NESNE YÖNELİMLİ PROGRAMLAMA 2(Object Oriented Programming 2/OOP)

Öğr. Gör. Celil ÖZTÜRK
Marmara Üniversitesi
Teknik Bilimler Meslek Yüksekokulu

İçerik

- ✓Yapısal Kalıplar(Structural Patterns)
  - Adapter
  - Facade

Yapısal Kalıplar(Structural Patterns)

- Yapısal tasarım kalıpları, farklı sınıfların ve nesnelerin daha büyük yapılar oluşturulması için nasıl birleştirileceğine dair bir taslak sunar.
- Çoğunlukla aynı temel amaç için farklı yolların kullanıldığı Creational kalıpların aksine, her bir yapısal kalıp farklı bir amaca sahiptir.

- Yapısal kalıpların tümü**, nesnelerin arasındaki bağlantıları geliştirir.**
- Yapısal tasarım kalıpları bileşenlerin veya modüllerin yapı içerisinde nasıl düzene gireceklerini açıklarlar.

Yapısal Kalıplar(Structural Patterns)

- Yapısal tasarım kalıpları ayrıca kalıp boyunca verinin nasıl hareket edeceğini de açıklar.
- Yapısal tasarım kalıpları, sistemin esnek olabilmesi için bileşenlerin nasıl yapılandırılması gerektiğini tanımlarlar.
- Yapısal kalıplar, sınıflar ve nesnelerin birleştirilerek daha geniş yazılım gruplarının kurulmasına olanak sağlayan öneriler sunar.
- Yapısal kalıplar ayrıca yazılım mühendislerinin uygulamalar geliştirdikleri projelerinde ihtiyaç duydukları çesitli durumlarda kullanılmak üzere yediye ayrılır.
- Yapısal kalıplar yazılım projelerinin tasarım aşamalarında veya bu projelerin genişletilme aşamalarında kullanılabilirler.

Tasarım Kalıpları

Structural Patterns(Yapısal Kalıplar)

- Adapter Pattern
- Bridge Pattern
- Composite Pattern
- Decorator Pattern
- Facade Pattern
- Flyweight Pattern
- Proxy Pattern

Adapter Pattern

- Nesneye yönelik programlamanın sıklıkla sözü geçen avantajlarından birisi de kodun yeniden kullanabilir olmasını sağlamasıdır.
- Verilerin ve davranışların bir sınıf merkezinde toplandığı bir sınıfın başka bir projeye aktarılması sonrası sınıfın fonksiyonelliği çok az bir maliyetle kullanılabilmektedir.
- Bir projenin gelecekteki kodlama gereksinimlerinin ne olacağı bilinmediğinden dolayı her zaman yeniden kullanılabilir bir sınıfın tasarımının nasıl olması gerektiğini bilemeyiz.

Adapter Pattern

- Benzer bir projede çalışmakta olan kişi, size ticari uygulamanın bir bileşeni olan adres sistemini sağlayabilmektedir.
- Siz bu dosyaları aldığınızda, projenizde kullandığınız arayüzler ile dosyadaki arayüzlerin eşleşmediğini fark ettiniz.
- Burada ihtiyaç olan şey ise bir tercümandır.
- İşte bu da tam olarak **Adapter tasarım** kalıbının yaptığıdır.
- Bir güç adaptörüne benzer şekilde davranarak, bir tipi bir diğer – uyumsuz- tipe dönüştürür.

Adapter Pattern

Diğer tanımlar,

- Adaptör kalıp sadece bir sınıfa **(class)** özel olan arayüzleri diğer sınıflarla uyumlu arayüzler haline getirir. Adaptörler uyumlu olmayan arayüzler sebebiyle birbirleri ile çalışamayan sınıflara da birbirleri ile çalışma imkanı sunarlar.(bidb.itu.edu.tr)
- Sistemdeki soyutlamaları sağlamak için interface'ler abstract sınıflar kullanılır.
- Bu desen, mevcut bulunan yapıya uymayan bir sınıfı entegre etme tekniğidir.
- **Adapter Deseni, sisteme yeni bir özelliğin eklenmesini kolaylaştıran bir tekniktir.**
- Adapter Desenininde, bir adaptör sınıfı yazılır ve bu sınıftan adapte edilmeye çalışılan sınıfa bir referans vardır**.**
- Bu sayede yazılmış olan adaptör sınıfı mevcut yapıya entegre bir şekilde çalışır.

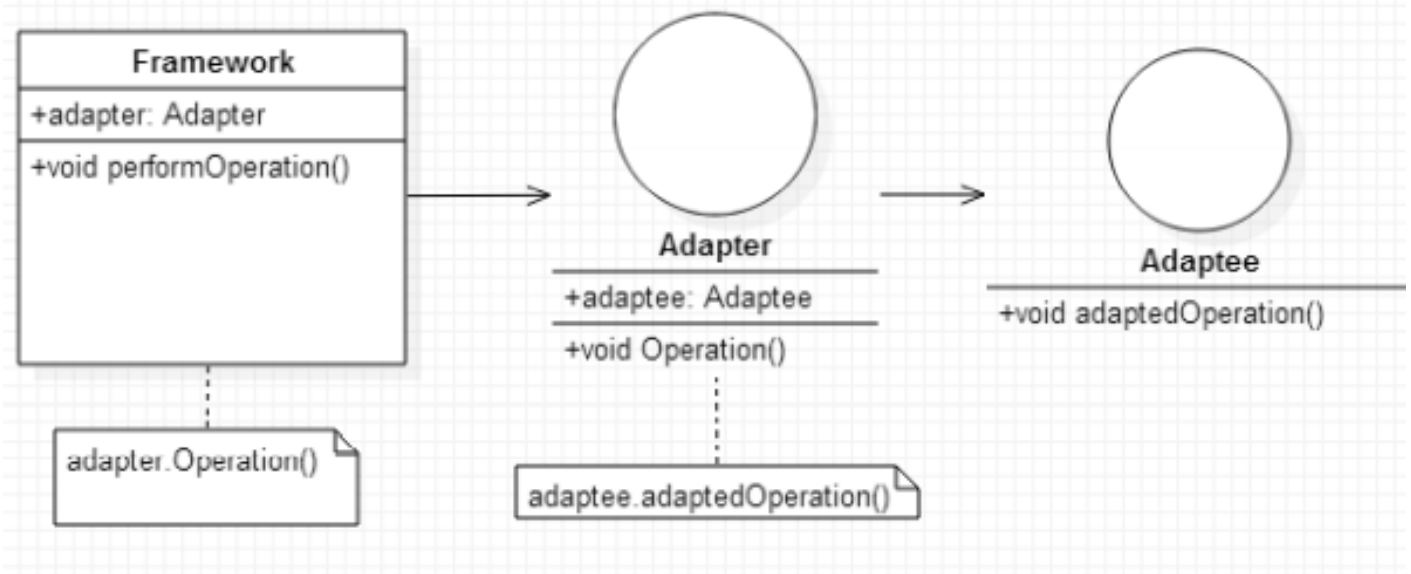
Adapter Pattern

- Adapter tasarım kalıbı

– Temelde birbiriyle uyumsuz ancak aynı işi yapması öngörülen iki interface'in haberleştirilmesi için kullanılır.

- Adapter tasarım kalıbının kullanımı, sizin uygulamalarınızın yeni bileşenlerin kullanımına izin verdiği gibi aynı zamanda sizin var olan arayüzlerinizi kullanmaya devam etmenize olanak sağlar.
- Yeni bir sürüm geldiğinde, yapmanız gereken sadece Adapter'i değiştirmektir.

Adapter Pattern



Adapter Pattern

- Framework: Adapter'i kullanan yapı.
- Adapter: Framework'un kullanacağı metotları tanımlayan arayüz.
- Adaptee: Adapte edilecek tipin metotlarını tanımlayan arayüz. Bu arayüz, çalışma zamanında dinamik olarak belirli Adaptee'nin yüklenmesine izin verir.

Adapter Pattern

- Adapter tasarım kalıbını; .mp3 uzantılı dosyaları çalabilen sıradan bir medya oynatıcısını, .mp4 ve .vlc formatını çalabilen bir medya oynatıcısına dönüştürürken kullanacağız.

- IMedyaOynatabilen arayüzüne ve somut (concrete) bir sınıf olan ve IMedyaOynatabilen arayüzünü implemente eden bir MedyaOynaticisi sınıfımız var.
- MedyaOynaticisi varsayılan olarak .mp3 uzantılı dosyaları çalabilmektedir.

Adapter Pattern

- IMedyaOynatabilen.java

```
public interface IMedyaOynatabilen {
    void Oynat(String dosyaUzantisi, String dosyaAdi );
}
```

MedyaOynaticisi.java

```
public class MedyaOynaticisi implements IMedyaOynatabilen
{
    @Override
    public void Oynat(String dosyaUzantisi, String dosyaAdi)
    {
        if (dosyaUzantisi.equalsIgnoreCase("mp3"))
            System.out.println("Mp3 uzantili " + dosyaAdi + " çalınıyor.");
        else
            System.out.println("Desteklenmeyen dosya uzantısı...");
    }
}
```

Adapter Pattern

Main.java

```
public class Program {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        MedyaOynaticisi medyaOynatıcı = new MedyaOynaticisi();

        medyaOynatıcı.Oynat("mp3", "Özkan Uğur - Olduramadım");

    }
```

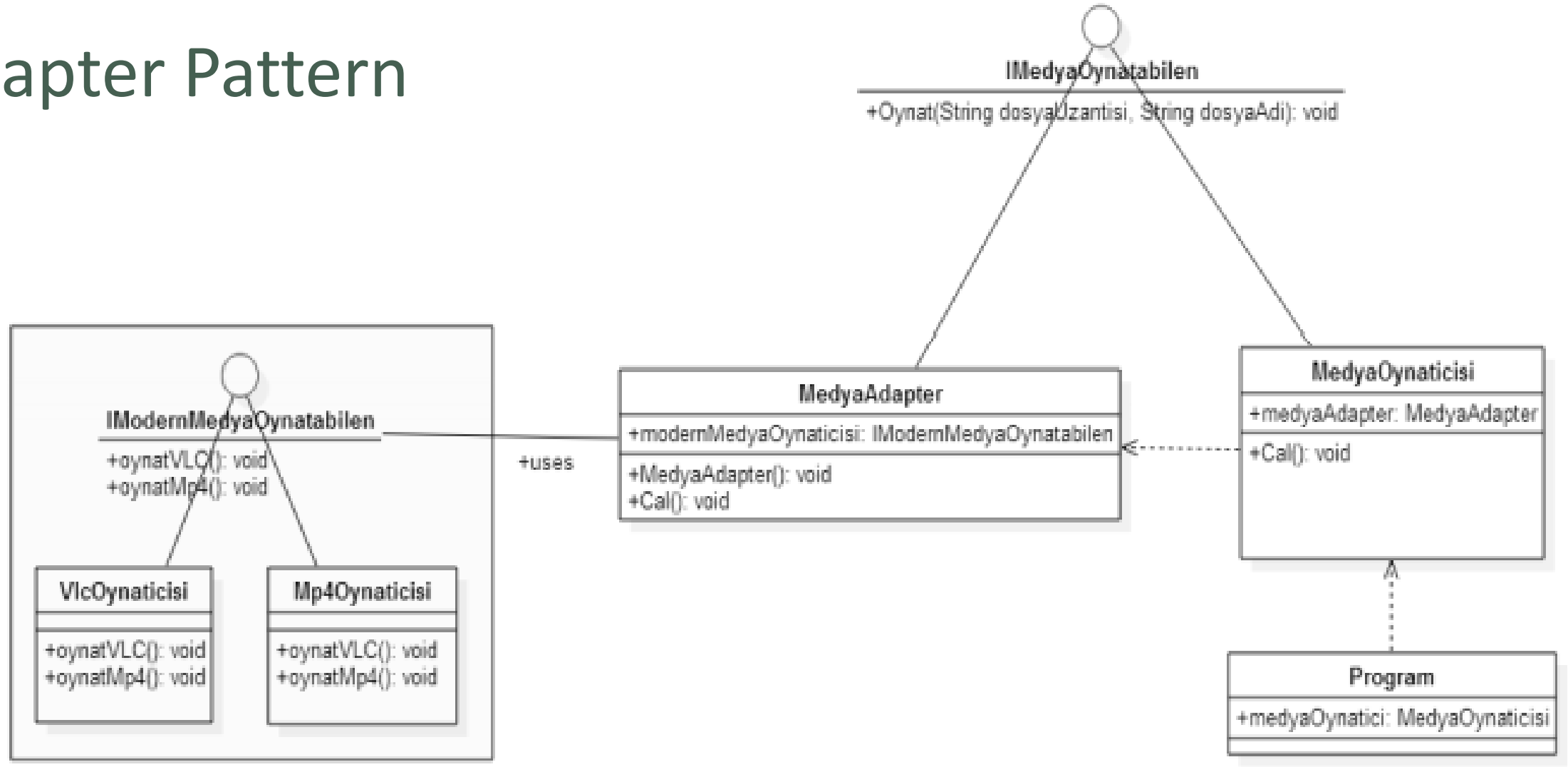
Adapter Pattern

- Biz diğer IModernMedyaOynatabilen arayüzüne ve bu arayüzü implemente eden somut sınıflara sahibiz. Bu sınıflar ayrı ayrı .vlc ve .mp4 oynatabilmektedir.
- Biz MedyaOynaticisi'nin diğer uzantıları da oynatmasını istiyoruz.
- Bunu yapabilmek için bir adapter sınıfına ihtiyaç duyarız.

Adapter Pattern

- İlk olarak Adapter sınıf olan IMedyaOynatabilen arayüzünü implemente eden ve gerekli formatta çalabilmek için IModernMedyaOynatabilen nesnelerini kullanan MedyaAdapter sınıfını yaratırız.
- MedyaOynaticisi sınıfı adapter sınıf olan MedyaAdapter sınıfını kullanarak istenilen medya tipini (bu medya tipini hangi sınıfın kullanabileceğini bilmeden) geçirmektedir.
- Program sınıfı ise çeşitli uzantıları çalmak MedyaOynaticisi'ni kullanan bir sınıftır.

Adapter Pattern



Adapter Pattern

- IMedyaOynatabilen ve IModernMedyaOynatabilen arayüzlerini yaratırız.

IMedyaOynatabilen.java

```
public interface IMedyaOynatabilen {
    void Oynat(String dosyaUzantisi, String dosyaAdi );
}
```

IModernMedyaOynatabilen.java

```
public interface IModernMedyaOynatabilen {
    void OynatVLC(String dosyaAdi);
    void OynatMP4(String dosyaAdi);
}
```

Adapter Pattern

- IModernMedyaOynatabilen arayüzünü implemente eden VlcOynaticisi ve Mp4Oynaticisi somut sınıflarını yaratırız.

VlcOynatici.java

```
public class VlcOynaticisi implements IModernMedyaOynatabilen{

    @Override
    public void OynatVLC(String dosyaAdi) {
        System.out.println("VLC uzantili " + dosyaAdi + " çalınıyor.");
    }

    @Override
    public void OynatMP4(String dosyaAdi) {
        // MP4 ile ilgili islem yapılmıyor.

    }
}
```

Adapter Pattern

- Mp4Oynaticisi.java

```
public class Mp4Oynaticisi implements IModernMedyaOynatabilen{

    @Override
    public void OynatVLC(String dosyaAdi) {
        // .vlc ile ilgili islem yapılmıyor.
    }

    @Override
    public void OynatMP4(String dosyaAdi) {
        System.out.println("MP4 uzantili " + dosyaAdi + " çalınıyor.");

    }

}
```

Adapter Pattern

- IMedyaOynatabilen arayüzünü implemente eden MedyaAdapter sınıfını yaratırız.
  - Bu sınıfa geçirilen medyanın uzantısına bağlı olarak **IModernMedyaOynaticisi** arayüzü nesneleri yaratırız

MedyaAdapter.java

```
public class MedyaAdapter implements IMedyaOynatabilen {
    IModernMedyaOynatabilen modernMedyaOynaticisi;

    public MedyaAdapter(String dosyaUzantisi) {
        if ( dosyaUzantisi.equalsIgnoreCase("vlc") )
        {
            modernMedyaOynaticisi = new VlcOynaticisi();
        }
        else if ( dosyaUzantisi.equalsIgnoreCase("mp4") )
        {
            modernMedyaOynaticisi = new Mp4Oynaticisi();
        }
    }
    @Override
    public void Oynat(String dosyaUzantisi, String dosyaAdi)
        if ( dosyaUzantisi.equalsIgnoreCase("vlc") )
        {
            modernMedyaOynaticisi.OynatVLC(dosyaAdi);
        }
        else if ( dosyaUzantisi.equalsIgnoreCase("mp4") )
        {
            modernMedyaOynaticisi.OynatMP4(dosyaAdi);
        }

    }
```

- IMedyaOynatabilen arayüzünü implemente eden MedyaOynaticisi somut sınıfını yaratırız.

MedyaOynaticisi.java

```
public class MedyaOynaticisi implements IMedyaOynatabilen{
    MedyaAdapter medyaAdapter;

    @Override
    public void Oynat(String dosyaUzantisi, String dosyaAdi) {
        // Mp3 formatını standart medya oynatıcıları
        // desteklediği için varsayılan olarak oynatabiliyor.

        if(dosyaUzantisi.equalsIgnoreCase("mp3"))
        {
            System.out.println("Mp3 uzantili " + dosyaAdi + " çalınıyor.");
        }
        else if (dosyaUzantisi.equalsIgnoreCase("mp4") || dosyaUzantisi.equalsIgnoreCase("vlc"))
        {
            medyaAdapter = new MedyaAdapter(dosyaUzantisi);
            medyaAdapter.Oynat(dosyaUzantisi, dosyaAdi);
        }

        else
        {
            System.out.println("Uygun olmayan uzantı biçimi! " +dosyaUzantisi
                               + " formatındaki dosyalar deteklenmemektedir.");
        }
    }
```

Adapter Pattern

- Program sınıfında MedyaOynaticisi nesnesi yaratıp çeşitli formattaki medyaları çalabiliriz. Program.java

```
public class Program {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        MedyaOynaticisi medyaOynatici = new MedyaOynaticisi();

        medyaOynatici.Oynat("mp3", "Özkan Uğur - Olduramadım");
        medyaOynatici.Oynat("mp4", "MFÖ - Mecburen");
        medyaOynatici.Oynat("vlc", "MFÖ - Peki peki anladık");
        medyaOynatici.Oynat("avi", "Kaç Canım Kalmış - Kafamı Hissetmiyorum");
    }

}
```

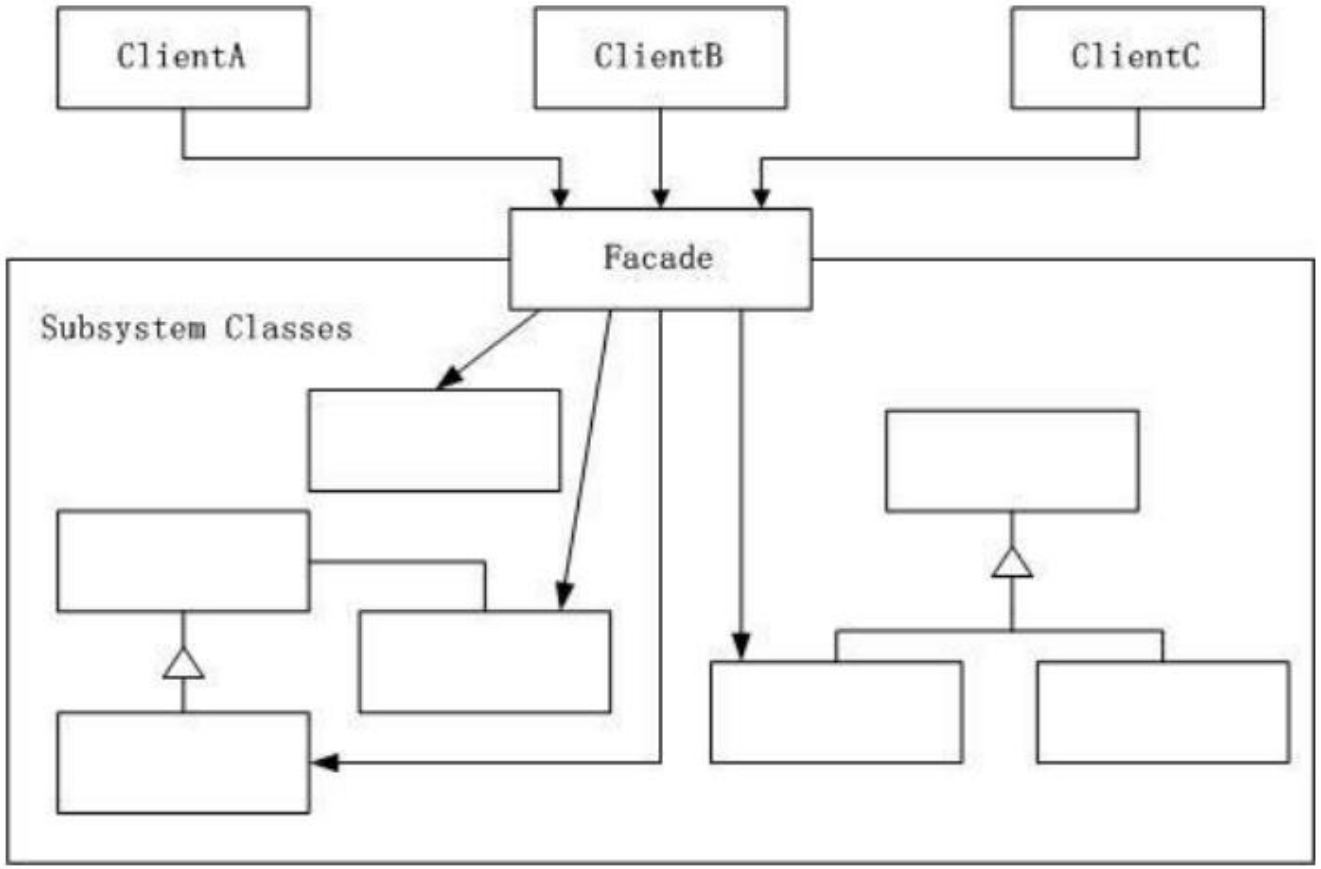
Facade (Cephe/Ön Yüz/Vitrin)

- Alt sistemlerin direk kullanılması yerine, alt sistemlerin kullanılmasını sağlayan arayüzler yazılarak kullanım kolaylaştırılır.
- Bu arayüzleri gerçekleyen sınıflara Facade sınıfları denir.
- Facade tasarım deseninde, alt sistemde birden fazla sınıf ve bu sınıflar arasında ilişkiler bulunur ve herhangi bir işlemi gerçekleştirmek için bu sınıflardaki metotları belli bir sırayla çağırmak gerekir
  - Facade, çok geniş boyutlardaki kod parçalarını onlara göre çok daha sadeleştirilmiş arayüzlere indirgenilmesini sağlar. Sınıf kütüphaneleri **(class library)** bu tür kalıplar için verilebilecek en iyi örnekleridir.(bidb.itu.edu.tr)

Facade

- Facade’nin asıl amacı alt sistemlerin saklanması değildir.
- Amacı, bir dizi alt sistem için bir ara yüz sağlamaktır; fakat daha ayrıntılı seçeneklere ihtiyaç duyan istemciler alt sistemlerle etkileşime geçebilmelidir.

Facade

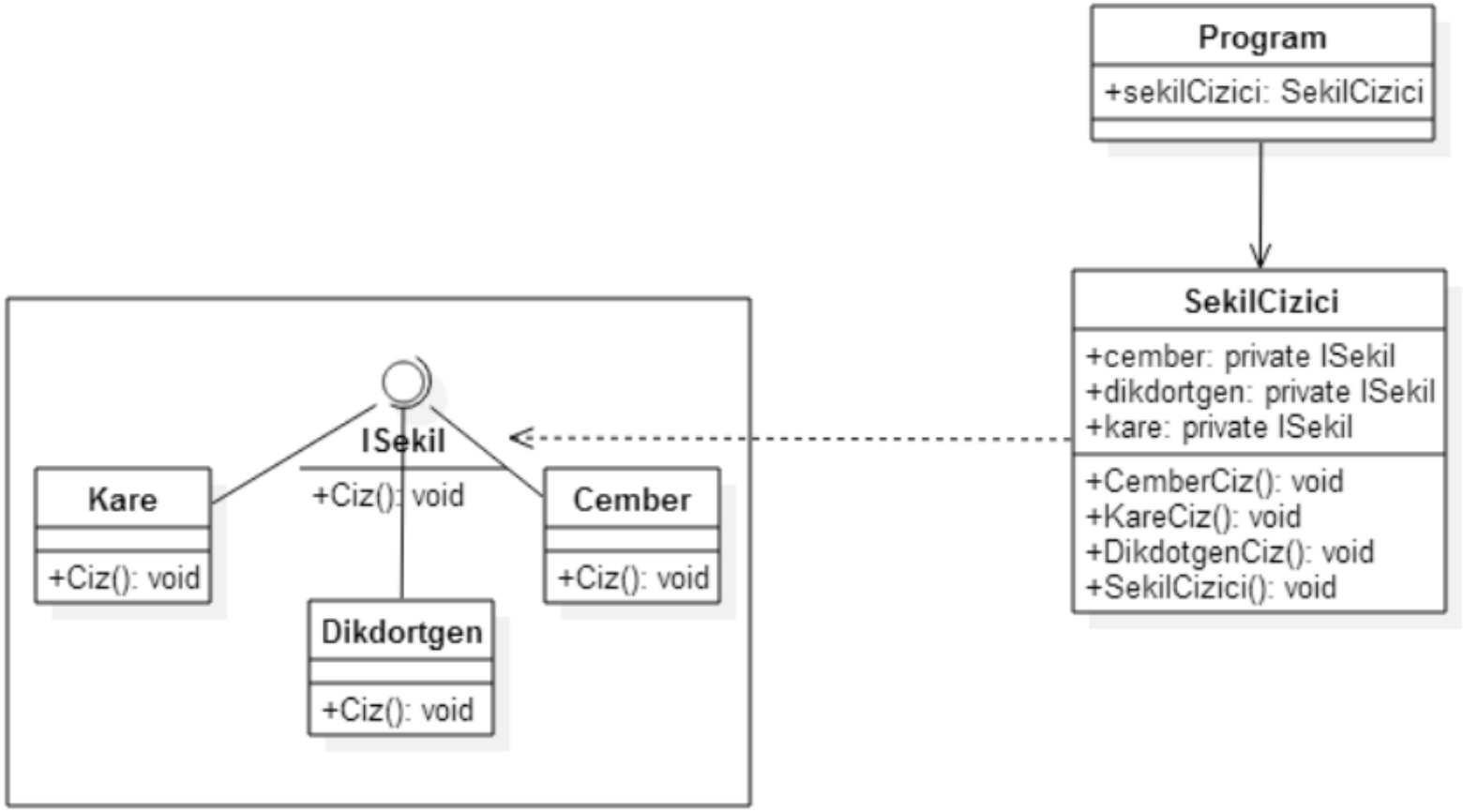


Facade

- Örnek:

void Ciz() fonksiyonuna sahip ISekil arayüzünü implemente eden Kare, Dikdortgen ve Cember sınıflarının (sınıfına bağlı kalmadan) Ciz() metodunu kullanımını sağlayan SekilCizici sınıfını yaratınız.

Facade


Program
+sekilCizici: SekilCizici
SekilCizici
+cember: private ISekil
+dikdortgen: private ISekil
+kare: private ISekil
+CemberCiz(): void
+KareCiz(): void
+DikdotgenCiz(): void
+SekilCizici(): void
ISekil
+Ciz(): void
Kare
+Ciz(): void
Cember
+Ciz(): void
Dikdortgen
+Ciz(): void

Facade

ISekil.java

```
public interface ISekil {
    void Ciz();
}
```

Kare.java

```
public class Kare implements ISekil{

    @Override
    public void Ciz() {
        System.out.println("Kare::Ciz() metodu calisti!");
    }

}
```

Facade

- Main.java

```
public class Program {

    public static void main(String[] args) {
        SekilCizici sekilCizici = new SekilCizici();

        sekilCizici.CemberCiz();
        sekilCizici.DikdortgenCiz();
        sekilCizici.KareCiz();
    }

}
```

SekilCizici.java

```
public class SekilCizici {
    private ISekil cember;
    private ISekil dikdortgen;
    private ISekil kare;

    public SekilCizici() {
        cember = new Cember();
        dikdortgen = new Dikdortgen();
        kare = new Kare();
    }

    public void CemberCiz()
    {
        cember.Ciz();
    }
    public void DikdortgenCiz()
    {
        dikdortgen.Ciz();
    }
    public void KareCiz()
    {
        kare.Ciz();
    }
}
```

Kaynaklar

- Java ve Java Teknolojileri, *Tevfik KIZILÖREN* – Kodlab Yayınları
- Dr Öğr. Üyesi Zehra Aysun ALTIKARDEŞ Nesne Yönelimli Programlama 2 Ders notları
- Yazılım Mühendisliği CBU-Dr. Öğr. Üyesi Deniz Kılınç Yazılım Mimarisi ve Tasarımı Ders Notları
- https://medium.com/gokhanyavas/structural-patterns-yap%C4%B1sal-desenler-7c84f174b7ae
- https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/yap%C4%B1sal-kal%C4%B1plar-(structural-patterns)#:~:text=Yap%C4%B1sal%20kal%C4%B1plar%20(Structural%20patterns)%2C,Design%20Patterns)%20olu%C5%9Fturan%20modellerden%20biridir.&text=Yap%C4%B1sal%20kal%C4%B1plar%20yaz%C4%B1l%C4%B1m%20projelerinin%20tasar%C4%B1m%20a%C5%9Famalar%C4%B1nda%20veya%20bu%20projelerin%20geni%C5%9Fletilme%20a%C5%9Famalar%C4%B1nda%20kullan%C4%B1labilirler.
- https://medium.com/gokhanyavas/structural-patterns-yap%C4%B1sal-desenler-7c84f174b7ae