Adapter Tasarım Şablonu

KurumsalJava.com
KurumsalJavaAkademisi.com

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

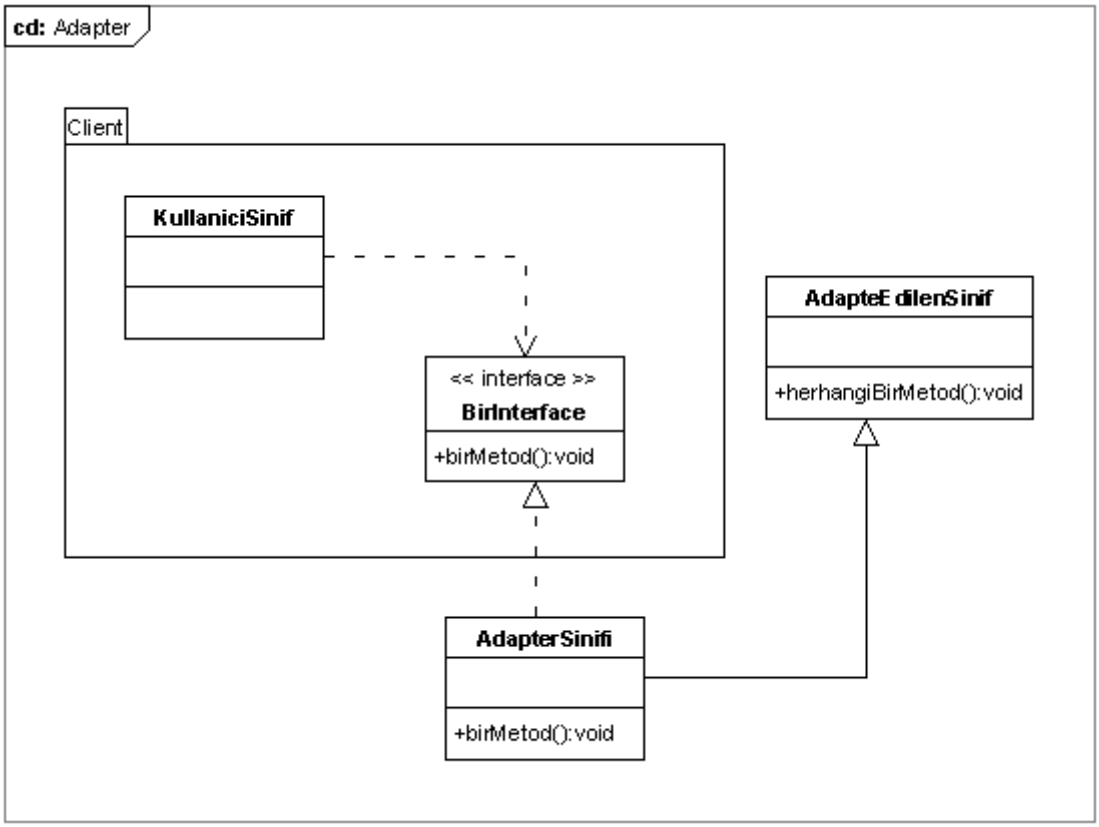
Giriş

Adapter tasarım şablonu yardımı ile, sistemde mevcut bulunan bir sınıfın sunduğu interface (sınıf metodları) başka bir sınıf tarafından kullanilabilir şekilde değiştirilir (adapte edilir). Bu adapter yardımı ile birbiriyle beraber çalışamıyacak durumda olan sınıflar, birlikte çalışabilir hale getirilir.

Adapter tasarım şablonu kendi içinde sınıf ve nesne adapteri olarak ikiye ayrılır. Önce sınıf adapterini inceliyelim.

Sınıf Adapter Tasarım Şablonu

Kurumsal projelerde, sistemin parçaları değişik ekipler tarafından tasarlanır ve implemente edilir. Bu her zaman geçerli olmasada, çoğu zaman daha önceden hazırlanmış sistem komponentlerini kullanmak zorunda kalabiliriz. Aşağıda yer alan UML diagramında da görüldüğü gibi **KullaniciSinif** isminde bir program hazırlanmıştır ve bu program ***BirInterface*** sınıfı içinde yer alan motodları kullanmaktadır. Bu sınıfın sisteme entegrasyonunu sağlıyabilmek için ya ***BirInterface*** sınıfını implemente eden yeni bir sınıf oluşturmamız , ya da mevcut bir sınıfı kullanarak entegrasyonu saglamamız gerekiyor. Ufak bir araştırma yaptıktan sonra, **KullaniciSinif** ismindeki programın işlevini yerine getirebilmesi için, başka bir programcı ekip tarafından hazırlanmış olan **AdapteEdilenSinif** isminde bir sınıf olduğunu ögreniyoruz.



Adapter tasarım şablonunu kullanarak, **KullaniciSinif** sınıfını **AdapteEdilenSinif** metodlarını kullanabilecek hale getirebiliriz. **KullaniciSini**f aşağıdakı yapıya sahiptir:

```java
package org.javatasarim.pattern.adapter;

import org.javatasarim.pattern.adapter.classadapter.AdapterSinifi;

/**
 * BirInterface sinifina bagimli
 * olan KullaniciSinif.
 *
 * @author Oezcan Acar
 *
 */
public class KullaniciSinif
{
	public static void main(String[] args)
	{
		/*
		 * AdapterSinifi, BirInterface interface
		 * sinifini implemente ettigi için
		 * BirInterface olarak sistem içinde
		 * kullanilabilir.
		 */
		BirInterface birInterface = new AdapterSinifi();
		birInterface.birMetod();
	}
}
```

KullaniciSinif ve ***BirInterface*** arasında sıkı bir bağ vardır. **KullaniciSinif** işlevini yerine getirebilmek için mutlaka bir ***BirInterface*** nesnesine ihtiyaç duymaktadır.

```java
package org.javatasarim.pattern.adapter;

/**
 * BirInterface
 *
 * @author Oezcan Acar
 *
 */
public interface BirInterface
{
	public void birMetod();
}
```

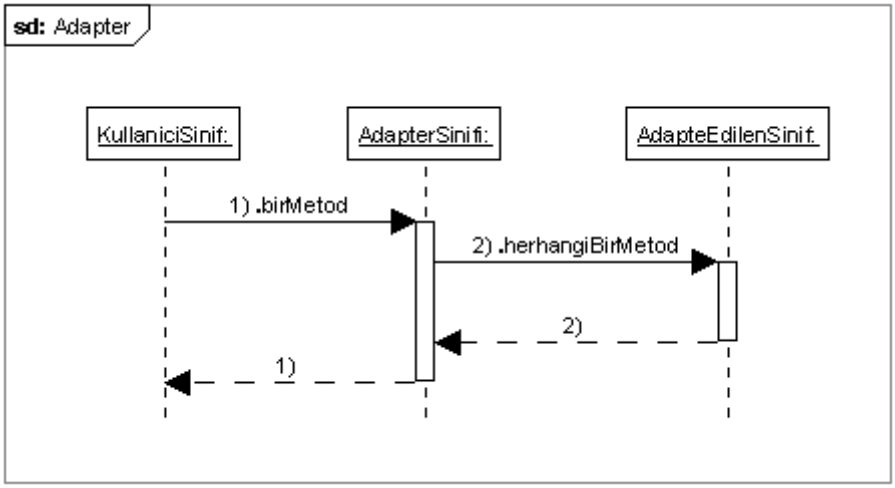
BirInterface sınıfında, ***KullaniciSinif*** sınıfı tarafından ihtiyaç duyulan birMetod() metodu yer almaktadır. Daha öncede belirttiğim gibi, ya ***BirInterface*** sınıfını implemente eden bir altsınıf oluşturacağız, ya da adapter şablonu ile sistemdeki mevcut bir sınıfı adapte etmeye çalışacağız. Yeni bir sınıf oluşturmak yerine, adaptasyon metodunu seçiyoruz. Bunun için **AdapterSinifi** isminde, adaptasyonu sağlamak üzere yeni bir sınıf oluşturuyoruz:

```java
package org.javatasarim.pattern.adapter.classadapter;

import org.javatasarim.pattern.adapter.BirInterface;

/**
 * Adaptasyonu gerceklestirmek için
 * kullanilan sinif. BirInterface sinifini
 * implemente ederek, KullaniciSinifi için
```

```
 * kullanilabilir bir sinif haline gelir.
 * Extends komutu ile AdapteEdilenSinif sinifinda
 * yeralan özellikleri miras olarak alir.
 *
 * @author Oezcan Acar
 *
 */
public class AdapterSinifi extends AdapteEdilenSinif implements
BirInterface
{
	/**
	 * BirInterface interface sinifinda tanimlanmis
	 * olan birMetod() metodu, AdapterSinifi tarafindan
	 * implemente edilir.
	 */
	public void birMetod()
	{
		/**
		 * AdapterEdilenSinif içinde yeralan
		 * herhangiBirMetod() metodunu miras olarak
		 * alir. KullaniciSinifi tarafindan birMetod()
		 * metodu kullanildiginda, bu metod herhangiBirMetod()
		 * metoduna delege edilir ve islem AdapteEdilenSinif
		 * sinifina devredilmis (adapte) edilmis olur.
		 */
		herhangiBirMetod();
	}
}
```

AdapterSinifi ***BirInterface*** interface sınıfını implemente ediyor. Bu andan itibaren **KullaniciSinif** sınıfı **AdapterSinifi** sınıfından oluşturulan bir nesneyi kullanabilir. Ancak bu yeterli degil! **AdapteEdilenSinifi** sınıfının **KullaniciSinif** tarafından dolaylı olarak kullanilabimesi için **AdapterSinifi** ile **AdapteEdilenSinif** arasinda extends komutu ile baglantı kurmamız gerekiyor.

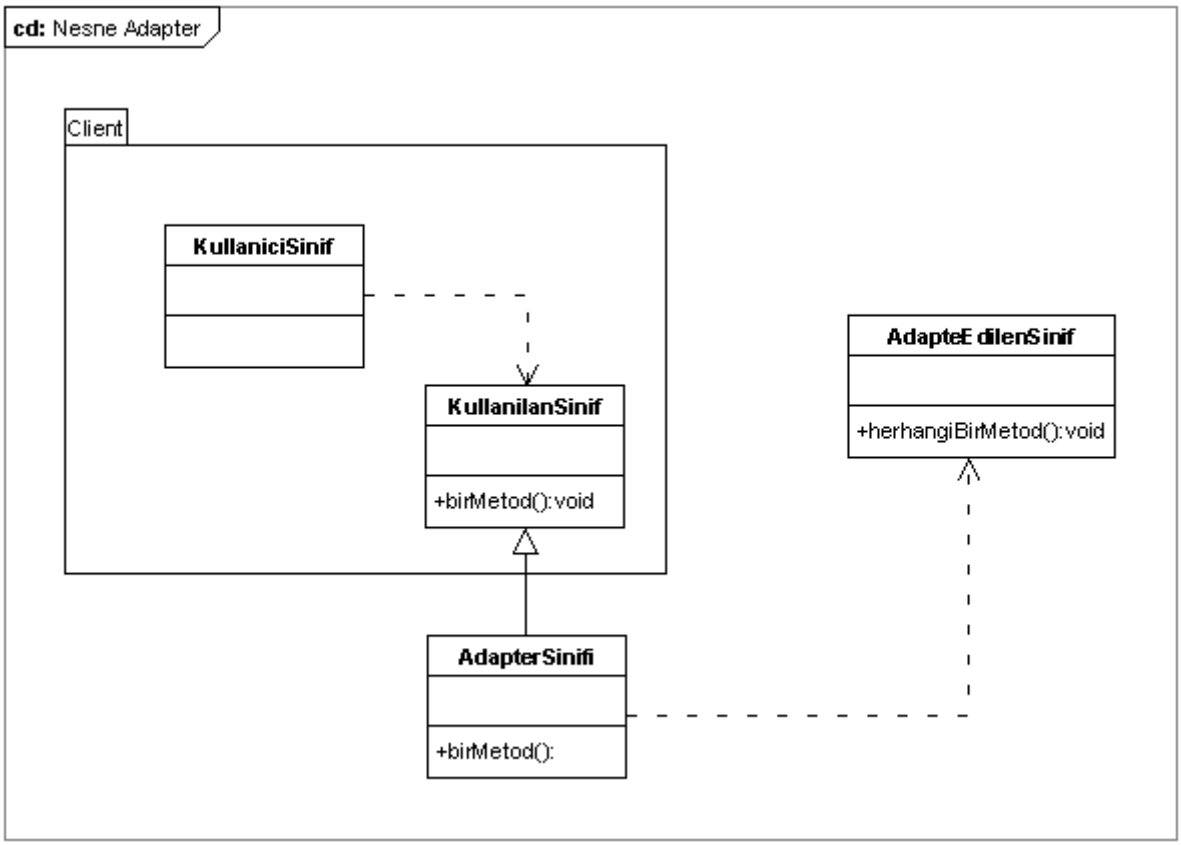
```
package org.javatasarim.pattern.adapter.classadapter;

/**
 * Adapte edilip, sistem için kullanilan
 * bir sinif.
 *
 * @author Oezcan Acar
 *
 */
public class AdapteEdilenSinif
{
	public void herhangiBirMetod()
	{
		System.out.println("Herhangi bir metod");
	}
}
```



Sequence diagramında görüldüğü gibi AdapterSinifi.birMetod() kullanıldığı zaman, işlem otomatik olarak AdapteEdilenSinif.herhangiBirMetod() metoduna delege edilecektir ve bu şekilde adaptasyon gerçekleşmektedir.

Nesne Adapter Tasarım Şablonu

Bazı durumlarda sınıf adapter tasarım şablonu kullanılamayabilir. Bu gibi durumlarda nesne adapter tasarım şablonu alternatif oluşturur.



Adaptasyonu gerçekleştirmek için **KullaniciSinifi** sınıfının kullandığı **KullanilanSinif** **AdapterSinifi** ile genişletilir (extend). Bu durumda **AdapterSinifi** ile **KullanilanSinif** aynı tipten olduklarından, iki sınıfın nesneleride **KullaniciSinif** tarafından kullanilabilir. İkinci adımda **KullaniciSinif** tarafından kullanılan, **KullanilanSinif** sınıfının metodu birMetod() **AdapterSinifi** bünyesinde tekrar implemente edilir. **AdapteEdilenSinif** ile **AdapterSinifi** arasındaki ilişkiyi oluşturmak için, **AdapteEdilenSinifi** sınıfından bir nesne sınıf degiskeni olarak **AdapterSinif** sınıfına eklenir (AdapteEdilenSinif değişkeni). Kullanılan sınıfların kodları aşağıda yer almaktadır.

```
package org.javatasarim.pattern.adapter.objectadapter;

/**
 * KullaniciSinif
 *
 * @author Oezcan Acar
 *
 */
public class KullaniciSinif
{
	public static void main(String[] args)
	{
		KullanilanSinif kullanilanSinif =
					(KullanilanSinif)new AdapterSinifi();
		kullanilanSinif.birMetod();
	}
}
```

KullaniciSinif.main() metodunda görüldüğü gibi, new operatörü ile **KullanilanSinif** sınıfından bir nesne oluşturmak yerine, **AdapterSinifi** sınıfından bir nesne oluşturuyoruz. Nesneyi cast [1] yaparak, bir **KullanilanSinif** nesnesine dönüştürüyoruz, çünkü **KullaniciSinif** sadece **KullanilanSinif** nesnelerini kullanabilir. Cast işleminden sonra bu nesne üzerinde birMetod() metodunu kullanabiliriz.

```
package org.javatasarim.pattern.adapter.objectadapter;

/**
 * KullanilanSinif
 *
 * @author Oezcan Acar
 *
 */
public class KullanilanSinif
{
	public void birMetod()
	{
		System.out.println("bir metod");
	}
}
```

[1] Bir altsınıfdan oluşturulan nesneyi, üstsınıf nesnesine dönüştürmek icin cast işlemi yapılır. Her altsınıf nesnesinde, üstsınıfın da nesnesi bulunduğu için, altsınıf nesnesini üstsınıf nesnesine dönüştürebiliriz. Cast işlemi yapıldıktan sonra sadece üstsınıf değişken ve metodlarına ulaşılabililr. Bu işleme upcast adı verilir. Downcast olarak bilinen işlem Java da mümkün değildir. Bir üstsınıftan oluşturulan nesne, bir altsınıf nesnesine dönüştürülemez.

```java
package org.javatasarim.pattern.adapter.objectadapter;


/**
 * Adaptasyonu gerceklestirmek için
 * kullanilan sinif. KullanilanSinif
 * sinifini genisleterek, birMetod()
 * metodunu yeniden implemente eder.
 *
 * @author Oezcan Acar
 *
 */
public class AdapterSinifi extends KullanilanSinif
{
	/*
	 * Adaptasyonu gerceklestirmek için
	 * AdapteEdilen sınıfından bir nesne
	 * olusturulur.
	 */
	private AdapteEdilenSinif AdapteEdilenSinif =
		new AdapteEdilenSinif();

	/**
	 * Bu metoda gelen cagrilar, sinif degiskeni
	 * olan AdapteEdilenSinif degiskenine
	 * gönderilir.
	 */
	public void birMetod()
	{
		getAdapteEdilenSinif().herhangiBirMetod();
	}

	public AdapteEdilenSinif getAdapteEdilenSinif()
	{
		return AdapteEdilenSinif;
	}

	public void setAdapteEdilenSinif(AdapteEdilenSinif
			AdapteEdilenSinif)
	{
		this.AdapteEdilenSinif = AdapteEdilenSinif;
	}
}
```

```java
package org.javatasarim.pattern.adapter.objectadapter;

/**
 * Adapte edilip, sistem için kullanilan
 * bir sinif.
 *
 * @author Oezcan Acar
 *
 */
public class AdapteEdilenSinif
{
	public void herhangiBirMetod()
	{
		System.out.println("Herhangi bir metod");
```

```
        }
}
```

AdapterSinifi sınıfında tanımladığımız **AdapteEdilenSinif** değişkeni, **AdapteEdilenSinif** sınıfına bir nevi köprü oluşturmaktadır. birMetod() kendisine gelen çağrıları bu nesne üzerinden **AdapteEdilenSinif** sınıfına iletir ve bu şekilde **KullaniciSinif** ile **AdapteEdilenSinif** arasındaki adaptasyon gerçekleşmis olur.

Adapter tasarım şablonu ne zaman kullanılır?

- Sistemde mevcut sınıflar kullanmak istenildiginde, lakin sınıfın sunmuş olduğu arayüz (metod ve değiskenler) istenilen cinsten degilse ;
- Tekrar kullanılabilir ve sistemin diğer bölümlerinden bağımsız bir sınıf ya da komponent oluşturulmak istendiginde.

İlişkili tasarım şablonları:

- Bridge tasarım şablonu yapı itibariyle Nesne Adaptar tasarım şablonuna benzer, ama iki tasarım şablonunun kullanım amaçları değişiktir. Bridge tasarım şablonu ile soyut sınıflar ve implementasyonlar birbirinden ayrılır. Adapter tasarım şablonu bir sınıfın sunmuş olduğu arayüz (metodlar) değiştirilir.
- Proxy tasarım şablonu ile başka bir nesne için temsilci oluşturulur. Proxy temsil ettiği nesnenin arayüzünü değiştirmez.

EOF (End Of Fun)

Özcan Acar
Bilgisayar Mühendisi
KurumsalJava.com
KurumsalJavaAkademisi.com