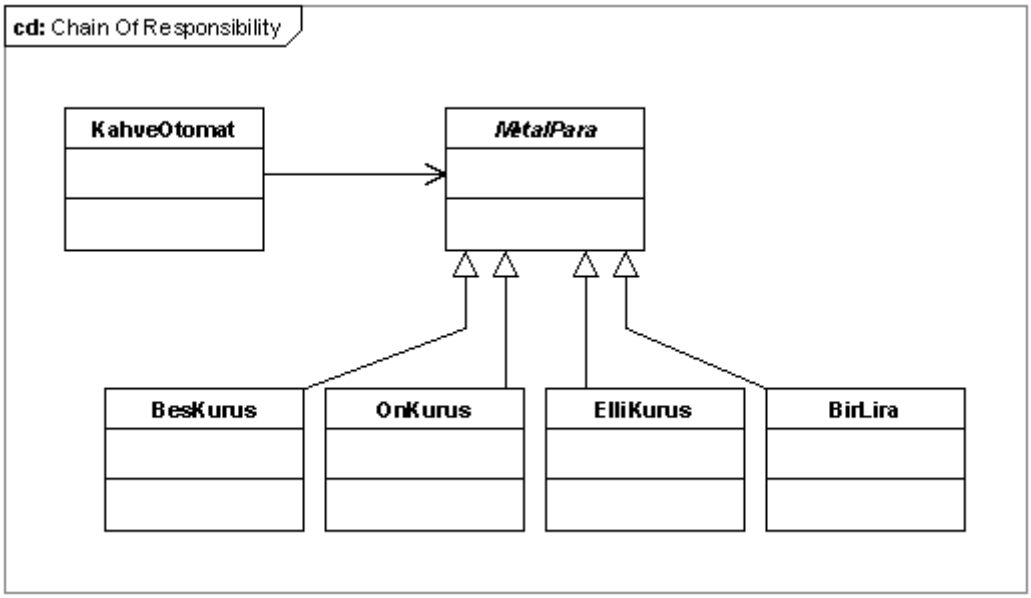
Chain of Responsibility Tasarım Şablonu

KurumsalJava.com

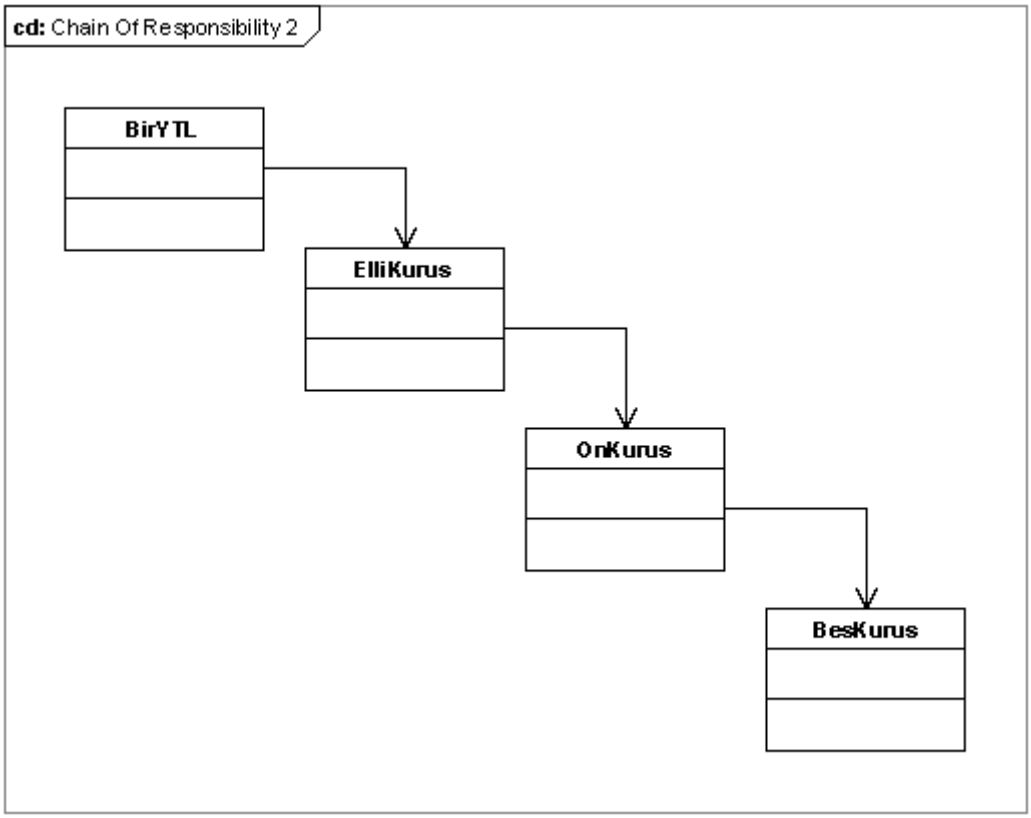
Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com

Chain of responsibility sorumluluk zinciri anlamına gelmektedir. Sisteme gönderilen bir istediğin (komut) hangi nesne tarafından cevaplanması gerektiğini bilmediğimiz durumlarda ya da isteği yapan nesne ve servis sağlayan nesne arasında sıkı bir bağ oluşmasını engellememiz gerektiğinde Chain of Responsibility tasarım şablonu kullanılır. Bu tasarım şablonunda servis sağlayan ilgili tüm nesneler bir kolye üzerindeki boncuklar gibi birbirleriyle ilişkili hale getirilir. Bir nesne zincirdeki kendinden sonraki nesneyi tanır ve isteği kendi cevaplayamadığı durumda, kendinden sonraki nesneye iletir. Bu işlem, zincirde bulunan doğru servis saglayıcı nesneyi bulana kadar devam eder.

Bu tasarım şablonu için ilginç bir örnek kullanmak istiyorum. İçine para atarak, kahve aldığımız bir kahve otomatı düşünelim. Bir kahvenin bedeli 1 TL olabilir. Kahveyi alabilmek için 1 TL değerindeki metal parayı otomata atmamız gerekiyor.



Otomatın, içine atılan metal paraları tanıyabilmesi için metal paraları temsil eden nesnelerden bir zincir oluşturmamız gerekiyor. Her metal para bu zinciri, doğru para nesnesini bulana kadar baştan sona kadar dolaşacaktır. Örneğin metal para nesneler zincirini su şekilde oluşturabiliriz:



Otomata atılan metal para, zincirin ilk nesnesi olan **BirLira** tarafından karşılanır ve metal paranın bir Lira olup olmadıgı kontrol edilir. Eğer metal para bir Lira ise, otomat parayı kabul eder. Eğer atılan metal para bir Lira değilse, işlem zincirin ikinci nesnesi **ElliKurus**'a gönderilir. Bu işlem zincirin en son elemanı olan **BesKurus** nesnesine kadar devam eder. Atılan metal para eğer zincirde bulunan bir nesne ile eşdeğerde ise, o zaman metal para kabul edilir, aksi taktirde reddedilir.

```
package org.javatasarim.pattern.chainofresponsibility;

import java.util.ArrayList;

/**
 * Metal para üstsinifi
 *
 * @author Oezcan Acar
 *
 */
public abstract class MetalPara
{

      /**
       * Otomata atilan paralarin tutuldugu
       * liste
       */
      private ArrayList metalParaListesi =
            new ArrayList();

      /**
       * Metal paranin sahip olduğu değer.
       * 5, 10, 50, 100 Kurus olabilir
       */
```

```
	private int value;

	/**
	 * Zincirde yeralan bir sonraki nesne
	 */
	private MetalPara next;

	public int getValue()
	{
		return value;
	}

	public void setValue(int value)
	{
		this.value = value;
	}

	public MetalPara getNext()
	{
		return next;
	}

	public MetalPara setNext(MetalPara next)
	{
		this.next = next;
		return this;
	}

	public void check(MetalPara para)
	{
		System.out.println("Siradaki nesne sadece "
				+ this +" isleyebilir.");

		if(para.getValue() != this.value)
		{
			System.out.println("Uymadi, zincirdeki bir " +
					"sonraki nesneye iletiyoruz.");

			if(getNext() != null)
			{
				getNext().check(para);
			}
			else
			{
				System.out.println("Zincirin sonundayiz. " +
					"Metal para "+ para.toString() + " " +
					"bu otomat için uygun degil.");
			}
		}
		else
		{
			metalParaListesi.add(para);
			System.out.println("Otomat tarafindan "
					+ this.toString() + " kabul edildi");
		}
	}
}
```

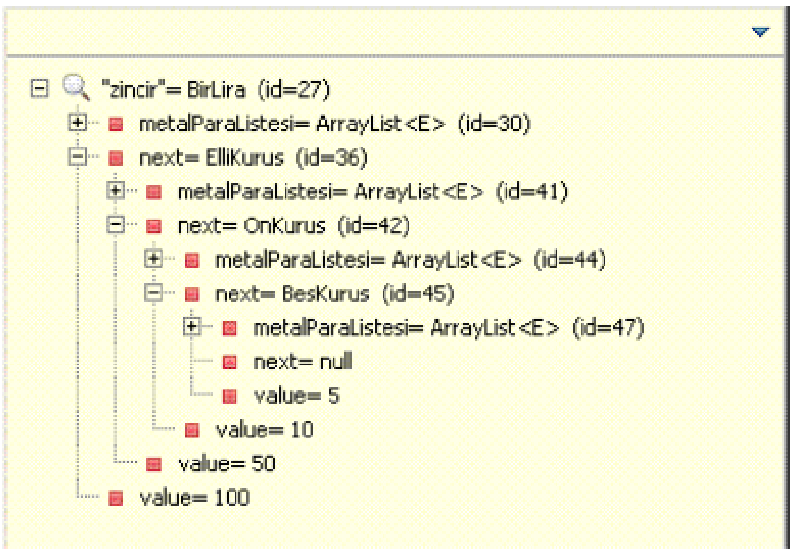
Metal paraları temsil eden **MetalPara** isminde soyut bir sınıf tanımlıyoruz. Metal paraları temsil eden nesnelerden oluşan bir zincir oluşturabilmek için **MetalPara** sınıfı bünyesinde next isminde bir değişken yer almaktadır. Gerçek metal paralar (BesKurus, ElliKurus vs)

MetalPara soyut sınıfını genişlettiği (extends) için, next değişkenini miras olarak alacaklardır. Daha sonra göreceğimiz gibi, next değişkeninde, zincirin bir sonraki elemanı yer alır.

check() ve *setNext()* metodlarının nasıl çalıştığıı anlayabilmek için önce bir zincirin nasıl oluşturulduğuna göz atalım:

```
private MetalPara zincir = null;
zincir = (new BirLira()).setNext((
                        new ElliKurus()).setNext((
                                    new OnKurus()).setNext(
                                                new BesKurus())));
```

Zinciri oluşturan nesneler MetalPara tipinde olacaktır. Bu yüzden MetalPara tipine sahip zincir isminde bir değişken tanımlıyoruz. Zincirin ilk nesnesini new BirLira() şeklinde oluşturduktan sonra *setNext()* ile bir sonraki nesneyi oluşturuyoruz. Zincir için gerekli tüm nesneleri arka arkaya *setNext()* metodu bünyesinde bu şekilde oluşturabiliriz. Programın bu satırlarını Eclipse altında debug ettiğimiz taktirde, şöyle bir nesne zinciri oluşacaktır:



Resimde de görüldüğü gibi zincir değişkeni aslında **BirLira** sınıfından oluşturulmuş bir nesnedir. Bu nesnenin next değişkeninde **ElliKurus** sınıfından bir nesne yer almaktadır. Aynı şekilde **ElliKurus** nesnesinin next değişkeni **OnKurus** ve **OnKurus** nesnesinin next değişkeni **BesKurus** sınıfından bir nesneyi ihtiva eder. Böylece

BirLira --> ElliKurus --> OnKurus --> BesKurus

şeklinde bir zincir oluşturmuş oluyoruz. Zincirdeki bir nesne, kendisinden sonraki nesneye next değişkeni üzerinden bağlıdır. Sadece zincirin en son elamanı olan BesKurus'un sahip olduğu next değişkeni null değerindedir. Null zincirin son buldugunu ifade eder.

MetalPara.*check()* metoduna tekrar geri dönelim. Bu metod bünyesinde otomata atılan para ile, zincirde sırası gelmiş olan nesne karşılaştırılır. Eğer sıradaki nesne, atılan metal para ile uyuşmuyorsa, **getNext()**.*check(para)* ile kontrol zincirdeki bir sonraki nesneye verilir. Bu işlem zincirin sonuna gelene kadar tekrarlanır.

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

/**
 * 100 Kurusluk (1YTL) metal para sinifi
 *
 * @author Oezcan Acar
 *
 */
public class BirLira extends MetalPara
{
	public String toString()
	{
		return "1 Lira";
	}

	public BirLira()
	{
		setValue(100);
	}
}
```

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

/**
 * 50 Kurusluk metal para sinifi
 *
 * @author Oezcan Acar
 *
 */
public class ElliKurus extends MetalPara
{

	public String toString()
	{
		return "50 kurus";
	}

	public ElliKurus()
	{
		setValue(50);
	}
}
```

BirLira ve **ElliKurus** sınıflarını gördük. Diğer sınıflarıda analog olarak aynı şekilde oluşturuyoruz.

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


public class KahveOtomati
{

	private static MetalPara zincir = null;

	public static void main(String[] args)
	{
		zincir = (new BirLira()).setNext((
				new ElliKurus()).setNext((
						new OnKurus()).setNext(
								new BesKurus())));
```

```
            paraAt(new BirKurus());
            paraAt(new ElliKurus());
            paraAt(new OnKurus());
            paraAt(new OnKurus());
            paraAt(new OnKurus());
            paraAt(new OnKurus());
            paraAt(new BesKurus());
            paraAt(new BesKurus());
            paraAt(new BirKurus());

      }

      public static void paraAt(MetalPara para)
      {
            System.out.println("+++ ------------------------ +++");
            System.out.println("Otomata " + para.toString()
                        + "  atildi.");
            zincir.check(para);
            System.out.println("+++ ------------------------ +++\n");
      }

}
```

Bir kahve otomatını simule etmek amacıyla **KahveOtomati** isminde bir test sınıfı oluşturuyoruz. Bu sınıfın bünyesinde, metal paraları kontrol etmek için gerekli nesne zincirini oluştuyoruz. *paraAt()* metodu ile otomata metal paralar atmaya başlıyoruz. Şimdi ekran çıktısını inceliyelim:

```
1.  +++ ------------------------ +++
2. Otomata 1 kurus  atildi.
3. Siradaki nesne sadece 1 Lira isleyebilir.
4. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
5. Siradaki nesne sadece 50 kurus isleyebilir.
6. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
7. Siradaki nesne sadece 10 kurus isleyebilir.
8. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
9. Siradaki nesne sadece 5 kurus isleyebilir.
10. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
11. Zincirin sonundayiz. Metal para 1 kurus bu otomat için uygun
degil.
+++ ------------------------ +++

+++ ------------------------ +++
12. Otomata 50 kurus  atildi.
13. Siradaki nesne sadece 1 Lira isleyebilir.
14. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
15. Siradaki nesne sadece 50 kurus isleyebilir.
16. Otomat tarafindan 50 kurus kabul edildi
+++ ------------------------ +++

+++ ------------------------ +++
17. Otomata 10 kurus  atildi.
18. Siradaki nesne sadece 1 Lira isleyebilir.
19. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
20. Siradaki nesne sadece 50 kurus isleyebilir.
21. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
22. Siradaki nesne sadece 10 kurus isleyebilir.
23. Otomat tarafindan 10 kurus kabul edildi
+++ ------------------------ +++
```

```
+++ ------------------------ +++
24. Otomata 10 kurus  atildi.
25. Siradaki nesne sadece 1 Lira isleyebilir.
26. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
27. Siradaki nesne sadece 50 kurus isleyebilir.
28. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
29. Siradaki nesne sadece 10 kurus isleyebilir.
30. Otomat tarafindan 10 kurus kabul edildi
+++ ------------------------ +++

+++ ------------------------ +++
31. Otomata 10 kurus  atildi.
32. Siradaki nesne sadece 1 Lira isleyebilir.
33. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
34. Siradaki nesne sadece 50 kurus isleyebilir.
35. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
36. Siradaki nesne sadece 10 kurus isleyebilir.
37. Otomat tarafindan 10 kurus kabul edildi
+++ ------------------------ +++

+++ ------------------------ +++
38. Otomata 10 kurus  atildi.
39. Siradaki nesne sadece 1 Lira isleyebilir.
40. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
41. Siradaki nesne sadece 50 kurus isleyebilir.
42. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
43. Siradaki nesne sadece 10 kurus isleyebilir.
44. Otomat tarafindan 10 kurus kabul edildi
+++ ------------------------ +++

+++ ------------------------ +++
45. Otomata 5 kurus  atildi.
46. Siradaki nesne sadece 1 Lira isleyebilir.
47. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
48. Siradaki nesne sadece 50 kurus isleyebilir.
49. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
50. Siradaki nesne sadece 10 kurus isleyebilir.
51. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
52. Siradaki nesne sadece 5 kurus isleyebilir.
53. Otomat tarafindan 5 kurus kabul edildi
+++ ------------------------ +++

+++ ------------------------ +++
54. Otomata 5 kurus  atildi.
55. Siradaki nesne sadece 1 Lira isleyebilir.
56. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
57. Siradaki nesne sadece 50 kurus isleyebilir.
58. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
59. Siradaki nesne sadece 10 kurus isleyebilir.
60. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
61. Siradaki nesne sadece 5 kurus isleyebilir.
62. Otomat tarafindan 5 kurus kabul edildi
+++ ------------------------ +++

+++ ------------------------ +++
63. Otomata 1 kurus  atildi.
64. Siradaki nesne sadece 1 Lira isleyebilir.
65. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
66. Siradaki nesne sadece 50 kurus isleyebilir.
67. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
68. Siradaki nesne sadece 10 kurus isleyebilir.
```

```
69. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
70. Siradaki nesne sadece 5 kurus isleyebilir.
71. Uymadi, zincirdeki bir sonraki nesneye iletiyoruz.
72. Zincirin sonundayiz. Metal para 1 kurus bu otomat için uygun
degil.
+++ ----------------------- +++
```

2. satirda otomata bir kuruşluk metal para atıldığını görmekteyiz. Bu metal para tüm zinciri baştan aşağı dolaştıktan sonra 11. satırda görüldüğü gibi otomat tarafından reddedilmektedir, çünkü otomat sadece 1 YTL, 50 kuruş, 10 kuruş ve 5 kuruşluk metal paraları kabul etmektedir.

12. satırda 50 kuruşluk bir metal para atılmaktadır. 50 kuruşluk metal para zincirin ilk nesnesi **BirLira**'ya 13. satırda verilir. Bu nesne sorumlu olmadığından, kontrolü 14 satırda **ElliKurus** nesnesine verir. 16. satırda görüldüğü gibi 50 kuruşluk metal para, zincirin bu elamanı tarafından kabul edilir. Tüm işlemlerin ardından atılan 1x50, 4x10 ve 2x5 kuruşluk metal paraların kabul edildiğini, lakin 2x1 kuruşun reddedildiğini görmekteyiz.

Chain of Responsibility tasarım şablonu ne zaman kullanılır?

- Sisteme yöneltilen bir isteğin (request) birden fazla nesne tarafından işlenmesi gerektiği durumlarda Chain of Responsibility tasarım şablonu kullanılır.
- Kullanıcı sınıf ile servis sağlayan nesne arasında sıkı bir bağ oluşmasını engellemek için Chain of Responsibility tasarım şablonu kullanılabilir. Chain of Responsibility ile kullanıcı ve servis sunucu nesneler birbirlerini tanımak zorunda değildirler.

İlişkili tasarım sablonları:

- Chain of Responsibility genelde Composite tasarım şablonu ile beraber kullanılır. Bir composite nesneyi oluşturan alt nesneler zincirin parcası olarak düşünülebilir. Chain of Responsibility tasarım şablonu ile bu zincir üzerinde işlem yapılır.

EOF (End Of Fun)
Özcan Acar