Nesne Tabanlı Programlama
Bölüm 12
Design Pattern (Tasarım Kalıpları)

Dr. Öğr. Üyesi Murat TAŞYÜREK (kayubmprogramlama1@gmail.com)

25 Aralık 2023

Kayseri Üniversitesi, Bilgisayar Mühendisliği Bölümü

Ders İçeriği

- Design Pattern (Tasarım Kalıpları)
- Design Pattern Types (Tasarım Desenlerinin Çeşitleri)
- Örnekler

Design Pattern (Tasarım Deseni)

- **Design Pattern (Tasarım Deseni)** Tasarım desenleri; yazılım geliştirirken karşımıza sıkça çıkan ortak sorunları çözmek için oluşturulmuş desenlerdir.
- Yazılım tasarımında ortaya çıkan yaygın sorunlara karşı en basit ve en efektif biçimde yeniden kullanılabilir çözümler sağlar.
- Her desen, çevremizde tekrar tekrar ortaya çıkan bir sorunu açıklar ve daha sonra bu soruna çözümün uygulanmasını, bu çözümü bir kez uygulayarak milyonlarca kez kullanabileceğiniz şekilde tanımlar.

Design Pattern

- Tasarım desenleri, tasarım kalıpları, tasarım örüntüleri veya tasarım şablonları, çok rastlanan, birbirine benzer sorunları çözmek için geliştirilmiş ve işlerliği kanıtlanmış genel çözüm önerileri olarak da tanımlanır.
- 1994 yılında Gang of Four (GOF) olarak bilinen Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Design Patterns — Elements of Reusable Object-Oriented Software adında kitap yayınlamışlar ve ilk kez yazılım alanında tasarım kalıpları kavramını ortaya atmışlar.
- Yaygın olarak bilin 23 adet (kitapda anlatılan) design pattern bulunmaktadır.

Design Pattern

- Yazılım tasarımı yaparken bir problem ortaya çıkarsa, hangi yolu kullanarak çözebilirim diye düşünülmeli ve uygun pattern(kalıp) bulunmalıdır.
- Bazı durumlarda kendi **desing pattern**'inizi oluşturabilir ve kendi çözümünüzü uygulayabilirsiniz.
- Ancak, daha önceden bilinen bir bir **desing pattern**'i kullanmak avantajlı olacaktır.

Design Pattern avantajları

- Bilinen problemler için genel bir çözümdür.
- Kanıtlanmış çözümlerdir.
- Etkileyicidirler ve işlerin bakımını kolaylaştırırlar.
- Birçok geliştirici (orta ve üzeri) Design Patterns'a aşinadır ve bu sayede bir yazılım geliştirme standardı olarak karşımıza çıkmaktadır.
- **Design Patterns**, test edilmiş, kanıtlanmış geliştirme paradigmaları sağlayarak geliştirme sürecini hızlandırabilir.
- Büyük sorunlara neden olabilecek ince sorunları önlemeye yardımcı olur ve ayrıca kod okunabilirliğini artırır.

Design Pattern Türleri

Yazılım tasarım kalıpları genel olarak 3 kategoriye ayrılarak incelenmektedir.

- **Creational Patterns (Oluşturucu Desenler)**: Nesnelerin oluşturulmasında ve yönetilmesinde kullanılan bir desendir. Bu desen tasarımları program akışında hangi nesneye ihtiyaç varsa onu oluşturmada esneklik ve kolaylık sağlar.

- **Structural Patterns (Yapısal Desenler):** Birden fazla sınıfın bir işi yerine getirirken nasıl davranacağını belirlemek için kullanılan desenlerdir.

- **Behavioral Patterns (Davranışsal desenler)**: Nesnelerin birbirleri ile ilişkisini düzenleyen desendir.

Creational Pattern Türleri

- **Singleton**: Bir sınıfın sadece bir örneği olmalıdır ve bu örneğe global bir erişim noktası sağlanmalıdır.
- **Factory:** Birbirleri ile ilişkili nesneleri oluşturmak için bir arayüz sağlar ve alt sınıfların hangi sınıfın örneğini oluşturacağına olanak sağlar.
- **Abstract Factory:** Birbirleri ile ilişkili ürün ailesini oluşturmak için kullanılır.
- **Builder**: Karmaşık yapıdaki nesnelerin oluşturulmasında istemcinin sadece nesne tipini belirterek üretimi gerçekleştirebilmesini sağlamak için kullanılan bir desendir. Bu desende istemcinin kullanmak istediği gerçek ürünün birden fazla sunumunun olduğu durumlarda kullanılır.
- **Prototype**: mevcut nesnelerin prototiplerinin oluşturulmasını yani nesnelerin kopyalarını elde etmeyi sağlayan bir tasarım desenidir.

Örnek sorun

- Elimizde bir veritabanımız var ve veritabanı aynı anda sadece bir bağlantıya izin veriyor.
- Sistemde bir çok yazılım geliştirimiz ve kullanıcımız var.
- Aynı anda sadece bir kullanıcının bağlanmasını istiyoruz ve bağlantıların tek bir yerden yönetilmesini istiyoruz.
- Bunun için bilinen bir çözüm (pattern) var mı ? (Singleton)

Singleton Pattern

- Sadece bir nesne olmalıdır ve bu öğe global olmalıdır.
- Bu nedenle sınıf içerisindeki kurucu fonksiyon private olmalıdır.
- Nesnenin global olması için statik olarak tanımlanması gerekir.
- Nesneye erişimin sınırlı olması için de Geriye nesneyi dönderen özel bir fonksiyonu olmalıdır.
- Kod üzerinde inceleyelim.

Singleton Pattern Class

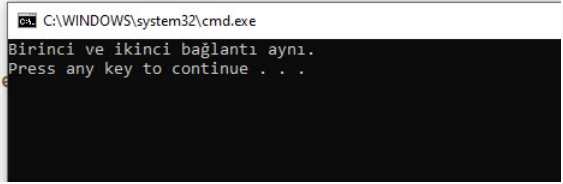
```
8 references
public class Database
{
    private static Database db;

    1 reference
    private Database()
    {
        //connection string vs. tanımlandı. bağlantı yapıldı.
    }
    2 references
    public static Database GetInstance()
    {
        if (db == null)
        {
            db = new Database();
        }
        return db;
    }
}
```

Source Kod

```
namespace NNDers11Ornek1SingilationPattern
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Database firstInstanceRequest = Database.GetInstance();

            Database secondInstanceRequest = Database.GetInstance();

            if (firstInstanceRequest == secondInstanceRequest)
            {
                Console.WriteLine("Birinci ve ikinci bağlantı aynı.");
            }
            else
            {
                Console.WriteLine("Birinci ve ikinci bağlantı farklı.");
            }
        }
    }
}
```

Çıktı

```
0 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        Database firstInstanceRequest = Database.GetInstance();

        Database secondInstanceRequest = Database.GetInstance();

        if (firstInstanceRequest == secondInstanceRequest)
        {
            Console.WriteLine("Birinci ve ikinci bağlantı aynı.");
        }
        else
        {
            Console.WriteLine
        }
    }
}
```



Structural Pattern Türleri

- **Adapter**: Bir sınıfın arayüzünü istemcinin beklediği arayüze çevirmek için kullanılır.
- **Bridge:** Soyutlanmış (abstract) yapıyı implementasyonundan (uygulamasından) ayırmak, bağımsız olarak geliştirilebilir iki yapı elde etmek için kullanılır.
- **Composite:** Nesneleri ağaç yapısına göre düzenleyerek ağaç yapısındaki alt üst ilişkisini kurmaya yarayan bir desendir.
- **Decorator**: Nesnelere dinamik olarak yeni sorumluluklar atamamızı sağlayan tasarım desenidir.
- **Facade**: Bir alt sistemdeki arayüzlere bir birleşik arayüz sağlayarak alt sistemin kullanımını daha kolay hale getirmeyi amaçlar.
- **Flyweight**: Sık kullanılan nesnelerin bellek yönetimini kontrol etmek için kullanılan bir tasarım desenidir.
- **Proxy**: istemcinin orijinal nesneye direkt erişimi yerine bu erişimi nesneyi temsil eden proxy (vekil) sınıflar üzerinden gerçekleştirmesini ve bu proxy (vekil) sınıfların sunduğu imkanları kullanmasını sağlayan tasarım desenidir.

Örnek sorun

- Online siparişlerin yapıldığı bir e-ticaret sistemi için ilgili metotların yazılması gerekiyor.
- Ancak burada kullanıcılar sipariş yapmadan önce kullanıcıların gerekli şartı sağlayıp sağlamadığı kontrol edilmek isteniyor.
- Ancak gerekli şartları sağlayan kullanıcılar için sipariş verilmeli.
- Kullanıcılar ön kontrolden gerçmeli ve sipariş metoduna direk erişimi olmamalı.
- Bunun için bilinen bir çözüm (pattern) var mı ? (Proxy)

Singleton Pattern

- Kişi için ayrı bir sınıf olmalıdır.
- Yapılacak işlemin belirtildiği bir interface olmalıdır.
- Sipariş işleminin yapıldığı bir sınıf olmalıdır.
- Kullanıcı ön kontrollerinin yapıldığı bir proxy sınıfı olmalıdır. Bu sınıfta sipariş işlemi sınıfından bir nense oluşturulmalı ve dışarı erişime kapalı olmalıdır.
- Kullanıcı sipariş vermek istediğinde proxy'nin yapacağı kontrolü geçerse sipariş verebilmelidir.
- Kod üzerinde inceleyelim.

Kisi Class ve Siparis Interface

```
6 references
public class Kisi
{
    public string adi, soyadi;
    public long TC;
    public string mail, adres;
}
3 references
interface ISiparis
{
    4 references
    void SiparisOlustur(Kisi k);
}
```

Sipariş Class

```
/*Siparişin oluşturulduğu sınıf. İstemcinin direk erişmesini istemiyoruz.
Bu neden özel bir sınıf yapıyoruz ve araya başka bir sınıf koyup kontrol ediyoruz.*/
2 references
public class SiparisOlusturma : ISiparis
{
    3 references
    public void SiparisOlustur(Kisi k)
    {
        Console.WriteLine(String.Format("{0} adına {1} adresine sipariş oluşturuldu " +
            "ve {2} adresine mail atıldı.",k.adi,k.adres,k.mail));
    }
}
```

Proxy Class

```
/*Sipariş doğrulama sınıfımız. İstemcinin muhatap olacağı sınıftır.
 Proxy sınıfıdır. Gerkek işi yapan SiparisOlusturma oluşturma sınıfının referansını tutar.
 Sipariş verilmeden önce gerekli kontrol işlemi yapılır.*/
2 references
class SiparisKontrolProxy : ISiparis
{
    private SiparisOlusturma _spr;

    1 reference
    public SiparisKontrolProxy()
    {
        _spr = new SiparisOlusturma();
    }
    2 references
    public void SiparisOlustur(Kisi k)
    {
        if (GercekKisimi(k))
        {
            _spr.SiparisOlustur(k);
        }
    }
    1 reference
    private bool GercekKisimi(Kisi k)
    {
        return k.TC.ToString().Length == 11 && k.adres != null && k.adi != null;
    }
}
```

Source Kod

```
namespace NNDers11Ornek2ProxyPattern
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Kisi mrt = new Kisi
            {
                adi = "Murat",
                soyadi = "Taşyürek",
                adres = "Kayseri",
                TC = 12345678910,
                mail = "murattasyurek@kayseri.edu.tr"
            };

            ISiparis sprs = new SiparisKontrolProxy();
            sprs.SiparisOlustur(mrt);
        }
    }
    6 references
    public class Kisi...
    3 references
    interface ISiparis...

    /* Siparişin oluşturulduğu sınıf. İstemcinin direk erişmesini istemiyoruz. ...
    2 references
    public class SiparisOlusturma ...
```

Çıktı

```
namespace NNDers11Ornek2ProxyPattern
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Kisi mrt = new Kisi
            {
                adi = "Murat",
                soyadi = "Taşyürek",
                adres = "Kayseri",
                TC = 12345678910,
                mail = "murattasyurek@kayseri.edu.tr"
            };

            ISiparis sprs = new SiparisKontrolProxy();
            sprs.SiparisOlustur(mrt);
        }
    }
    6 refere
    publi
    3 refere
    inter
```

```
C:\WINDOWS\system32\cmd.exe
Murat adına Kayseri adresine sipariş oluşturuldu ve murattasyurek@kayseri.edu.tr adresine mail atıldı.
Press any key to continue . . .
```

Behavioral Pattern Türleri

- **Chain of Responsibility**: Bir amaca yönelik bir dizi işlemi gerçekleştiren nesnelerin birbirinden bağımsız bir şekilde çalışmasını ve herbir nesnenin sadece kendisiyle tanımlı işleri yapmasını sağlayan bir tasarım desenidir.
- **Command:** Kullanıcı isteklerini gerçekleştiren kod yapısını sarmallayarak nesneler halinde saklanmasını daha sonra da bu isteklerin gerçekleştirilmesini veya geri alınmasını sağlayan tasarım desenidir.
- **Interpreter:** Belli bir düzen veya kurala göre sıralanmış verilerin, yorumlanarak istenilen çıktı üretmesini sağlar.
- **Iterator**: Koleksiyon üzerindeki elemanların üzerinde dolaşmak için kullanılan tasarım desenidir..
- **Mediator**: Aynı tipteki veya aynı arayüzü uygulayan nesneler arasında iletişimi sağlayan tasarım desenidir.
- **Memento**: Bir nesnenin önceki durumunu kaydetmemizi ve istenildiği takdirde eski haline dönmemizi sağlayan tasarım desenidir.

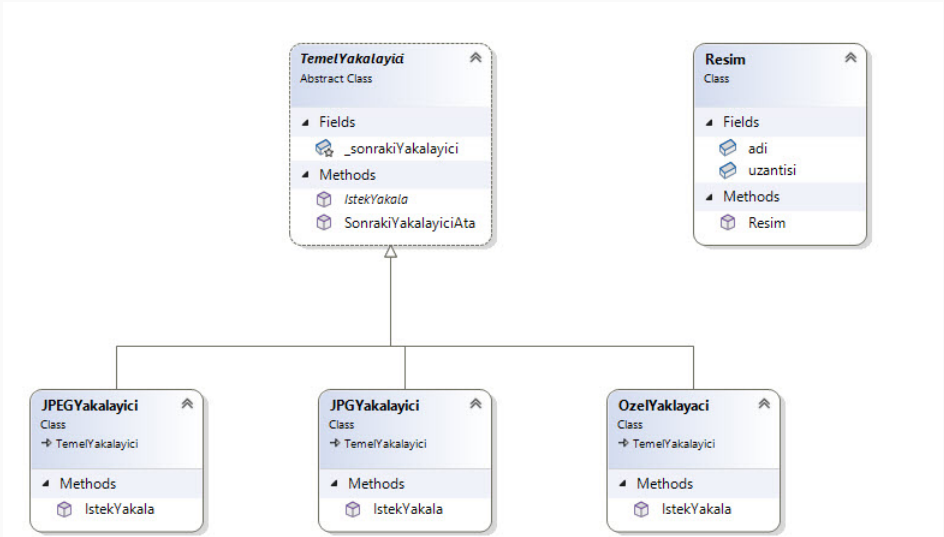
Behavioral Pattern Türleri

- **Observer**: Bir nesnede meydana gelen değişikliği içinde bulunduğu listedeki tüm elemanlara bildiren tasarım desenidir.
- **State:** Bir nesnenin iç durumu değiştiğinde meydana gelecek değişimler sonrası çalışma zamanında dinamik olarak farklı davranışları sergileyebilmesini sağlayan bir tasarım desenidir.
- **Strategy:** Bir algoritma ailesi tanımlamamızı, her birini ayrı bir sınıfa koymamızı ve nesnelerinin birbiriyle değiştirilebilir hale getirmenizi sağlayan davranışsal bir tasarım modelidir.
- **Template Method**: Üst sınıflarda bir algoritma iskeleti tanımlamamızı ve alt sınıfların algoritma iskeleti yapısını bozmadan belirli adımları yeniden tanımlamasını (override) sağlayan tasarım desenidir.
- **Visitor**: Üzerinde çalıştığımız nesnelerin sınıflarını değiştirmeden yeni bir özellik tanımlamamızı sağlayan bir tasarım desenidir.

Problem (Örnek)

- Elimizde bir çok fotoğraf var.
- Bu fotoğarları yakalamak ve hepsini png formatında kaydetmek istiyorum.
- Bunu kodlamamız gerekiyor.
- Chain of Responsibility deseni en uygunudur.

Chain of Responsibility UML diyagramı



Abstract Class, Resim Class

```
7 references
class Resim
{
    public string adi;
    public string uzantisi;

    1 reference
    public Resim(string adi, string uzantisi)
    {
        this.adi = adi;
        this.uzantisi = uzantisi;
    }
}
5 references
abstract class TemelYakalayici
{
    protected TemelYakalayici _sonrakiYakalayici;
    2 references
    public void SonrakiYakalayiciAta(TemelYakalayici sonrakiYakalayici)
    {
        _sonrakiYakalayici = sonrakiYakalayici;
    }

    6 references
    public abstract void IstekYakala(Resim rsm);
}
```

JPEG Class, JPEG Class

```
2 references
class JPEGYakalayici : TemelYakalayici
{
    4 references
    public override void IstekYakala(Resim rsm)
    {
        Console.WriteLine("Resim formatı:" + rsm.uzantisi);
        if (rsm.uzantisi == "JPEG")
        {
            Console.WriteLine("JPEG to PNG");
            // JPEG PNG formatında dönüştürülür. Ekrana bilgi yazdırılır.
        }
        else
        {
            Console.WriteLine("JPEG yakalayıcı yakalayamdı bir sonraki yakalaycıya yönlendirdi.\n");
            // Bu sınıfa ait bir işlem değilse zincirin bir sonraki halkasına aktarılır.
            _sonrakiYakalayici.IstekYakala(rsm);
        }
    }
}
2 references
class JPGYakalayici : TemelYakalayici
{
    3 references
    public override void IstekYakala(Resim rsm)
    {
        Console.WriteLine("Resim formatı:" + rsm.uzantisi);
        if (rsm.uzantisi == "JPG")
        {
            Console.WriteLine("JPG to PNG");
            // JPG görüntünüsü PNG görünütüsüne dönüştürülür.
        }
        else
        {
            Console.WriteLine("JPG yakalayıcı yakalayamdı bir sonraki yakalaycıya yönlendirdi.\n");
            // Bu sınıfa ait bir işlem değilse zincirin bir sonraki halkasına aktarılır.
            _sonrakiYakalayici.IstekYakala(rsm);
        }
    }
}
```

Diğer Uzantılar için Class

```
// Herhangi bir tipteki dosyayı dönüştüren sınıf.
2 references
class OzelYaklayaci : TemelYakalayici
{
    3 references
    public override void IstekYakala(Resim rsm)
    {
        Console.WriteLine("Resim formatı:" + rsm.uzantisi);
        if (rsm.uzantisi == "OtherExtension")
        {
            Console.WriteLine("OtherExtension to PNG");
            // Herhangi bir tipteki dosyanın PGN formatına dönüştürme işlemi
        }
        // Burada else ifadesi bulunmamaktadır.
        // Else ifadesi yoksa zincirin son halkası olduğu anlamına gelir.
    }
}
```

Source Kod

```
using System.Text;
using System.Threading.Tasks;
using static System.Net.Mime.MediaTypeNames;

namespace NNDers11Ornek3ChainofResponsibility
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Resim rsm = new Resim("öğrencilik fotoğrafım", "OtherExtension");

            JPEGYakalayici jpegYakala = new JPEGYakalayici();
            JPGYakalayici jpgYakala = new JPGYakalayici();
            OzelYaklayaci digerTurkleriYakala = new OzelYaklayaci();

            // Zincirlerin birbirlerine bağlanmalı
            jpegYakala.SonrakiYakalayiciAta(jpgYakala);
            jpgYakala.SonrakiYakalayiciAta(digerTurkleriYakala);

            jpegYakala.IstekYakala(rsm);
        }
    }
    7 references
    class Resim...
    5 references
    abstract class TemelYakalayici...
    2 references
    class JPEGYakalayici ...
    2 references
    class JPGYakalayici ...
    // Herhangi bir tipteki dosyayı dönüştüren sınıf.
    2 references
    class OzelYaklayaci ...
}
```

Çıktı

```
0 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        Resim rsm = new Resim("öğrencilik fotoğrafim", "OtherExtension");

        JPEGYakalayici jpegYakala = new JPEGYakalayici();
        JPGYakalayici jpgYakala = new JPGYakalayici();
        OzelYaklayaci digerTurkleriYakala = new OzelYaklayaci();

        // Zincirlerin birbirlerine bağlanmala
        jpegYakala.Sonr
        jpgYakala.Sonra

        jpegYakala.Iste
    }
}
7 references
class Resim...
5 references
abstract class TemelYak
```

```
C:\WINDOWS\system32\cmd.exe
Resim formatı:OtherExtension
JPEG yakalayıcı yakalayamdı bir sonraki yakalaycıya yönlendirdi.

Resim formatı:OtherExtension
JPG yakalayıcı yakalayamdı bir sonraki yakalaycıya yönlendirdi.

Resim formatı:OtherExtension
OtherExtension to PNG
Press any key to continue . . .
```

Ödev

- İçerisinde ad, soyad ve tc gibi bilgilerini tutulduğu kişi bilgilerini binary ağaç üzerinde gösterilmektedir.
- İsim ve soyisim public olabilir TC nesne oluşturulurken girilmelidir.
- İlgili nesneler ağaç üzerinde gösterilirken öncelikle soyadına göre sıralanmalıdır. Soyadları aynı ise adlarına göre sağa veya sola yerleştirilmelidir.
- İlgili programı dessing pattern yapısı kullanarak kodlayınız.
- 23 tane nesneyi ağaca yerleştiriniz ve çıktısını gösteriniz.
- 1 Ocak 06.59'a kadar gönderilenler 10+2 puan, Final kadar gönderenler 10 puan (kayubmprogramlama1@gmail.com).