Command (Komut) Tasarım Şablonu

Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com

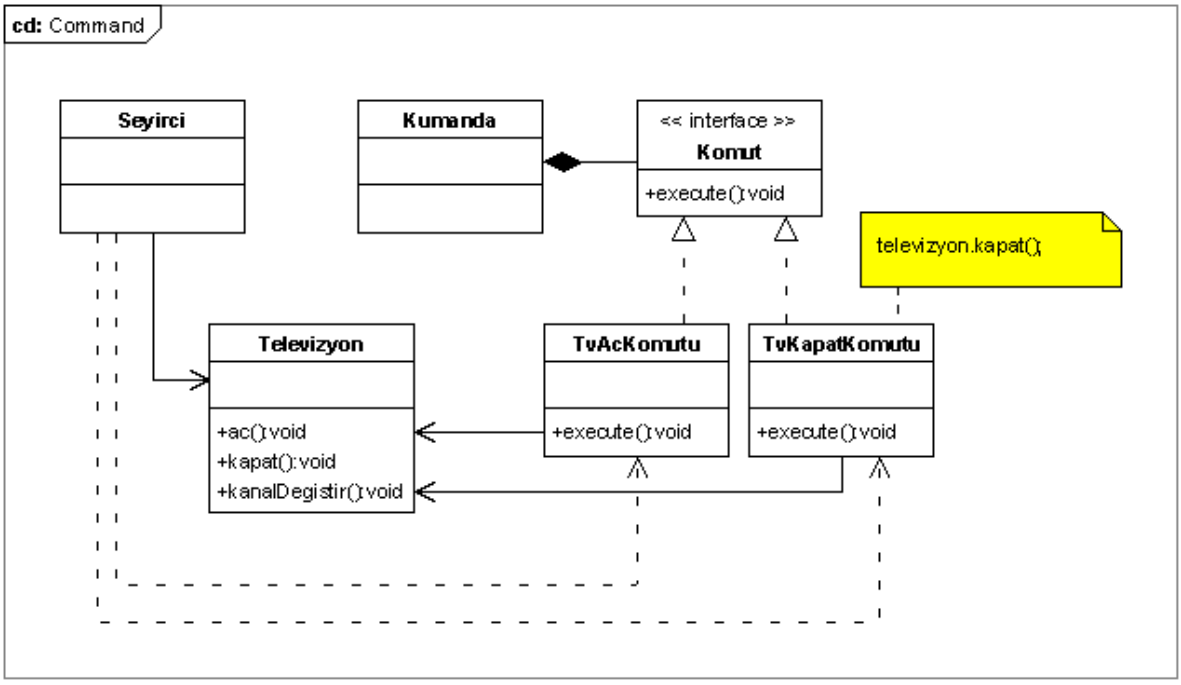
Tavsiye Edilen Diger Makaleler

- Adapter Tasarım Şablonu
- Strategy (Strateji) Tasarım Şablonu
- Iterator (Tekrarlayıcı) Tasarım Şablonu
- Data Access Object (DAO) Tasarım Şablonu
- Singleton (Tekillik) Tasarım Şablonu
- Tasarım Prensipleri
- Test Edilebilir Tasarım
- BizimAlem.com - Bir Sistemin Tasarlanış Hikayesi
- Tasarım Prensipleri
- Çevik Java EE 5 Web Uygulamaları ve Mimarileri

Command (Komut) Tasarım Şablonu

Command tasarım şablonunu açıklamak için televizyonu uzaktan kumanda etme aletini metafer (mecazi örneklendirme) olarak kullanmak istiyorum. Kanalları değiştirmek için kumanda aleti üzerinde belirli bir tuşa basarız. Tusa basıldığı anda kumanda aleti televizyona bir komut göndererek, kanalın değişmesini sağlar. Aynı şekilde televizyonu açıp, kapatmak ve ses ve renk ayarlarını yapmak için değişik tuşlar kullanılır. Kullanıcı olarak tuşa basıldığında, televizyon bünyesinde ne gibi bir işlemin yapıldığı hakkında bilgi sahibi olmamız gerekmiyor. Bunu alıcının (televizyon) bilmesi yeterlidir. Bilmemiz gereken tek şey, hangi tuşun altında hangi komutun olduğudur.

Bir nesne üzerinde bir işleminin nasıl yapıldığını bilmediğimiz ya da kullanılmak istenen nesneyi tanımadığımız durumlarda, Command tasarım şablonu ile yapılmak istenen işlemi bir nesneye dönüştürerek, alıcı nesne tarafından işlemin yerine getirilmesi sağlıyabiliriz.



UML diagramında görüldüğü gibi televizyonu açıp, kapatma işlemleri için *TvAcKomutu* ve *TvKapatKomut* isimlerinde iki sınıf tanımlıyoruz. Bu komutlar *Komut* interface sınıfını implemente ederek, birer *Komut* nesnesi haline geliyorlar. *Komut* interface sınıfında *execute()* isminde bir metod tanımlıyoruz.

```
package org.javatasarim.pattern.command;

/**
 * Komut interface sinifi.
 *
 * @author Oezcan Acar
 *
 */
public interface Komut
{
      /**
```

```
        * Alt siniflar execute() metodunu
        * implemente ederler.
        */
       void execute();
}
```

Gerçek bir komut sınıfı, *Komut* interface sınıfını implemente ettiği için *execute()* metoduna sahip olmak zorundadır. Bu metot bünyesinde yapılmak istenen işlem (örneğin televizyonu aç) alıcı nesneye (televizyon) yönlendirilir (delegation).

```
package org.javatasarim.pattern.command;

/**
 * Televizyonu acmak için kullanilan
 * komut.
 *
 * @author Oezcan Acar
 *
 */
public class TvAcKomutu implements Komut
{

       private Televizyon tv = null;

       public TvAcKomutu(Televizyon tv)
       {
              this.tv = tv;
       }

       public void execute()
       {
              this.tv.ac();
       }
}
```

TvAcKomutu sınıfı bünyesinde görüldüğü gibi *Televizyon* isminde bir sınıf değişkeni tanımlayarak, *execute()* metodu bünyesinde bu nesnenin *ac()* metodunu kullanıyoruz. Analog olarak *TvKapatKomutu* sınıfı sahip olduğu *execute()* metodu bünyesinde *tv.kapat()* metodunu kullanır.

```
package org.javatasarim.pattern.command;

/**
 * Televizyonu kapatmak için kullanilan
 * komut.
 *
 * @author Oezcan Acar
 *
 */
public class TvKapatKomutu implements Komut
{

       private Televizyon tv = null;

       public TvKapatKomutu(Televizyon tv)
       {
```

```
            this.tv = tv;
      }
      public void execute()
      {
            this.tv.kapat();
      }
}
```

Televizyonu açıp kapatma işlemlerini yerine getirebilmek için bir televizyon kumanda aletine ihtiyacımız var. İlk iki tuşu yukarda yer alan komutlar ile programlayarak, televizyon nesnesinin metodlarını kullanacağız.

```
package org.javatasarim.pattern.command;

/**
 * Bir tv kumanda aleti
 *
 * @author Oezcan Acar
 *
 */
public class Kumanda
{
      public Komut[] tus = new Komut[2];

      public Kumanda()
      {
            Televizyon tv = new Televizyon();
            tus[0] = new TvAcKomutu(tv);
            tus[1] = new TvKapatKomutu(tv);
      }

      public void tusla(int i)
      {
            if(i > tus.length || i < 0)
            {
                  throw new RuntimeException("" +
                              "Tus gecersiz!");
            }
            tus[i].execute();
      }
}
```

Kumanda sınıfında, *Komut* interface sınıfından olan nesneleri barındırmak üzere bir *Array* (dizi) tanımlıyoruz. Bu array, kumanda aletinin sahip olduğu tuşları represente etmektedir. Yukarıda yer alan örnekte sahip olduğumuz komundanın iki tuşu mevcuttur. Sınıf konstruktörü bünyesinde bir televizyon nesnesi oluşturuyoruz. Bu nesneyi *TvAcKomutu* ve *TvKapatKomutu* sınıflarından birer nesne oluşturmak için konstruktör parametresi olarak kullanıyoruz. Örnekte görüldüğü gibi komut nesneleri, yapılmak istenen operasyonu bünyesinde barındıran nesneleri tanırlar (*TvAcKomutu* Televizyon nesnesini tanıyor) . Aksi taktirde *TvKapatKomutu* ya da *TvAcKomutu* ne yapmalari gerektiğini bilemezler ve bizim istediğimiz işlemi yerine getiremezler.

Kumanda aletinin sıfırıncı ve birinci tuşlarına basmak için aşağıda yer alan Test sınıfını kullanıyoruz.

```
package org.javatasarim.pattern.command;

/**
 * Test programi
 *
 * @author Oezcan Acar
 *
 */
public class Test
{
	public static void main(String[] args)
	{
		// Kumanda aletini olusturur
		Kumanda kumanda = new Kumanda();

		// sifir nolu tusa basar
		kumanda.tusla(0); // televizyonu acar

		// bir nolu tusa basar
		kumanda.tusla(1); // televizyonu kapatir
	}
}
```

New operatörü ile yeni bir Kumanda nesnesi oluşturduktan sonra, *tusla(0)* ve *tusla(1)* metodları ile sıfırıncı ve birinci tuşlara basıyoruz. Ekran çıktısı aşağıdaki şekilde olacaktır:

```
Televizyon acildi.
Televizyon kapandi
```

Test sınıfı, Televizyon sınıfını ve sahip olduğu *ac()* ve *kapat()* metodlarını tanımadan, televizyonu uzaktan kontrol edebilmektedir. Bu *Command* tasarım şablonu sayesinde gerçekleşmiştir.

Command tasarım şablonu ne zaman kullanılır?

- Yapmak istediğimiz işlemi ya da işlemin ait olduğu nesneyi tanımadığımız durumlarda *Command* tasarım şablonu kullanılır. Bir komut haline getirilen işlem, alıcı nesneye iletilerek işlem gerçekleştirilir. Böyle komutu gönderen nesne ile işlemi gerçekleştirien nesne arasında esnek bir bağ oluşturulur. İki nesne birbirini tanımak zorunda değildir.

İlişkili tasarım şablonları

- *Command* tasarım şablonu *Memento* nesneleri kullanarak, yapılan işlemlerin tekrar geri alınmasında sağlıyabilir.

Bu makale Özcan Acar tarafından yazılmış olan Java Tasarım Şablonları ve Yazılım Mimarileri isimli kitaptan alıntıdır.

Detaylı bilgiyi http://www.pusula.com adresinden edinebilirsiniz.