Decorator Tasarım Şablonu

KurumsalJava.com

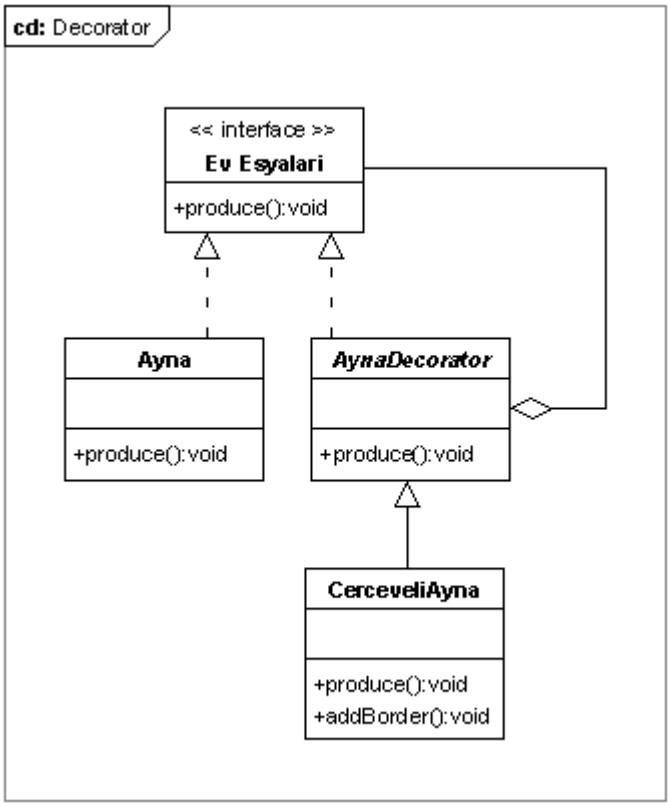
Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com

http://www.javadergisi.com

Mevcut bir sınıf hiyerarşisini ya da sınıfın yapısını değiştirmeden, oluşturulan nesnelere yeni özelliklerin eklenme işlemini gerçekleştirmek için Decorator tasarım şablonu kullanılır.

Alt sınıfların oluşturulması yöntemiyle, sınıflara yeni özelliklerin eklenmesi, daha sonra sisteme eklenecek alt sınıflar için değiştirilmesi zor kalıpların oluşmasını beraberinde getirir. Bu durumda, üst sınıflarda tanımlanmış olan bazı özellikler statik ve alt sınıflar için değistirilemez ya da kullanımı engellenemez bir hal alabilir. Kullanıcı sınıflar içinde bu sorun teşkil edebilir, çünkü kendi istekleri doğrultusunda bir nesnenin ne zaman ve nasıl oluşturulması gerektiğini yönlendiremeyebilirler.

Nesnelere, sahip oldukları sınıfların yapılarının değistirilmeden yeni özelliklerin eklenmesini sağlayan Decorator tasarım şablonu ile, istenilen özelliklerin ekleneceği nesne başka bir nesne içine gömülür. Yeni özellik eklenen nesneyi içine alan nesneye dekoratör ismi verilir. Dekoratör nesnesi ile yeni özellik eklenen nesne aynı üst sınfa dahil olduklarından, birbirleriyle değiştirilebilir haldedirler. Bu özellikten dolayı kullanıcı sınıf, dekoratör sınıf ile dekoratör nesne bünyesinde bulunan diğer nesne arasında ayrım yapmaz. Nesneler arası ilişkiye aşagıda ye ralan Uml diagramında görüyoruz.



Ev eşyaları üzerine uzmanlasmış bir firma, bir fabrikadan hammadde halinde ayna satın alıp, bunları çesitli ürünler haline getirdikten sonra sonra pazarlamaktadır. Programcı olarak bize verilen görev, mevcut Ayna sınıfını kullanarak bir üretim modeli oluşturmaktır. Ayna sınıfı üzerinde bir değişiklik yapılması istenmemektedir. Daha ziyade, değişik aynalı ürünleri modellemek için yeni sınıfların oluşturulması istenmektedir. Pazarlama firması ilk etapta değişik çerçevelere sahip aynaları piyasaya sürmeyi planladığından, hazırlıyacağımız program bu ihtiyaca cevap vermelidir.

Pazarlama firmasının ihtiyaçlarını analiz ettikten sonra, Decorator tasarım şablonu ile istenilen modelin oluşturulabileceği kanaatına varıyoruz. Bir ayna nesnesini kullanarak,

çerçeveli bir ayna nesnesi oluşturabiliriz. Burada yapılması gereken tek işlem, mevcut bir ayna nesnesine çerçeve eklemektir yani bir ayna nesnesini değişik özellikler kullanarak (örneğin çerçeve sahibi olması) dekore etmektir. EvEsylari interface sınıfı ve Ayna sınıfı aşağıdaki yapıya sahiptirler ve bizim tarafımızdan değiştirilmeden program içinde kullanılmaktadırlar.

```
package org.javatasarim.pattern.decorator;

/**
 * Bir ev esyasini temsil
 * eden interface sinif.
 *
 * @author Oezcan Acar
 *
 */
public interface EvEsyalari
{
	/**
	 * Üretimi gerceklestirmek için
	 * kullanilan metod.
	 */
	public void produce();
}
```

```
package org.javatasarim.pattern.decorator;

/**
 * Ayna sınıfi.
 *
 * @author Oezcan Acar
 *
 */
public class Ayna implements EvEsyalari
{

	public void produce()
	{
		System.out.println("Ayna imal edildi.");
	}
}
```

Çerveveli aynalar üretebilmek için, EvEsyalari interface sınıfını implemente eden AynaDecorator sınıfını aşağıdaki şekilde programlıyoruz:

```
package org.javatasarim.pattern.decorator;

/**
 * Degisik tipte aynali ürünleri
 * temsil eden üstsinif.
 *
 * @author Oezcan Acar
 *
 */
public abstract class AynaDecorator implements EvEsyalari
```

```
{
	/*
	 * Bünyesinde mevcut bir ayna nesnesi
	 * bulundurur ve degisik metodlar kullanarak
	 * bu ayna nesnesini dekore eder.
	 */
	private EvEsyalari ayna = new Ayna();

	public EvEsyalari getAyna()
	{
		return ayna;
	}

	public void setAyna(EvEsyalari ayna)
	{
		this.ayna = ayna;
	}
}
```

AynaDecorator sınıfı nesne agregasyon yöntemiyle bünyesinde bir Ayna nesnesi tutmaktadır. Dekorasyonu, yani aynaya bir çerçeve ekleme işlemini gerçeklestirmebilmek için Ayna nesnesi ile böyle bir bağ oluşturulması gerekmektedir. Ayna'nın üretilme işlemi Ayna.produce() metoduna delege edildikten sonra, dekorasyon için gerekli metod, yanı aynaya çerçeve takma işlemi AynaDecorator ya da bir altsınıfı tarafından kullanılacaktır.

AynaDecorator sınıfını soyut olarak tanımlıyoruz. Amacımız değişik türdeki aynaları üretebilmek için esnek bir model oluşturabilmektir. Bugün çerçeveli aynalar üretimi için kullanılan programımız, yarın yeni bir altsınıf oluşturularak başka türde bir aynanın üretimine izin verebilecek yapıda olmalıdır. Bu yüzden AynaDecorator isminde bir soyut sınıf tanımlıyarak, gelecekte üretimi yapılacak tüm aynalı ürünler için bir baz oluşturmuş oluyoruz. Çerçeveli aynaların üretimi için CerceveliAyna sınıfını oluşturuyoruz.

```
package org.javatasarim.pattern.decorator;

/**
 * Cerceveli ayna üretimi
 * yapmak için kullanilan
 * sinif.
 *
 * @author Oezcan Acar
 *
 */
public class CerceveliAyna extends AynaDecorator
{

	/**
	 * Üretim için kullanilan sinif.
	 * addBorder metodu ile
	 * aynaya cerceve ekler.
	 */
	public void produce()
	{
		getAyna().produce();
		addBorder();
	}

	/**
```

```
         * Cerceve ekleme islemini gerceklestirmek
         * için kullanilan metod.
         *
         */
        public void addBorder()
        {
                System.out.println("Aynaya cerceve eklendi.");
        }
}
```

CerceveliAyna sınıfı, ayna üretim işlemini üst sınıfı AynaDecorator üzerinden gerçekleştiriyor. AynaDecorator sınıfında ayna isminde bir sınıf değişkeni olduğu için, CerceveliAyna sınıfi, getAyna() metodu ile, üretilen aynaya ulaşabilir. Üretimi yapılmış bir aynaya çerçeve takmak için addBorder() metodu tanımlanmıştır.

CerveveliAyna.produce() metodu içinde önce ayna üretiliyor, daha sonra addBorder() metodu ile bu aynaya bir çerçeve takılıyor yani ayna dekore edilmiş oluyor.

Sonuç itibariyle aynanın ve aynalı ürünlerin üretimini birbirinden ayırmış olduk ve mevcut bir nesneye (Ayna), sahip olduğu sınıfın yapısını değiştirmeden yeni bir özellik (çerçeve) ekleyebildik. Aşağıda yer alan Test sınıfi ile, çerçeveli ayna üretimine başlıyabiliriz:

```
package org.javatasarim.pattern.decorator;

/**
 * Test programi.
 *
 * @author Oezcan Acar
 *
 */
public class Test
{

        public static void main(String[] args)
        {
                EvEsyalari ayna = new CerceveliAyna();
                ayna.produce();
        }
}
```

Ekran çıktısı aşagıdakı şekilde olacaktır:

```
Ayna imal edildi.
Aynaya cerceve eklendi.
```

Decorator tasarım şablonu ile dekore edilmiş nesneler zincirleme tarzı diğer dekorasyon işlemleri içinde kullanılabilir. Örneğin çercevesi takılmış bir ayna, AynaliDolapDecorator sınıfi bünyesinde aynalı bir dolabın oluşturulmasında tekrar kullanilabilir. Bu durumda fabrikadan çıkan ayna önce çerçeve takılarak çerçeveli bir ayna nesnesi ve daha sonra aynalı bir dolabın üretilmesi için kullanılabilir. Ayna nesnesi iki sefer dekore edilerek aynalı bir dolabın üretimi gercekleşmiş olacaktır.

Decorator tasarım şablonu ne zaman kullanılır?

- Mevcut nesnelere, sınıf yapıları değiştirilmeden yeni özelliklerin eklemesi gerektiği durumlarda;
- Daha sonra kullanımdan kaldırmak üzere yeni metod ve fonksiyonların eklemesi gerektiği durumlarda;
- Sınıf yapısının ya da sınıf hiyerarşilerinin, yeni özelliklerin eklenmesine izin vermedikleri durumlarda Decorator tasarım şablonu kullanılmalıdır.

İlişkili tasarım şablonları:

1. Adapter: Decorator ve Adapter arasındaki tek fark, Decorator tasarım şablonunun nesnenin sunduğu fonksiyonları değiştirmesidir. Decorator, nesnenin dış dünyaya sunduğu metod yapılarını değiştirmez. Adapter tasarım şablonu ile nesnenin dış dünyaya sunduğu metod yapıları tamamen değiştirilir.
2. Strategy: Decorator ile bir nesnenin dış görüntüsü değiştirilir. Strategy tasarım şablonu nesnenin iç dünyasını değiştirir. Kullanılan strateji tipine göre nesne başka türlü bir davranış sergileyecektir.

EOF (End Of Fun)
Özcan Acar