KTO KARATAY
ÜNİVERSİTESİ

T.C.
KTO KARATAY ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRİK BİLGİSAYAR MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS BÖLÜMÜ

NESNEYE YÖNELİK YAZILIM MÜHENDİSLİĞİ (OBJECT ORIENTED SOFTWARE ENGINEERING)

DERSİ FİNAL ÖDEVİ

ÖDEV KONUSU

Observer Design Pattern

Golden Hammer, Wolf Ticket, Intellectual Violence Anti Pattern

ÖĞRETİM ÜYESİ

Doktor Öğretim Üyesi Ali ÖZTÜRK

ÖĞRENCİ

Fatih KAPLAN

21743401

Konya, 2018

Birinci Kısım Design Pattern : Observer Pattern

a) Bu tasarım şablonunun kullanılmasındaki motivasyon nedir?

Motivasyon: Tasarlanmış olan sistem içerisinde, değişimini izlemek istediğimiz bir değer için kullanılır.

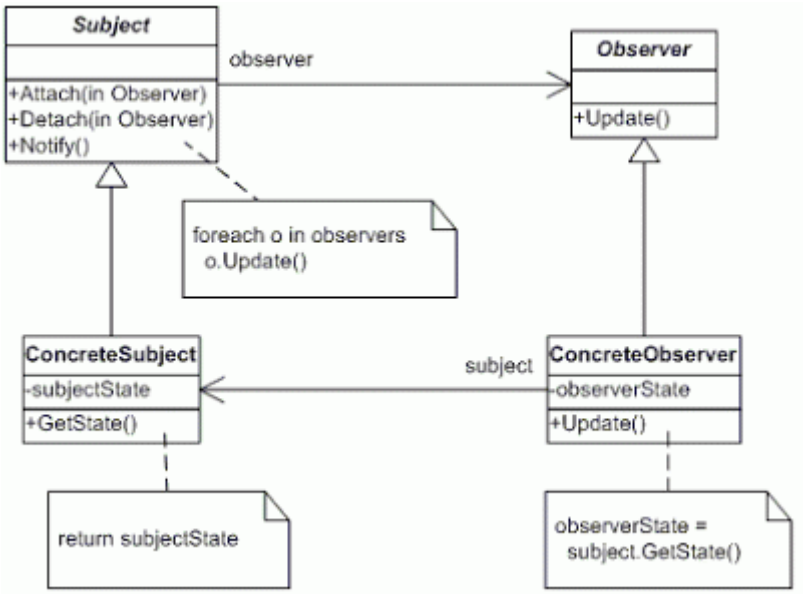
Diğer tasarım şablonlarında olduğu gibi Observer tasarım şablonu da oldukça önemlidir ve önemli bir amaca hizmet etmektedir. Observer yani gözlemci tasarım şablonları bir hayli yaygın tasarım şablonlarından bir tanesidir. Temel olarak hedef olarak belirlenen bir nesnenin (subject) kendisine bağlı olan observer yani gözlemcilerinin bir listesini içerisinde tutar. Ve bu liste içerisinde bir durumda (state) değişiklik olduğunda hedefin otomatik olarak listeyi modify eder.

Observer Pattern'in arkasındaki temel motivasyon ilgili nesneler arasında sıkı sıkıya bağlılık göstermeden tutarlılığı sürdürme arzusudur. Sınıfları sıkıca birleştirmeden ilgili nesneler arasında tutarlılık sağlama ihtiyacına bir çözümdür. Yaygın olarak web sitelerindeki üyelik işlemleri gösterilir. Örneğin Youtube'da üyeliğimiz var ve abone olduğumuz bir kanal bulunuyor. O kanala abone olan herkesin kanalın yeni bir video yüklediğinde haberdar olmasını sağlayan metottur. Üyelerin her biri farklı kişiler oldukları için bu üyelerin bir arada haberdar almaları yerine her biri için ayrı ayrı bildirimlerin sağlanmasıdır da diyebiliriz.

b) Bu tasarım şablonu hakkında detaylı bilgi veriniz.

Bu tasarım şablonu one-to-many olayını destekleyen bir tasarım desenidir. Mesela bir nesnenin değişikliğinden farklı nesneler de etkilenecek ise bu kalıbın kullanılması tavsiye edilmektedir. Bu yapı birçok yerde karşımıza çıkmaktadır. Observer yani gözlemci tasarım şablonu davranışsal (behavioral) kalıplar içerisinde yer almakta ve incelenmektedir.

Observer, Türkçe' de gözlemci, izleyici, gözcü veya gözetmen kelimeleri ile karşılık bulmaktadır. Elimizdeki mevcut bir nesnenin durumunda herhangi bir değişiklik olduğunda, bu değişikliklerden diğer nesneleri haberdar eden bir tasarım desenidir. Örnek vermek gerekirse elimizde bulunan bir 'X' nesnesinin 'Y' özelliğinde bir güncelleme, değişiklik veya belirli bir şartın gerçekleşmesi gibi bir durum söz konusu olduğunda bu 'X' nesnesini izleyen-gözlemleyen diğer 'Z', 'K' vb. nesnelerine bu yeni durumu bildiren sisteme Observer Tasarım Deseni denilmektedir.



Şekil 1 UML Diyagramı

Observer tasarım şablonunu detaylı inceleyebilmek adına bir örnek üzerinde gidecek olursak elimizde bir öğrenci nesnesi olduğunu farz edelim. Bu öğrenciyi takip edenleri ise ailesi ve öğretmeni olarak düşünelim. Okul öğrencisi derse devamsızlık yaptığı zaman onu takip edenlerden olan anne ve babası ile öğretmenine doğrudan bu durumu haber veriyor olsun.

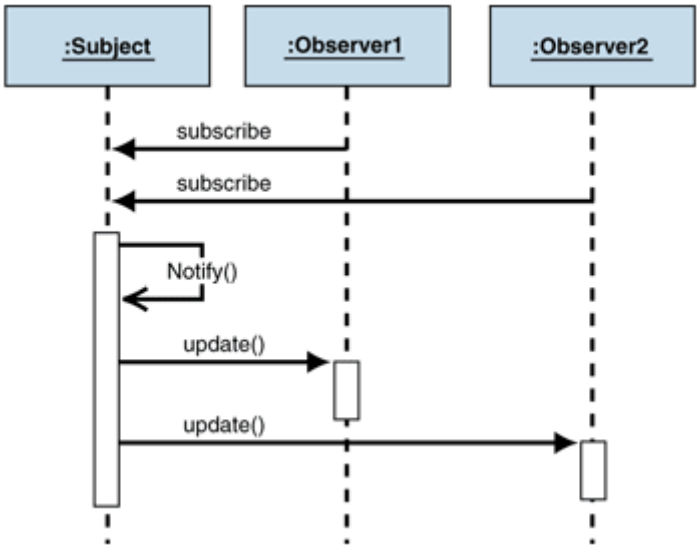
Observer Design Pattern'de yukarıdaki örneğe denk gelen aşağıdaki yapılar bulunmaktadır. Bu yapıları inceledikten sonra örneğe devam edelim.

- **Subject**
  Takip edilecek nesneyi Subject terimi ile ifade ederiz. Yukarıdaki örnek olayda Öğrenci nesnesi bizim için Subject niteliğindedir.

- **Observer**
  Subject'I takip edecek olan aboneler tek tip olmayabilirler. Yukarıdaki örnek olayda Subject'i anne, baba ve öğretmen nesneleri takip etmektedirler. Bu gibi durumlarda birden fazla tipe arayüz görevi yapacak abstract ya da interface yapımıza observer denilmektedir. Örnek olayda öğrenci nesnesini takip edenlerin birden fazla tip olmasını sağlayan system Observer Arayüzüdür.

- **Concrete**
  Subject'I takip eden nesnelerdir. Örneğimizde Anne, baba ve öğretmen nesneleri Concrete nesneleridir.

Observer Design Pattern yapısını konu başında Youtube üyelik ve kanal abonelerine aynı anda gönderilen bildirimlerin işletilmesine benzeterek anlatmaya çalışmıştım. Burada da öğrencinin okula devamsızlık yaptığı anda diğer abonelerine haber veren Okul yapısına benzetebilirim. Yukarıda da paylaştığım UML diyagramını teknik olarak incelemek gerekirse;



Şekil 2 UML Diyagramı

Diyagramda görüldüğü gibi Subject(Hedef) olarak belirtilen nesnemize Observer'lar(gözlemciler) subscribe yani abone olmuşlardır. Subject nesnemiz içerisinde herhangi bir 'Y' özelliği güncellendiğinde Notify metodu tetiklenecek ve bu metod Subject'e abone olan tüm observer'ların Update metodunu çalıştıracaktır.

Diyagramda işletilen mantığın kod ile pekiştirelim.

İlk olarak Subject'e (hedefe) birçok tipin abone olabilmeni sağlayacak Observer arayüzü geliştirelim.

```
abstract public class Observer
{
  public abstract void Update();
}
```

Observer arayüzünü abstract class olarak belirledim fakat interface olarak da belirlenebilirdi. Observer sınıfından kalıtım alan her abone sınıfa Update metodu uygulanacaktır. Haliyle Subject'de ilgili alan dğeiştiğinde abonelerin Update metodu tetiklenecektir. Şimdi ise Concrete nesneleriniz yani takip edecek olan sınıfları oluşturalım.

```
public class BabaObserver: Observer
{
  public override void Update()
  {
    Console.WriteLine("Öğrencinin devamsızlığından babasının haberi oldu.");
  }
}

public class AnneObserver: Observer
{
  public override void Update()
  {
    Console.WriteLine("Öğrencinin devamsızlığından annesinin haberi oldu.");
  }
}

public class OgretmenObserver: Observer
{
  public override void Update()
  {
    Console.WriteLine("Öğrencinin devamsızlığından öğretmenin haberi oldu.");
  }
}
```

Şimdi ise Subject'i yani takip edilecek nesneyi oluşturalım.

```
public class Ogrenci
{
  public string Adi { get; set; }
  public string Soyadi { get; set; }
  public string Memleket { get; set; }
  public int Sinif { get; set; }

  bool DevamsizlikYaptiMi;
  public bool DevamsizlikYaptiMi {
    get { return DevamsizlikYaptiMi; }
    set {
      if (value = true) {
        Notify();
        DevamsizlikYaptiMi = value;
      }else
        DevamsizlikYaptiMi = value;
    }
  }
```

```
  List<Observer> Gozlemciler ;
  public Ogrenci()
  {
    this.Gozlemciler = new List<Observer>();
  }
  public void AboneEkle(Observer observer){
    Gozlemciler.Add(observer);
  }
  public void AboneCikar(Observer observer){
    Gozlemciler.Remove(observer);
  }
  public void Notify(){
    Gozlemciler.ForEach(g=>
          {
            g.Update();
          }); } }
```

Ve son olarak tasarımı test edelim.

```
class Program
{
  public static void Main(string[] args)
  {

    Ogrenci o = new Ogrenci();
    o.AboneEkle(new BabaObserver());
    o.AboneEkle(new AnneObserver());
    o.AboneEkle(new OgretmenObserver());

    o.Adi = "Fatih";
    o.SoyAdi = "KAPLAN";
    o.Memleket = "Konya";
    o.Sinif = 1;
    o.DevamsizlikYaptiMi = true;

    Console.ReadKey(true);
  }
}
```

DevamsizlikYaptiMi propertysine “True” değerini set ettiği satırdan sonra öğrencinin annesinin, babasının ve öğretmeninin devamsızlık yaptığından haberi olduğu mesajı sonucu alınacaktır.

c) Bir örnekle kullanımını açıklayınız.

Yukarıda da örnekle anlatmaya çalıştığıma benzer olarak burada da bir başka örnekle konuyu pekiştirelim. Bu örnekte ise bir duyuru sistemimiz olduğunu düşünelim ve bu duyuruya kayıt olan kullanıcılara duyuru yapıldığında mesaj gönderilmesi senaryosunu işleyelim.

```
Public interface Observer {

void notify(String message);
}
```

Yukarıda gözlemci interface tanımlanmıştır. Duyurular bu interface üzerinden sağlanacaktır.

```
public interface Observable
{
  void addObserver(Observer observer);
  void removeObserver(Observer observer);
  void notifyObserver();
}
```

Observable interface'ini ise duyuru sistemindeki kullanıcıların temsili olarak düşünebiliriz.

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

public class NoticeObservable implements Observable {
private List<Observer> observerList = new ArrayList<>();
private String message = "Notice... !";


    @Override
    public void addObserver(Observer observer) {
    observerList.add(observer);
                            }
    @Override
    public void removeObserver(Observer observer) {
    observerList.remove(observer);
                            }
    @Override
    public void notifyObserver() {
    for (Observer observer : observerList) {
    observer.notify(message); }}}
```

NoticeObservable sınıfı ile Observable interface'ini genişleşttik ve gerekli özelleştirmeleri yaptık.

```
public class UserMan implements Observer {



    private Observable observable;


    @Override
    public void notify(String message) {
    System.out.println(message + " UserMan Mesaj Geldi.");
    }


    public void removeObserver(){
    observable.removeObserver(this);
    }

}



public class UserWoman implements Observer {

     private Observable observable;

     public UserWoman() {
     }

     public void setObservable(Observable observable) {
         this.observable = observable;
     }

     @Override
     public void notify(String message) {
         System.out.println(message + " UserWoman Mesaj Geldi.");
     }

     public void removeObserver(){
         observable.removeObserver(this);
     }

}
```

Yukarı da görmüş olduğunuz iki tane kullanıcı sınıfımız var ikisi de Observer interface'ini implement etmiş yani artık ikisi de birer gözlemci ve Duyuru yapıldığında notify() metoduna gelen parametre ile duyuru okunacaktır.

```
public class Main {

    public static void main(String[] args) {

        UserMan userMan = new UserMan();
        UserWoman userWoman = new UserWoman();

        NoticeObservable noticeObservable = new NoticeObservable();

        userWoman.setObservable(noticeObservable);

        noticeObservable.addObserver(userMan);
        noticeObservable.addObserver(userWoman);
        noticeObservable.notifyObserver();

        userWoman.removeObserver();
        noticeObservable.notifyObserver();

    }
}
```

Duyuru ile ilgili bir senaryo ile anlatmaya çalıştığım örnek ise bu şekilde çalışmaktadır.

d) Bu tasarım şablonunu kullanan örnek bir program veriniz.

Yukarıdaki başlıklarda açıklamaya ve örneklerle göstermeye çalıştığım üzere aslında observer pattern yani gözlemci tasarım şablonunun üyelik yönetimi olan tüm yazılımların ilgili modüllerinde kullanılması gerektiğini düşünülebiliriz. Asp.net Membership sınıfını Observer Pattern kullanan örnek bir program olarak gösterebiliriz. Ayrıca en güzel Observer Pattern örneği olarak MVC yapısını gösterebiliriz. Son zamanlarda Microsoft Visual Studio içerisinde yer almasından dolayı .Net teknolojisi olarak anılsa da aslında MVC yani Model View Controller yapısı çok eskidir ve Java ile temelleri atılmış bir yapıdır. Bu yapı da ister Java olsun ister .net çatısı altında olsun geliştirilen yazılımlarda Model veri tabanını, View arayüzü ve Controller ise yapı taşı fonksiyonel programlamanın yapıldığı yerdir ve bu üç katman arasında çok güzel bir etkileşim vardır. Bu yüzden en güzel observer örneği MVC olarak gösterilebilir.

Yine bir başka örnek olarak güncel olarak observer pattern yapı taşı ile çalışan ve hemen herkesin anımsayacağı üzere Facebook da bir gruba üye olduğumuzda grup bildirimlerini açtığımızda bize ve diğer tüm abonelere gidecek olan bildirimlerin observal olduğunu söyleyebiliriz.

İkinci Kısım Anti Pattern : Golden Hammer, Wolf Ticket, Intellectual Violence

a) Bu Antipattern'I ortaya çıkaran kök neden (root cause) nedir?

Golden Hammer Anti Pattern

Bu antipattern'i ortaya çıkaran kök neden literatürde Ignorance, Pride, Narrow-Mindedness olarak bilinmektedir. Yani bilgisizlik, ilgisiz olma veya dar görüşlülük anlamlarına gelen kök sebeplerden oluşmaktadır.

Wolf Ticket Anti Pattern

Wolf Ticket mimari AntiPatternler, mimarlığın ortaya çıkarılmasında, uygulanmasında ve yönetiminde bazı yaygın sorunlara ve hatalara odaklanmaktadır.

Intellectual Violence Anti Pattern

Entelektüel Şiddet olarak bilinen bu antipattern'in çıkış sebebi bir proje içerisinde bulunan insanların yeni bir kavramı anlayamadığında bu durumdan utanmalarından kaynaklı bu duygu ile uğraşırken konuyu tamamen ortadan kaldırmaları veya bilgiyi paylaşmak yerine kontrol etmeye çalışmaları ve gizlemelerinden dolayı ortaya çıkan problemlerdir.

Entelektüel Şiddet, bir teoriyi, teknolojiyi anlayan birinin, bu bilgiyi, bir toplantı durumunda başkalarını sindirmek için kullandığı zaman ortaya çıkar. Bu, teknik insanların, cehaletlerini açığa vurmalarına yönelik normal retikliğinden dolayı yanlışlıkla meydana gelebilir.

b) Bu Antipattern'e verilen başka isimler varsa nelerdir?

Golden Hammer Antipattern'i: Old Yeller, Head-in-the sand isimleri ile de bilinmektedir.

Wolf Ticket Antipattern'i: Bilinen yaygın bir farklı ismi yoktur.

Intellectual Violence Antipattern'i: Bilindn yaygın bir farklı ismi yoktur.

c) Bu Antipattern yazılım tasarım-seviyesi modelinde nereye karşılık gelir?

Golden Hammer Antipattern'i: yazılım tasarım seviyesi modelinde Application (uygulama) karşılık gelmektedir. Refactored çözüm türü ise process olarak bilinir. Bu antipattern Yazılım Geliştirme antipattern'leri arasında yer almaktadır. (Software Development AntiPattern)

Wolf Ticket Antipattern'i: yazılım tasarım seviyesi modelinde Software Architecture AntiPattern yani yazılım mimari antipattern'leri arasında yer almaktadır.

Intellectual Violence Antipattern'i: yazılım tasarım seviyesi modelinde Project Management AntiPattern yani proje yönetimi antipattern'leri arasında yer almaktadır.

d) Bu Antipattern hangi problem sınıfına bir çözüm önermektedir?

Golden Hammer Antipattern'i: Process sınıfına bir çözüm üretmektedir. Yani refactored solution typle process olarak bilinmektedir. Golden Hammer bir ürünü, teknolojiyi ya da spesifik bir mimari modeldeki süreçleri temsil eder. Ayrıca geliştirme sürecindeki yaygın tuzaklara çözüm önermektedir. Bir geliştirme antipattern'i olan Golden Hammer yani altın çekiç, bir sistemin yeniden düzenlenebilmesinin yollarını tanımlamaktadır. En temel anlamda bu antipattern programcıların karşılaştıkları problemlere çözüm önermektedir.

Wolf Ticket Antipattern'i: Ürüne bağlı arayüzler ve uygulama yazılımının çoğu arasında bir yalıtım katmanı sağlamak, karmaşıklık ve mimarinin yönetimini sağlar.

Intellectual Violence Antipattern'i: Bir geliştiricinin belirsiz teorilere, ezoterik standartlara ve kısa vadeli kazanım için test edilmemiş teorilere bağlı kalması gerektiğindeki sorunlara çözüm önermektedir.

e) Bu Antipattern hakkında detaylı bilgi veriniz.

Golden Hammer Antipattern: bu antipattern'de favori bir çözüm yolunun evrensel anlamda kabul gördüğünü varsaymak yanılgısı söz konusudur. Daha önceden başarılı bir şekilde uygulanmış bir çözümün, sonraki problemlerde de kullanılmaya çalışılması olarak düşünülebilir. Oysaki bazı problemler aynı yöntemler ve yaklaşımlar ile çözümlenemeyebilir. Bu biraz da çözümü arayan kişilerin daha önce başarılı bir şekilde uyguladıkları yaklaşımları sahiplenmesinden kaynaklanmaktadır.

Örneğin tüm yazılımların SOA (Service Orienter Architecture) mimari bütünü içerisinde ele alınması gerektiğini düşünmek bu duruma örnek olarak verilebilir. En sık görülen antipattern'ler arasında yer alır. Söz konusu durumun oluşmasının nedenlerinden birisi teknolojik gelişmelerden ekiplerin haberdar olmamasıdır. Bazı firmalar gerekli eğitimlerin getirdiği ek maliyetlerden kaçınır ve ekibi gerekli teknik eğitim ile donatmaz. Hatta bazı ekiplerin bireysel anlamda gelişmenin önünü açacak aktivite ve çalışmalara müsaade etmemesi de bu sebepler arasında sayılabilir.

Wolf Ticket Antipattern: Bir Wolf Ticket, uygulanabilir bir anlamı olmayan standartlara açıklık ve uygunluk iddia eden bir üründür. Ürünler, yayınlanan standarttan önemli ölçüde değişebilen patentli arayüzlerle teslim edilir. Burada önemli bir sorun, teknoloji tüketicilerinin genellikle açıklığın bazı faydalar ile geldiğini varsaydıklarıdır. Standartlar, teknoloji geçiş maliyetlerini düşürür ve teknoloji istikrarını artırır, ancak standartların uygulanmasındaki farklılıklar, genellikle, birden çok kuruluşla birlikte çalışabilirlik ve yazılım taşınabilirliği gibi varsayılan faydalarını ortadan kaldırır.

Uygunluğu sağlamak için mekanizmalar olduğundan çok daha fazla bilgi sistemi standardı vardır. Bilgi sistemleri standartlarının sadece yüzde 6'sı test süitlerine sahiptir. Test edilebilir standartların çoğu, FORTRAN, COBOL, Ada ve benzeri dil derleyicilerinin programlanması içindir. Standartlar, teknoloji geçiş maliyetlerini düşürür ve teknoloji istikrarını artırır, ancak standartların uygulanmasındaki farklılıklar, genellikle, birden çok kuruluşla birlikte çalışabilirlik ve yazılım taşınabilirliği gibi varsayılan faydalarını ortadan kaldırır. Ayrıca, birçok standart özellikleri birlikte çalışabilirliği ve taşınabilirliği sağlamak için çok esnektir; diğer standartlar aşırı derecede karmaşıktır ve bunlar ürünlerde eksik ve tutarsız olarak uygulanır. Sıklıkla, standardın farklı alt kümeleri çeşitli satıcılar tarafından uygulanır. Wolf Biletleri, fiili standartlar için (popüler kullanım veya piyasaya maruz kalma yoluyla oluşturulan gayri resmi bir standart) önemli bir sorundur. Ne yazık ki, bazı fiili standartların etkili bir özelliği yoktur; Örneğin, her satıcısına özgü, çoklu tescilli arayüzlerle ticari olarak temin edilebilen, yeni bir veri tabanı teknolojisi, fiili bir standart haline gelmiştir.

Intellectual Violence Antipattern: bir teoriyi, teknolojiyi anlayan birinin, bu bilgiyi, bir toplantı durumunda başkalarını sindirmek için kullandığı zaman ortaya çıkar. Bu, teknik insanların, cehaletlerini açığa vurmalarına yönelik normal retikliğinden dolayı yanlışlıkla meydana gelebilir. Kısacası, Entellektüel Şiddet, iletişimin bir dökümüdür. Bir projedeki bazı ya da çoğu insan yeni bir kavram anlayamadığında, aşağılık duygusu ile uğraşırken ya da konuyu tamamen ortadan kaldırdıkça, ilerleme süresiz olarak durdurabilir. Entelektüel Şiddet yaygın olduğunda, üretkenliği engelleyen bir savunma kültürü ortaya çıkar. İnsanlar paylaşmak yerine bilgiyi kontrol eder ve gizlerler.