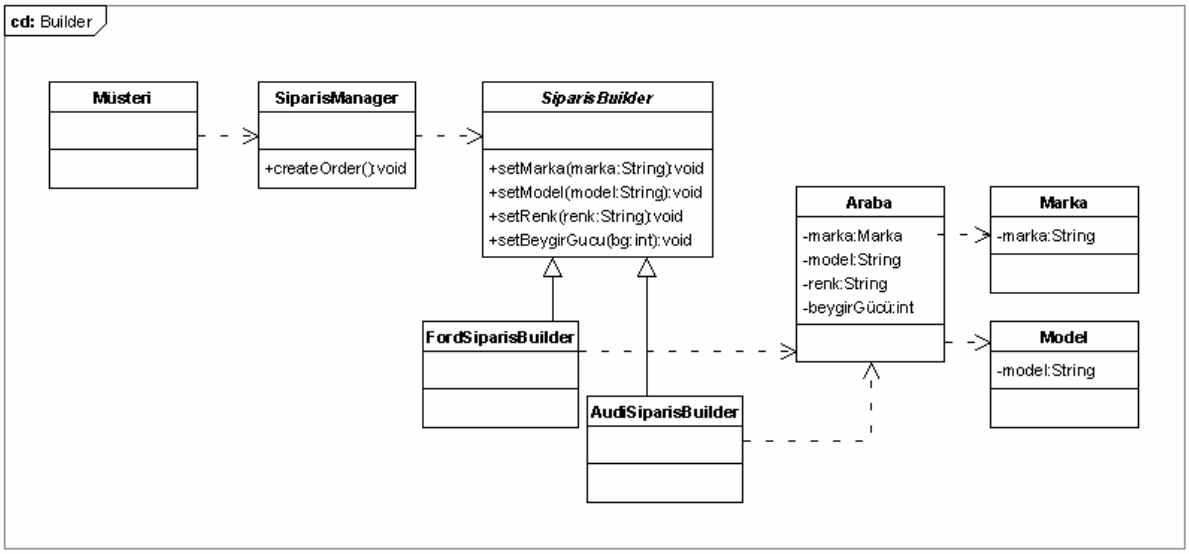
Builder Tasarım Şablonu

KurumsalJava.com

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

Daha önceki bölümlerde Abstract Factory tasarım şablonu ile değişik nesne ailelerinden nasıl nesneler üretildiğini incelemiştik. Builder tasarım şablonu da Abstract Factory tasarım şablonunda oldugu gibi istenilen bir tipte nesne oluşturmak için kullanılır. İki tasarım şablonu arasındaki fark, Builder tasarım şablonunun kompleks yapıdaki bir nesneyi değişik parçaları bir araya getirerek oluşturmasında yatmaktadır. Birden fazla adım içeren nesne üretim sürecinde, değişik parçalar birleştirilir ve istenilen tipte nesne oluşturulur.

Diğer bölümlerde olduğu gibi bir örnek üzerinde bu tasarım şablonunu yakından inceliyelim.



Araba sınıfından oluşturulacak bir nesne, bünyesinde Marka ve Model sınıflarından birer nesne ihtiva edecektir, yani bir araba nesnesi bereberinde bir marka ve bir model nesnesi getirir. Bu örnekte Builder tasarım şablonunu kullanma amacımız, Araba, Marka ve Model sınıflarından oluşan bir nesnenin değişik parametreler kullanarak konfigüre edilmesidir. Sipariş edilen her arabanın markası, modeli, rengi ve beygirgücü değişik olabileceğinden, sipariş için kullanılan Araba nesnesinin bu parametrelere dayalı olarak oluşturulması gerekmektedir. Builder tasarım şablonu bu işlemi, sistemin diğer bölümlerinden bağımsız bir şekilde gerçekleştirmek için kullanılır.

Araba sınıfı aşağıdakı yapıya sahiptir. Her sipariş verilen arabanın markası, modeli, rengi ve hangi beygir gücüne sahip olduğu bu sınıftan oluşturulan nesne içinde tutulur.

```
package org.javatasarim.pattern.builder;

/**
 * Araba sinifi
 *
 * @author Oezcan Acar
 *
 */
public class Araba
{
      /*
       * Markayi temsil etmek icin
       * Marka sinifi kullaniliyor.
       */
      private Marka marka = null;
```

```
	/*
	 * Model temsil etmek icin
	 * Model sinifi kullaniliyor.
	 */
	private Model model = null;

	private String renk = null;
	private int beygirGucu = 0;


	public Araba()
	{
	}

	public Model getModel()
	{
		return model;
	}

	public void setModel(Model model)
	{
		this.model = model;
	}

	public String getRenk()
	{
		return renk;
	}

	public void setRenk(String renk)
	{
		this.renk = renk;
	}

	public int getBeygirGucu()
	{
		return beygirGucu;
	}

	public void setBeygirGucu(int beygirGucu)
	{
		this.beygirGucu = beygirGucu;
	}

	public Marka getMarka()
	{
		return marka;
	}

	public void setMarka(Marka marka)
	{
		this.marka = marka;
	}


}
```

```
package org.javatasarim.pattern.builder;

/**
```

```
 * Arabanin markasi.
 *
 * @author Oezcan Acar
 *
 */
public class Marka
{
	private String marka;

	public Marka(String marka)
	{
		setMarka(marka);
	}

	public String getMarka()
	{
		return marka;
	}

	public void setMarka(String marka)
	{
		this.marka = marka;
	}

	public String toString()
	{
		return marka;
	}
}
```

```
package org.javatasarim.pattern.builder;

/**
 * Arabanin modeli.
 *
 * @author Oezcan Acar
 *
 */
public class Model
{
	private String model;

	public Model(String model)
	{
		setModel(model);
	}

	public String getModel()
	{
		return model;
	}

	public void setModel(String model)
	{
		this.model = model;
	}

	public String toString()
	{
		return model;
	}
```

```
}
```

Değişik marka ve modellerde arabaların sipariş edilebilmesi için soyut olan *SiparisBuilder* sınıfını tanımlıyoruz.

```
package org.javatasarim.pattern.builder;

/**
 * Siparis sürecinde konfigürasyon
 * icin kullanilacak metodlar bu
 * sinif icinde tanimlanir. Belirli
 * bir marka arabanin siparis icin
 * altsiniflarin olusturulmsi gerekmektedir.
 *
 * @author Oezcan Acar
 *
 */
public abstract class SiparisBuilder
{
	private Araba araba = null;

	public SiparisBuilder()
	{
	}

	public Araba getAraba()
	{
		if(araba == null) araba = new Araba();
		return araba;
	}

	public abstract void setMarka(String marka);
	public abstract void setModel(String model);
	public abstract void setRenk(String renk);
	public abstract void setBeygirGucu(int bg);

}
```

Ford ve Audi marka arabaların siparişinde kullanılmak üzere *SiparisBuilder* sınıfının alt sınıflarını tanımlıyoruz.

```
package org.javatasarim.pattern.builder;

/**
 * Audi marka bir arabanin
 * konfigürasyon ve siparisinde
 * kullanilir.
 *
 * @author Oezcan Acar
 *
 */
public class AudiSiparisBuilder extends SiparisBuilder
{

	public AudiSiparisBuilder()
	{
	}

	public void setBeygirGucu(int bg)
	{
		getAraba().setBeygirGucu(bg);
```

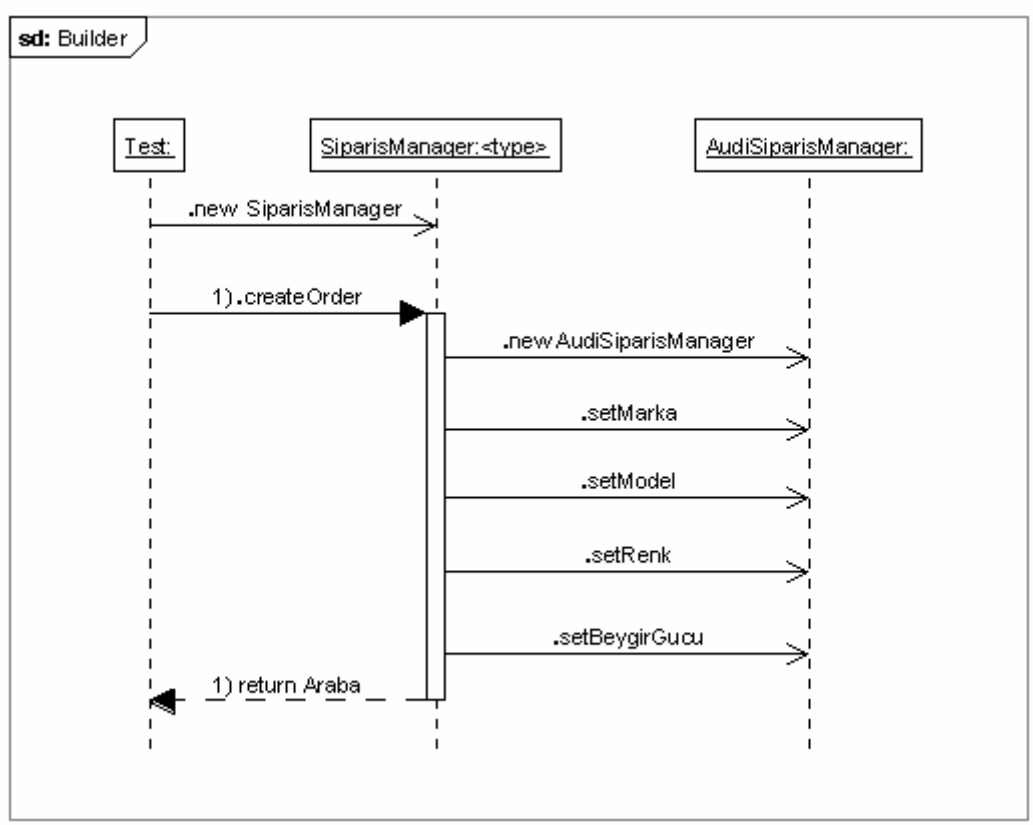
```
	}

	public void setMarka(String marka)
	{
		getAaraba().setMarka(new Marka(marka));
	}

	public void setModel(String model)
	{
		getAaraba().setModel(new Model(model));
	}

	public void setRenk(String renk)
	{
		getAaraba().setRenk(renk);
	}
}
```

```
package org.javatasarim.pattern.builder;

/**
 * Ford marka bir arabanin
 * konfigürasyon ve siparisinde
 * kullanilir.
 *
 * @author Oezcan Acar
 *
 */
public class FordSiparisBuilder extends SiparisBuilder
{

	public FordSiparisBuilder()
	{
	}

	public void setBeygirGucu(int bg)
	{
		getAraba().setBeygirGucu(bg);
	}

	public void setMarka(String marka)
	{
		getAraba().setMarka(new Marka(marka));
	}

	public void setModel(String model)
	{
		getAraba().setModel(new Model(model));
	}

	public void setRenk(String renk)
	{
		getAaraba().setRenk(renk);
	}
}
```

Geriye sipariş işlemini gerçekleştirmek için kullanılan *SiparisManager* sınıfı kalıyor:

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

/**
 * Siparis islemini gerceklestirmek
 * icin SiparisManager sinifi
 * kullanilir.
 *
 * @author Oezcan Acar
 *
 */
public class SiparisManager
{
	private SiparisBuilder builder;

	/**
	 * Müsterinin istedigi marka, model, renk ve
	 * beygirgücüne sahip bir arabanin siparisi
	 * icin kullanilan metod.
	 *
	 * @param marka Siparis edilen arabanin markasi
	 * @param model Siparis edilen arabanin modeli
	 * @param renk Siparis edilen arabanin rengi
	 * @param beygirgucu Siparis edilen arabanin beygir gücü
	 * @return Araba Konfigüre edilmis araba nesnesi
	 */
	public Araba createOrder(String marka, String model,
			String renk, int beygirgucu)
	{
		if(marka.equals("Ford"))
		{
			builder = new FordSiparisBuilder();
		}
		else if(marka.equals("Audi"))
		{
			builder = new AudiSiparisBuilder();
		}

		this.builder.setMarka(marka);
		this.builder.setModel(model);
		this.builder.setRenk(renk);
		this.builder.setBeygirGucu(beygirgucu);
		return this.builder.getAraba();
	}


	/**
	 * Siparis edilen arabanin özelliklerini ekranda görüntüler.
	 */
	public void printOrder() {
		System.out.println("Marka: " +
				this.builder.getAraba().getMarka());
		System.out.println("Model: " +
				this.builder.getAraba().getModel());
		System.out.println("Renk: " +
				this.builder.getAraba().getRenk());
		System.out.println("Beygirgücü: "
				 + this.builder.getAraba().getBeygirGucu());
	}
}
```

createOrder() metodu ile istenilen tipte bir arabanın siparişi gerçekleştiriliyor. Bu metod parametre olarak arabanın marka, model, rengi ve beygir gücünü alıyor. Hangi marka arabanın sipariş edildiğini marka isimli değişkene bakarak bulabiliyoruz. Eğer marka Ford ise FordSiparisBuilder, Audi ise AudiSiparisBuilder sınıflarından bir nesne oluşturuluyor. Akabinde *setMarka(), setModel(), setRenk()* ve *setBeygirGucu()* metodları kullanılarak, Araba sınıfından oluşturulan nesnenin konfigürasyonu gerçeklestiriliyor.



Sequence diagramında da görüldügü gibi, Araba nesnesini oluşturmak için dört adım atılması gerekiyor, bunlar *SiparisBuilder* sınıfının soyut olarak tanımlanmış olan set metodlarıdır. Daha öncede belirttiğim gibi kompleks yapıya sahip olan Araba nesnesini değişik parçaları (Marka, Model, renk, beygirgücü) bir araya getirerek oluşturduk. Nesnenin nasıl oluşturulup, parçalarının biraraya getirileceğini *SiparisManager* sınıfının *createOrder()* metodunda tanımlıyoruz. Builder tasarım şablonunu kullanabilmek için, *createOrder()* metodunda olduğu gibi, kompleks nesnenin oluşumunda kullanılan metodların tanımlanması ve belirli bir sıraya göre kullanılmasi gerekmektedir.

Builder tasarım şablonu ne zaman kullanılır?

- Değişik parametreler kullanılarak aynı tip kompleks yapıda bir nesnenin olusturulmasında;
- Kompleks yapıya sahip nesnenin oluşturulma sürecinin, sistemin diğer bölümlerinden bağımsız bir şekilde yapılması gerektiği durumlarda.

İlişkili tasarım sablonları:

- Composite Tasarım şablonunda oluşturulan composite nesne, Builder tasarım şablonunda oluşturulan kompleks nesne gibidir.

- Abstract Factory kullanılarak kompleks nesneler oluşturulabilir.

EOF (End Of Fun)
Özcan Acar