KurumsalJava.com
Java Enterprise Architecture

Iterator -Tekrarlayıcı Tasarım Şablonları Serisi

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

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

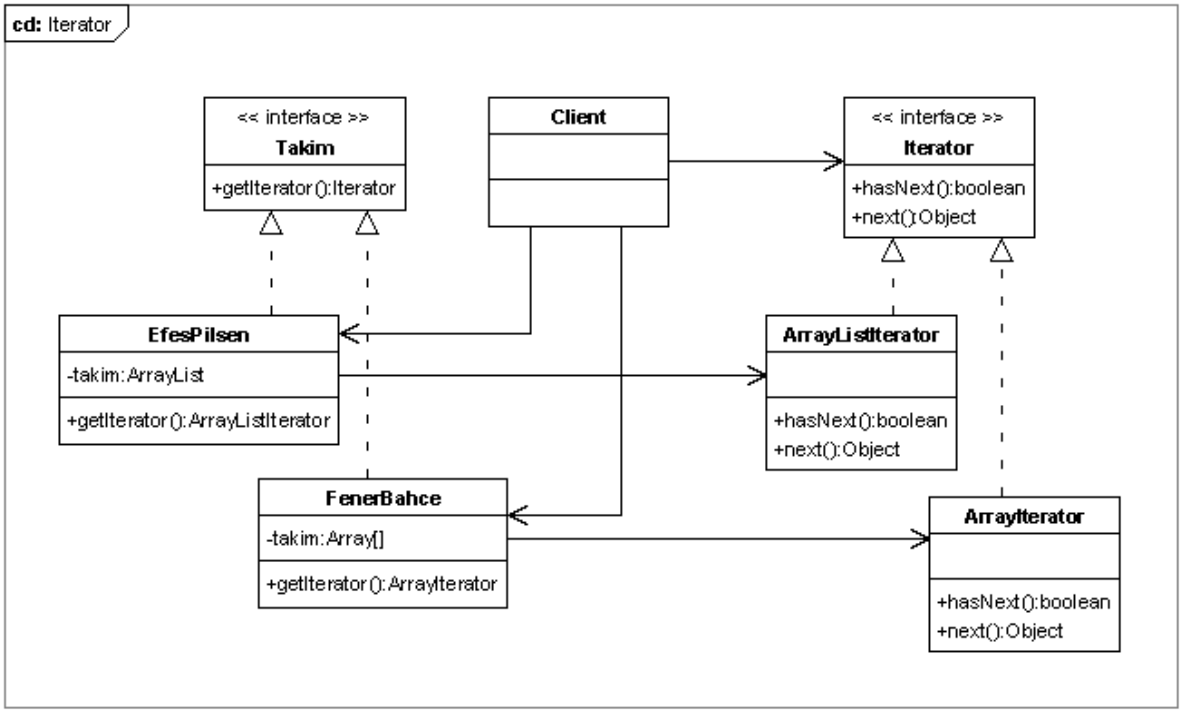
Detaylı bilgiyi http://www.pusula.com adresinden edinebilirsiniz.

Okunması Tavsiye Edilen Diğer Makaleler

- Tasarım Şablonu Nedir?
  http://www.kurumsaljava.com/2008/11/20/tasarim-sablonu-nedir/
- Singleton (Tekillik) Tasarım Şablonu
  http://www.kurumsaljava.com/2008/11/27/singleton-yanlizlik-tasarim-sablonu/
- Data Access Object (DAO) Tasarım Şablonu
  http://www.kurumsaljava.com/2008/12/01/data-access-object-dao-tasarim-sablonu/
- Java'da Interface ve Soyut (Abstract) Sınıf Kullanımı
  http://www.kurumsaljava.com/2008/11/19/javada-interface-ve-soyut-abstract-sinif-kullanimi/

Iterator (Tekrarlayıcı) Tasarım Şablonu

Iterator tasarım şablonu ile bir listede yer alan nesnelere sırayla, listenin yapısını ve çalışma tarzını bilmek zorunluluğumuz olmadan erişebilir ve bu nesneler üzerinde işlem yapabiliriz.



UML diagramında ***Takim*** interface sınıfını implemente eden iki basketbol takımı yer almaktadır: Efes Pilsen ve Fenerbahçe Ülker Basketbol takımları. Ligde oynayan her takım için ***Takim*** interface sınıfını implemente eden bir sınıf oluşturulabilir. Takım içinde yer alan oyuncuları tutmak için her sınıf kendi bünyesinde bir liste oluşturabilir. ***Takim*** interface sınıfı, bu listenin yapısı hakkında bir zorunluluk getirmediği için her takım kendi listesini istediği yapıda oluşturabilir. **EfesPilsen** sınıfını incelediğimiz zaman, oyuncuların bir ArrayList içinde tutulduğunu görmekteyiz.

```
package org.javatasarim.pattern.iterator;

import java.util.ArrayList;

/**
 * Efes Pilsen Basketbol Takimi
 *
 * @author Oezcan Acar
 *
 */
public class EfesPilsen
{

      private ArrayList<Oyuncu> takim;

      public EfesPilsen()
      {
            // Isimler ve numaralar Efes Pilsen
```

```
            // takimi resmi sitesinden alinmistir
            // http://www.efesbasket.org
            takim = new ArrayList();
            takim.add(new Oyuncu("Drew", 4));
            takim.add(new Oyuncu("Ender", 6));
            takim.add(new Oyuncu("Cenk", 7));
            takim.add(new Oyuncu("Kerem", 12));
            takim.add(new Oyuncu("Loren", 15));
      }

      public ArrayList<Oyuncu> getTakim() {
            return takim;
      }

      public void setTakim(ArrayList<Oyuncu> takim) {
            this.takim = takim;
      }
}
```

FenerBahce sınıfı, oyuncularını bir Array[] içinde tutmaktadır:

```
package org.javatasarim.pattern.iterator;


/**
 * Fenerbahce Ülker Basketbol Takimi
 * @author acar
 *
 */
public class FenerBahce implements Takim
{
      private Oyuncu[] takim = new Oyuncu[5];

      public FenerBahce()
      {
            // Isimler ve numaralar Fenerbahce Ülker
            // takimi resmi sitesinden alinmistir
            // http://www.fenerbahce.org/kurumsal/
            // kategori.asp?ContentCategoryID=38

            takim[0] = new Oyuncu("Ismail", 4);
            takim[1] = new Oyuncu("Willie", 5);
            takim[2] = new Oyuncu("Semih", 9);
            takim[3] = new Oyuncu("Ibrahim", 10);
            takim[4] = new Oyuncu("Serhat", 33);
      }
}
```

Görüldüğü üzere iki takım oyuncuları iki değişik tip listede yer aldı. İki takımın sahip olduğu oyuncu isimlerini ekranda görüntülemek için aşağıdaki sınıf kullanılabilir:

```
package org.javatasarim.pattern.iterator;

/**
 * Test sinifi
 *
 * @author Oezcan Acar
 *
```

```
 */
public class Test
{

	public static void main(String[] args)
	{
		EfesPilsen efesPilsen = new EfesPilsen();

		for(int i=0; i < efesPilsen.getTakim().size(); i++)
		{
			System.out.println(efesPilsen.getTakim()
					.get(i).getIsim());
		}

		FenerBahce fenerBahce = new FenerBahce();

		for(int i=0; i< fenerBahce.getTakim().length; i++)
		{
			System.out.println(fenerBahce.getTakim()[i].getIsim());
		}
	}

}
```

İki takımın oyuncularını ekranda görüntüleyebilmek için her takımın listesini edinip, o listenin gereksinimlerini göz önünde bulundurarak işlem yapmamız gerekiyor. Bu sebepten dolayı **Test** sınıfı hem komplike bir yapı alıyor hem de takımların oyuncularını nasıl ve hangi bir tip listede tuttuklarını bilmemiz gerekiyor. Bu bakımı çok zor bir **Test** sınıfının oluşması için yeterli iki sebeptir. Ligde onlarca takımın yer aldığını ve her takımın değişik bir oyuncu listesi olduğunu düşünürsek, Test sınıfının yapısının ne kadar dahada zorlaşacağını hemen görebiliriz.

Iterator tasarım şablonunu kullanarak **Test** sınıfının yapısını çok daha basit bir hale getirebiliriz. Amacımız takımlar tarafından kullanılan liste yapısını gizlemek ve kullanıcı sınıfların (örneğin Test) tanımlanmış interface metotları üzerinden değişik tipteki listeler üzerinde işlem yapmalarını sağlamaktır. Takımlar ne tip bir liste kullanırlarsa kullansınlar, kendilerine uygun bir iterator nesnesine sahip oldukları sürece, kullanıcı sınıflar tarafından oyuncu listeleri, Iterator interface metotları kullanılarak işlenebilir hale gelecektir. Böylece kullanıcı sınıf takımların kullandıkları listelerden bağımsız bir hale gelir.

Iterator tasarım şablonunu uygulayabilmek için önce ***Takim*** isminde bir interface sınıf tanımlıyoruz:

```
package org.javatasarim.pattern.iterator;

/**
 * Takim interface sinifi
 *
 * @author Oezcan Acar
 *
 */
public interface Takim
{
	Iterator getIterator();
}
```

Takim interface sınıfı bünyesinde getIterator() isminde, kullanılan takıma göre uyumlu bir Iterator nesnesi geri veren bir metot tanımlıyoruz.

EfesPilsen ve **FenerBahce** takımları, ***Takim*** interface sınıfını implemente ederek, dış dünyaya, sahip oldukları takım listesi üzerinde işlem yapmayı kolaylaştıracak bir iterator nesnesi sunarlar. Değişik tipte listeler üzerinde işlem yapabilecek şekilde Iterator sınıfları oluşturmadan önce, listeler üzerinde uygulanmak üzere kullanılacak **Iterator** interface ve metotları tanımlıyoruz:

```
package org.javatasarim.pattern.iterator;

/**
 * Iterator interface sinifi
 *
 * @author Oezcan Acar
 *
 */
public interface Iterator
{
      boolean hasNext();
      Object next();
}
```

hasNext() metodu ile bir takımın sahip olduğu listenin bir sonraki pozisyonunda oyuncu olup, olmadığı tespit edilir. Eğer liste sonuna gelinmişse false değeri geri verilerek, listede oyuncu kalmadığı sinyali verilir. next() metodu ile listenin bir sonraki elementi edinilir.

Basketbol takımlarını ***Takim*** inteface sınıfını implemente edecek şekilde değiştiriyoruz:

```
package org.javatasarim.pattern.iterator;

import java.util.ArrayList;

/**
 * Efes Pilsen Basketbol Takimi
 *
 * @author Oezcan Acar
 *
 */
public class EfesPilsen implements Takim
{

      private ArrayList<Oyuncu> takim;

      public EfesPilsen()
      {
            // Isimler ve numaralar Efes Pilsen
            // takimi resmi sitesinden alinmistir
            // http://www.efesbasket.org
            takim = new ArrayList();
            takim.add(new Oyuncu("Drew", 4));
            takim.add(new Oyuncu("Ender", 6));
            takim.add(new Oyuncu("Cenk", 7));
            takim.add(new Oyuncu("Kerem", 12));
            takim.add(new Oyuncu("Loren", 15));
      }
```

```
	public ArrayList<Oyuncu> getTakim() {
		return takim;
	}

	public void setTakim(ArrayList<Oyuncu> takim) {
		this.takim = takim;
	}

	public Iterator getIterator()
	{
		return new ArrayListIterator(getTakim());
	}
}
```

EfesPilsen sınıfı, Takim interface sınıfını implemente ettiği için getIterator() isminde yeni bir metoda sahip oluyor. Bu metot, **EfesPilsen** sınıfının sahip olduğu takım listesi üzerinde işlem yapmayı kolaylaştıracak olan ArrayListIterator sınıfından bir nesne oluşturarak, kullanıcı sınıfa verir. ArrayListIterator sınıfı aşağıdaki yapıya sahiptir:

```
package org.javatasarim.pattern.iterator;

import java.util.ArrayList;

/**
 * ArraListIterator sinifi
 *
 * @author Oezcan Acar
 *
 */
public class ArrayListIterator implements Iterator
{

	private ArrayList<Oyuncu> takim;
	private int pozisyon;

	public int getPozisyon()
	{
		return pozisyon;
	}

	public void setPozisyon(int pozisyon)
	{
		this.pozisyon = pozisyon;
	}

	public ArrayListIterator(ArrayList<Oyuncu> takim)
	{
		setTakim(takim);
	}

	public boolean hasNext()
	{
		if(pozisyon >= getTakim().size() ||
				getTakim().get(pozisyon) == null)
			return false;
		else return true;
	}

	public Object next()
```

```
	{
		Oyuncu oyuncu = getTakim().get(pozisyon);
		pozisyon ++;
		return oyuncu;
	}

	public ArrayList<Oyuncu> getTakim()
	{
		return takim;
	}

	public void setTakim(ArrayList<Oyuncu> takim)
	{
		this.takim = takim;
	}
}
```

Bu sınıf bünyesinde dikkatimizi çeken hasNext() ve next() metotlarının implementasyonudur. **EfesPilsen** sınıfı takım oyuncularını tutmak için bir ArrayList kullandığı için, ArrayListIterator sınıfı, hasNext() ve next() metotlarını ArrayList tipi bir listede işlem yapabilecek şekilde implemente etmiştir.

EfesPilsen sınıfı üzerinde uyguladığımız değişiklikleri **FenerBahc**e sınıfına da uyguluyoruz.

```
package org.javatasarim.pattern.iterator;


/**
 * Fenerbahce Ülker Basketbol Takimi
 * @author acar
 *
 */
public class FenerBahce implements Takim
{
	private Oyuncu[] takim = new Oyuncu[5];

	public FenerBahce()
	{
		// Isimler ve numaralar Fenerbahce Ülker
		// takimi resmi sitesinden alinmistir
		// http://www.fenerbahce.org/kurumsal/
		// kategori.asp?ContentCategoryID=38

		takim[0] = new Oyuncu("Ismail", 4);
		takim[1] = new Oyuncu("Willie", 5);
		takim[2] = new Oyuncu("Semih", 9);
		takim[3] = new Oyuncu("Ibrahim", 10);
		takim[4] = new Oyuncu("Serhat", 33);
	}

	public Oyuncu[] getTakim()
	{
		return takim;
	}

	public void setTakim(Oyuncu[] takim)
	{
		this.takim = takim;
	}
```

```
	public Iterator getIterator()
	{
		return new ArrayIterator(getTakim());
	}
}
```

FenerBahce takımı oyuncuların yer aldığı Array (Oyuncu[]) tipi bir liste kullanmaktadır. getIterator() metodunda, Array üzerinde işlem yapmasını bilen ArrayIterator sınıfından bir nesne oluşturulur. ArrayIterator sınıfı aşağıdaki yapıya sahiptir:

```
package org.javatasarim.pattern.iterator;

/**
 * ArrayIterator sinifi
 *
 * @author Oezcan Acar
 *
 */
public class ArrayIterator implements Iterator
{

	private Oyuncu[] takim;
	private int pozisyon;

	public int getPozisyon()
	{
		return pozisyon;
	}

	public void setPozisyon(int pozisyon)
	{
		this.pozisyon = pozisyon;
	}

	public ArrayIterator(Oyuncu[] takim)
	{
		setTakim(takim);
	}

	public boolean hasNext()
	{
		if(pozisyon >= getTakim().length ||
				getTakim()[pozisyon] == null)
			return false;
		else return true;
	}

	public Object next()
	{
		Oyuncu oyuncu = getTakim()[pozisyon];
		pozisyon ++;
		return oyuncu;
	}

	public Oyuncu[] getTakim()
	{
		return takim;
	}
```

```
	public void setTakim(Oyuncu[] takim)
	{
		this.takim = takim;
	}
}
```

ArrayIterator sınıfı bir Array üzerinde işlem yapmak üzere oluşturulduğu için, hasNext() ve next() metotları bir Array listesi üzerinde işlem yapacak şekilde implemente edilmiştir.

Bu değişikliklerin ardından, takımların sahip oldukları liste tiplerini bilme zorunluluğunu kullanıcı sınıftan (**Test**) alarak, Iterator interface sınıfını implemente eden sınıflara aktarmış olduk. Kullanıcı sınıflar tarafından kullanılmak üzere her liste tipi için geçerli metotları ***Iterator*** interface sınıfında tanımlayarak, listeler üzerinde yapılan işlemlere genel bir çerçeve verilmiş oluyor. Bu sayede kullanıcı sınıflar, takımlar tarafından kullanılan liste tipinden bağımsız bir hale geldi, yani kullanıcı sınıf, bir takımın kullandığı liste tipini bilmek zorunda değil. Bu böyle olunca, yapılan işlemler ve kullanıcı sınıfın yapısı daha kolay bir hal alır. **Test** sınıfının yeni hali aşağıda yer almaktadır:

```
package org.javatasarim.pattern.iterator;

/**
 * Test sinifi
 *
 * @author Oezcan Acar
 *
 */
public class Test
{

	public static void main(String[] args)
	{
		EfesPilsen efesPilsen = new EfesPilsen();
		Iterator it = efesPilsen.getIterator();

		while(it.hasNext())
		{
			Oyuncu oyuncu = (Oyuncu)it.next();
			System.out.println(oyuncu.getIsim());
		}

		FenerBahce fenerBahce = new FenerBahce();
		it = fenerBahce.getIterator();

		while(it.hasNext())
		{
			Oyuncu oyuncu = (Oyuncu)it.next();
			System.out.println(oyuncu.getIsim());
		}
	}
}
```

Test sınıfı her takımdan bir iterator nesnesi edinerek, hasNext() ve next() metotları ile o takımın sahip olduğu liste üzerinde işlem yapabilir, bunun için listenin hangi tipte olması gerektiğini bilmek zorunda değildir. Böylece Test sınıfı bakımı daha kolay bir hale gelmektedir.

Iterator tasarım şablonu ne zaman kullanılır?

- Sınıflar, bünyelerinde başka nesneleri barındırmak için değişik tipte listelere sahip olabilirler. Bu sınıfların nasıl implemente edildiği gizlemek ve sahip oldukları listeler üzerinde işlem yapmayı kolaylaştırmak için Iterator tasarım şablonu kullanılır.

İlişkili tasarım şablonları:

- Factory tasarım şablonu kullanılarak uyumlu bir Iterator alt sınıfı oluşturulabilir.
- Memento tasarım şablonu çoğu zaman Iterator tasarım şablonu ile beraber kullanılır. Iterator bir memento nesnesini kullanarak, sahip olduğu değeri saklayabilir (hafızada tutmak). Bunun için iterator memento nesnesini sınıf değişkeni olarak tanımlar.
- Iteratörler kompozit (composite) nesnelerde olduğu gibi çoğu zaman rekursif (tekrarlanan) yapılarda kullanılır.