Singleton Tasarım Şablonu
Tasarım Şablonları Serisi

KurumsalJava.com

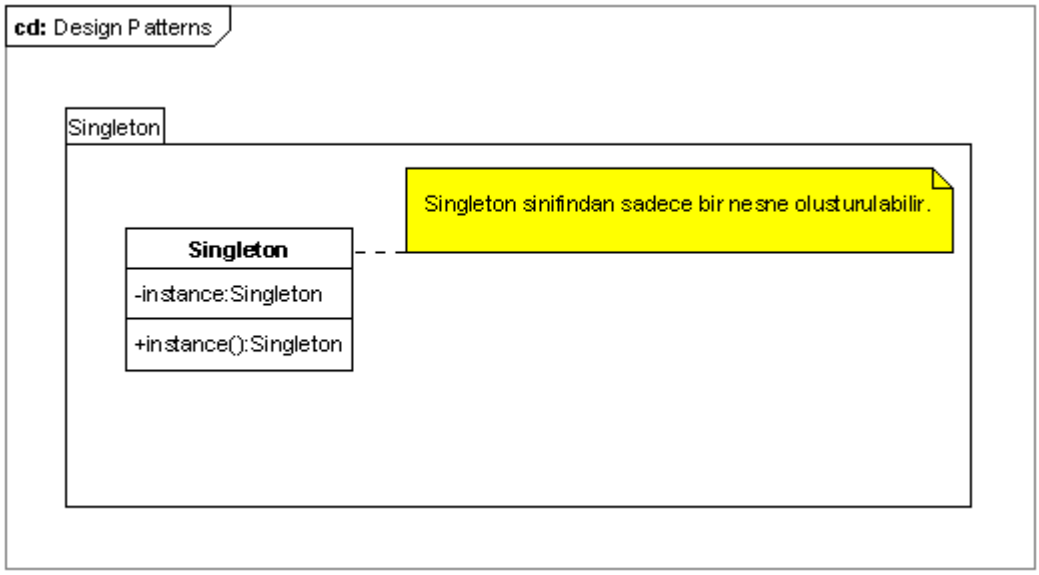
Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com

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

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

Singleton (Tekillik) Tasarım Şablonu

Bazı şartlar altında bir sınıftan sadece bir nesnenin oluşturulması ve oluşturulan bu nesnenin tüm sistemde kullanılması gerekebilir. Örneğin bilgibankası için bir connection pool (bilgibankası bağlantı havuzu) sadece bir defa oluşturulmalı ve kullanılmalıdır. Bu durumda Singleton tasarım şablonu kullanılarak, bir sınıftan sadece bir nesnenin oluşturulması sağlanabilir.



Singleton yapıya sahip bir sınıfı inceliyelim:

```
package org.javatasarim.pattern.singleton;

/**
 * Singleton tasarim sablonu örnegi
 *
 * @author Oezcan Acar
 *
 */
public class Singleton
{
	/*
	 * Singleton sinifindan olusturulabilecek
	 * tek nesne static sinif degiskeni olarak
	 * tanimlaniyor.
	 */
	private static Singleton instance = null;

	/*
	 * Double check locking yapabilmek
	 * icin kullanilan nesne.
	 */
	private static Object lock = new Object();


	/**
```

```
         * Baska siniflarin new Singleton()
         * seklinde nesne olusturmalarini,
         * sinif konstruktorünü private
         * yaparak engellemis oluyoruz.
         *
         */
        private Singleton()
        {
                System.out.println("singletion init()");
        }

        /**
         * Singleton sinifindan olusturulabilen
         * tek nesneye ulasmak icin instance()
         * metodu kullanilir.
         *
         * @return Singleton static Singleton nesnesi
         */
        public static Singleton instance()
        {
                if(instance == null)
                {
                        // Double checked locking
                        synchronized (lock)
                        {
                                if(instance == null)
                                {
                                        instance = new Singleton();
                                }
                        }
                }
                return instance;
        }

        /**
         * Singleton sinifinda bulunan
         * bir metod.
         */
        public void printThis()
        {
                System.out.println(this);
        }
}
```

Bir singleton sınıfın taşıması gereken bazı özellikler vardır. Bunlar:

- Sınıf konstruktörlerinin private olması gerekiyor. Konstruktörleri private olan bir sınıftan, başka bir sınıf new operatörü ile nesne oluşturamaz.

- Singleton sınıfından sadece bir tane nesne oluşturulması gerektiği için, oluşturulması gereken nesneyi sınıfın static değiskeni olarak tanımlamamız gerekiyor. Yukardaki örnekte **private static Singleton instance = null;** şeklinde bu tanımlamayı yapıyoruz.

- Singleton sınıfında instance() isminde static bir metodun olması ve bu metodun static olarak tanımlanmış nesneyi geri vermesi gerekiyor. instance() metodu içinde sınıfın tek nesnesi olacak değişken oluşturulur.

Sınıf bünyesinde bulunan instance() static metodu büyük önem taşımaktadır. Amatörce yazılmış singleton sınıflarında genelde aşağıdakı gibi bir instance() metodu görülebilir:

```
public static Singleton instance()
{
	if(instance == null)
	{
		instance = new Singleton();
	}
	return instance;
}
```

Java multi threaded bir sistem olduğu için, yukardaki instance() metodunda new ile birden fazla Singleton nesne oluşturulabilir. Gözümüzde çalışan iki thread canlandırarak, bu sorunun nasıl oluştuğunu inceliyelim: Sistemde T1 ve T2 isimlerinde iki thread mecvut. T1 ilk olarak intance() metoduna girdi ve if(instance == null) satırını geçtikten sonra, java thread scheduler tarafından bloke edildi. Bu noktada T1 instance = new Singleton() yapamadan devre dışı kalmış oldu. Akabinde T2 çalışmaya başladı ve if(instance == null) satırını geçti ve zamanı yeterli olduğu için instance = new Singleton() yaptı. Zamanı dolduğu için T2 thread scheduler tarafından bloke edildi. Kontrol tekrar T1 threadine geçti. T1, T2'nin daha önce new Singleton() yaptığından habersiz olduğu için, instance = new Singleton() yaparak tekrar instance değişkenini oluşturdu. Bu durumda aynı nesne arka arkaya iki sefer oluşturulmuş oldu. Bu durum, sistem tarafından kullanılan kıymetli kaynakların oluşturulmasında sorun yaratabilir. Singleton sınıfı ve barındırdığı nesne sadece ve sadece bir defa oluşturulmalı. Yukarda yer alan sorunu ortadan kaldırmak için double checked locking[1] mekanizması kullanılabilir. Bizim örneğimizde yeralan instance() metodu double checked locking kullanmaktadır:

```
	/**
	 * Singleton sinifindan olusturulabilen
	 * tek nesneye ulasmak icin instance()
	 * metodu kullanilir.
	 *
	 * @return Singleton static Singleton nesnesi
	 */
	public static Singleton instance()
	{
		if(instance == null)
		{
			// Double checked locking
			synchronized (lock)
			{
				if(instance == null)
				{
					instance = new Singleton();
				}
			}
		}
		return instance;
	}
```

[1] Bakınız: http://www.ibm.com/developerworks/library/j-dcl.html

Tekrar T1 ve T2 threadlerimizi kullanarak, nasıl double checked locking metodunun, iki sefer new Singleton() yapılmasını önlediğini görelim:

T1 instance() metoduna girer ve if(instance == null) satırını değerlendirir. Burada T1 thread scheduler tarafından bloke edilebilir. Biz edilmediğini ve T1'in devam ettigini düşünelim. T1 synchronized bloguna girdigi andan itibaren, bu bloga T2'nin girmesi imkansızdır. Şimdi T1'in syncronozed bloğuna girdikten sonra bloke edildigini düşünelim. Bu durumda T2 sadece synchronized metodunun önüne kadar gelip, T1 bu bloktan çıkana kadar beklemek zorundadır, çünkü T1 lock nesnesinin synchronized lock mekanızmasını elinde tutmaktadır. T1 bunu geri vermedigi sürece, yani synchronized bloğundan çıkmadığı sürece, diğer threadler bu bloğa girmezler. T1 kontrülü tekrar eline aldıktan sonra, emin olmak icin bir daha if(instance== null) satiri ile, nesnenin oluşturulup, oluşturulmadığını kontrol eder. Eger T1 synchronized metoduna girmeden bloke olmuşsa, T2 bloğa girmiş ve new Singleton() yapmış olabilir. Tekrar kontrol edildiği için bu mekanizmanin ismi double checked locking'dir. Metoda ilk girişte if(instance == null) ile kontrol edilir ve kontrol synchronized bloğunda tekrarlanır. Bu şekilde singleton nesnesinin birden fazla init edilmesi önlenmiş olur.

Bir test sınıfı ile,singleton nesnesi kontrol edilebilir.

```
package org.javatasarim.pattern.singleton;


/**
 * Singleton tasarim sablonu test sinifi
 *
 * @author Oezcan Acar
 *
 */
public class Test
{
	/**
	 * main
	 * @param args
	 */
	public static void main(String[] args)
	{
		for(int i =0; i < 10; i ++)
		{
			Singleton.instance().printThis();
		}
	}
}
```

Ekran cikisi:

```
singletion init()
org.javatasarim.pattern.singleton.Singleton@1add2dd
org.javatasarim.pattern.singleton.Singleton@1add2dd
org.javatasarim.pattern.singleton.Singleton@1add2dd
org.javatasarim.pattern.singleton.Singleton@1add2dd
org.javatasarim.pattern.singleton.Singleton@1add2dd
org.javatasarim.pattern.singleton.Singleton@1add2dd
org.javatasarim.pattern.singleton.Singleton@1add2dd
org.javatasarim.pattern.singleton.Singleton@1add2dd
```

```
org.javatasarim.pattern.singleton.Singleton@1add2dd
org.javatasarim.pattern.singleton.Singleton@1add2dd
```

Ekran çıktısında görüldüğü gibi, singleton sınıfından sadece bir nesne oluşturuluyor. Bunu **singleton init()** çıktısı ile görüyoruz. Daha sonraki satırlarda aynı nesne kullanılarak, printThis() metodunun çıktısı ekrana geliyor.

Singleton tasarım şablonu ne zaman kullanılır?

- Sistem bünyesinde bir sınıftan sadece bir nesne üretilerek bu nesnenin kullanılması gerektiği durumlarda singleton tasarım şablonu kullanılır.

İlişkili tasarım şablonları:

- Abstract Factory, Builder ve Prototype tasarım şablonlarında olabileceği gibi birçok tasarım şablonu Singleton tasarım şablonu kullanılarak implemente edilir.