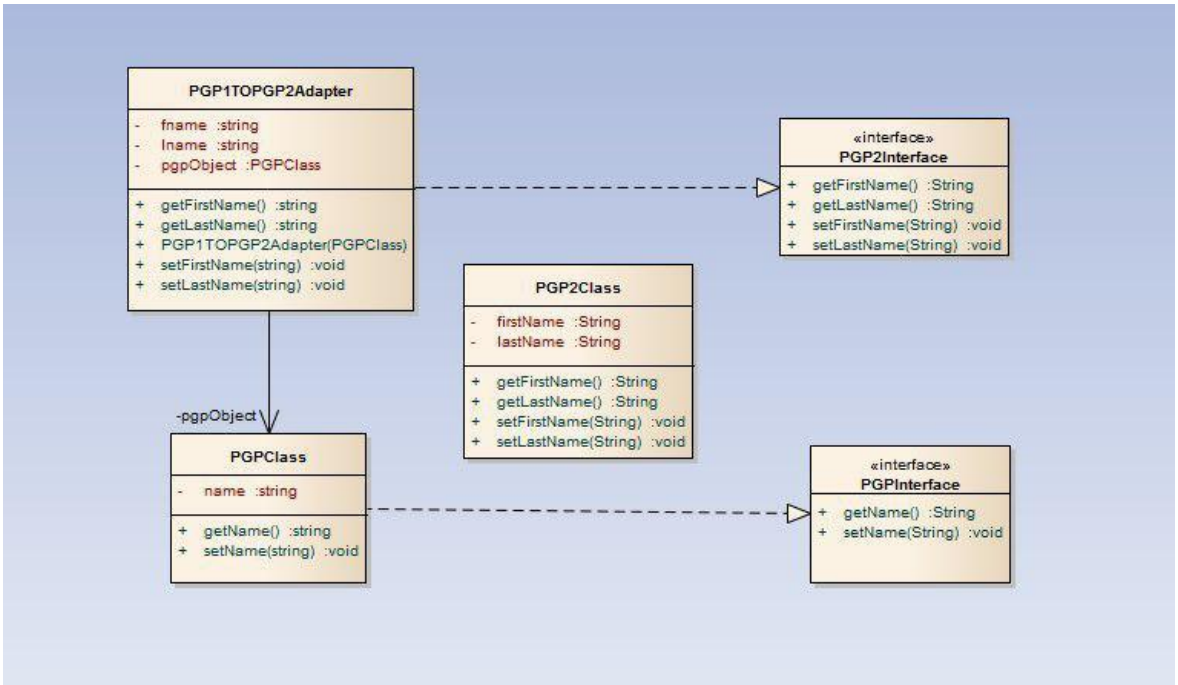
شما می توانید با مطالعه فصل چهارم کتاب اول بخش Bibliography

و فصل دوم کتاب دوم بخش Bibliography

این مباحث را پوشش دهید

Bounds

یک شرکت نرم افزاری قصد دارد Back-End خود را که از یک Vendor دیگه به نام PGP تهییه کرده بود را به دلایل افزایش قیمت عوض کرده و از Vendor ای به نام GPG2 ، یک Back-End جدید تهییه کند. با استفاده از Adapter Pattern تغییرات لازمه را اعمال کردیم و Class Diagram و Source کد آن را به زبان های Java و C# برایتان قرار دادیم



تیم دیکدر یک سناریو برای شما آماده کرده و آن را با استفاده از Adapter Design Pattern به سه زبان Java و C# پیاده سازی کرده است که می توانید سورس کد پیاده سازی شده آن را از کانال زیر دانلود کنید

http://t.me/de_coder

Our Plan For Next Session

در جلسات آینده شروع به معرفی Design Pattern ها خواهیم کرد و هر یک را در یک مثال به زبان های Java , C# , C++ پیاده سازی میکنیم و سورس کدش را برایتان قرار خواهیم داد

Bibliography

1. **Design Patterns Elements of Reusable Object-Oriented Software -Gmma and Helm and Johnson and Vlissides -Addison Wesley**

   Download Link : https://t.me/debrary/525

2. **Java Design Patterns -Rohit Joshi -Exelixis Media**

   Download Link : https://t.me/debrary/529