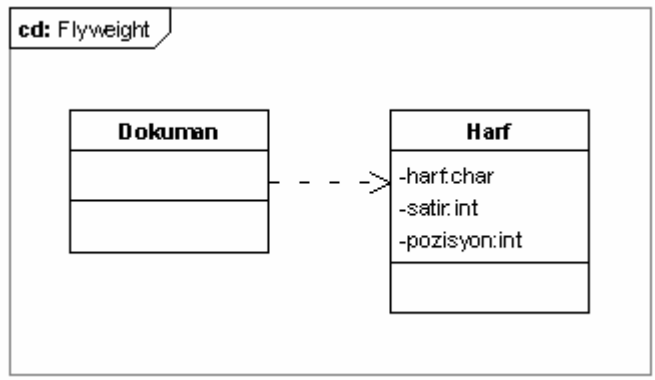
Flyweight (Sinek Siklet) Tasarım Şablonu

KurumsalJava.com

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

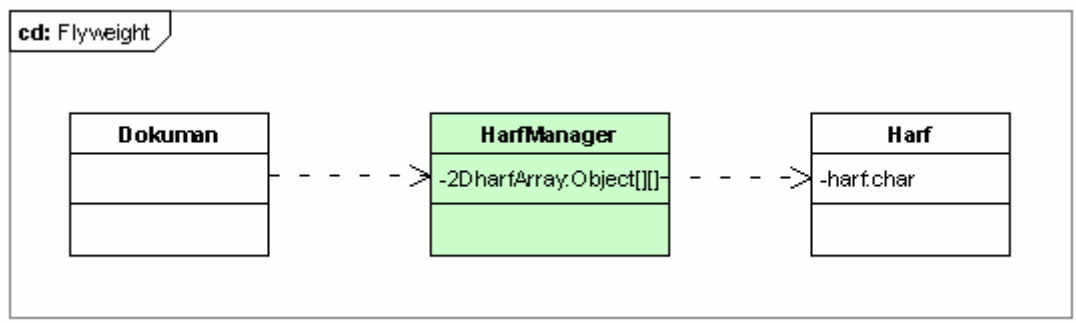
Java dilinde yazılan programlar içinde sınıflar ve bu sınıflardan oluşturulan nesneler kullanır. Bazen aynı sınıftan yüzlerce, belki binlerce nesne oluşturup, kullanıyor olabiliriz. Bu gibi durumlarda çok nesne oluşturulduğu için sistem performansı kötüye gidebilir. Flyweight tasarım şablonu kullanılarak, kullanılan nesne adedini aşağıya çekebiliriz.

Bu satırlar oluşurken, büyük bir ihtimalle kullandığım editör flyweight tasarım şablonunu kullanıyor olabilir. Yazdığım her cümle kelimelerden, her kelime birden fazla harften oluşmaktadır. Kullandığım editörün Java dilinde yazıldığını ve her harf için bir nesne kullandığını farzedersek, bir satırlık doküman için 80 ila 100 arası harf nesnesi oluşturması gerekir. 100 satırlık bir doküman için bu 10.000 civarı harf nesnesinin oluşturulması anlamına gelir.



Uml diagramında da görüldüğü gibi her harf bulunduğu satırı ve pozisyonu bilmektedir.

Flyweight tasarım şablonu yardımı ile gerekli harf nesne adedini elliye düşürebiliriz. Her kelime ve satırda aynı harfler mutlaka birden fazla kullanılacaktır. Örneğin “kullanılacaktır” kelimesi içinde üç L harfı, üç A harfi, iki ı harfi vs. mecvuttur. L harfi için üç yeni L harfi nesnesi oluşturmak yerine, L harfini bir kere oluşturup, üç yerde kullanabiliriz. Böylece onlarca L harfi kullanmaktansa, L harfini bir kere oluşturup, L harfinin gerektiği her yerde kullanabiliriz. Aynı işlemi alfabedeki diğer harf, rakam ve işaretler içinde yaptığımız zaman, sistem bünyesinde kullanılan harf nesnesi adedi elliyi geçmeyecektir. Her harf nesne olarak bir kere mevcut olacağından, harfin nerede kullanıldığı bilgisini başka bir mekanizma aracılığıyla sabitlememiz gerekiyor. İmplementasyonumuzu aşağıdaki şekilde değiştiriyoruz.




```
package org.javatasarim.pattern.flyweight;

/**
 * Alfabenin bir harfini temsil eder.
 *
 * @author Oezcan Acar
```

```
 *
 */
public class Harf
{
	private String harf;

	public Harf(String h)
	{
		this.harf = h;
	}

	public String getHarf()
	{
		return harf;
	}

	public void setHarf(String harf)
	{
		this.harf = harf;
	}
}
```

Tüm sistem bünyesinde, her harf sadece bir nesne olarak kullanılacağı için bu nesnelerin sahip olduğu satır ve o satırdaki pozisyon değişkenlerini yok etmemiz gerekiyor. Aynı harf birden fazla yerde kullanıldığı için satır ve pozisyon değişkenleri anlamlarını yitiriyorlar. Bir harfin nerede kullanıldığını bilmek için HarfManager isminde yeni bir sınıf programlıyoruz. HarfManager bünyesinde harfArrray2D isminde iki boyutlu bir Harf array tanımlıyoruz. Bu oluşan dokümanın satırlarını ihtiva eden bir yapı olacaktır. Birinci boyut satırları, ikinci boyut her satırın belirli bir pozisyonunu adreslemek için kullanılır. Örneğin 5. satırın 16. pozisyonunda L harfini kullanmak istiyorsak

```
harfArray2D[5][16] = Harf.getHarf("L");
```

şeklinde bir tanımlama yapabiliriz.

```
package org.javatasarim.pattern.flyweight;

import java.util.ArrayList;

/**
 * Flyweight tasarım şablonunu kullanarak,
 * kullanilan harf nesne adedi alfabedeki harf
 * adedine indirilir.
 *
 * @author Oezcan Acar
 *
 */
public class HarfManager
{
	/**
	 * Singleton degisken
	 */
	private static final HarfManager manager =
						new HarfManager();

	/**
```

```
	 * Harf nesnelerinin yeraldigi liste
	 */
	private ArrayList<Harf> harfList = new ArrayList<Harf>();


	/**
	 * Doküman satirlarinin ve satirlarda kullanilan
	 * harf pozisyonlarinin tutuldugu iki boyutlu array.
	 */
	private Harf[][] document = new Harf[50][50];


	/**
	 * Singleton oldugu için konstrüktörü private
	 * olarak deklare ediyoruz.
	 */
	private HarfManager()
	{

	}

	public static final HarfManager instance()
	{
		return manager;
	}


	/**
	 * Bir harf nesnesi olusturur.
	 * Mevcut harf nesneleri harfList
	 * degiskeninde tutulur. Bu listede
	 * yeralmayan harfler new ile olusturularak
	 * listeye dahil edilirler.
	 *
	 * @param h Harf
	 * @return Harf harf nesnesi
	 */
	public static Harf getHarf(String h)
	{
		Harf harf = null;
		for(int i=0; i < manager.harfList.size(); i++)
		{
			Harf temp = manager.harfList.get(i);

			if(temp.getHarf().equals(h))
			{
				harf = temp;
				break;
			}
		}

		// harf bulunamadi, listeye ekliyoruz.
		if(harf == null)
		{
			harf = new Harf(h);
			manager.harfList.add(harf);
		}
		return harf;
	}


	/**
```

```java
	 * Dokümana bir satir ekler.
	 * @param satir Doküman satiri
	 * @param satirNo Dokümandaki satir no
	 */
	public void addSatir(String satir, int satirNo)
	{
		/**
		 * Bir döngü içinde satirin tüm harflerini
		 * ziyaret ederek, dokümanin satirini harf nesneleri
		 * ile olusturuyoruz.
		 */
		for(int i=0; i < satir.length(); i++)
		{
			String harf = satir.substring(i, i+1);
			document[satirNo][i] = getHarf(harf);
		}
	}


	/**
	 * Doküman ekranda görüntülenir.
	 */
	public static void getDokument()
	{
		for(int x=0; x < 50; x++)
		{
			for (int y=0; y < 50; y++)
			{
				if(manager.document[x][y] != null)
				{
					System.out.print((
							manager.document[x][y])
							.getHarf());
				}
				else break;
			}
			if(manager.document[x][0] != null)
			{
				System.out.println("");
			}
		}
	}


	public ArrayList<Harf> getHarfList()
	{
		return harfList;
	}

	public void setHarfList(ArrayList<Harf> harfList)
	{
		this.harfList = harfList;
	}

	public Harf[][] getDocument()
	{
		return document;
	}

	public void setDocument(Harf[][] document)
	{
		this.document = document;
```

```
	}
}
```

HarfManager sınıfını bir singleton olarak tanımlıyoruz. Static olarak tanımlanmış olan *getHarf()* metodu ile harf nesneleri oluşturulmaktadır. Harf nesneleri oluşturulduktan sonra harfList isimli listede tutulur. *addSatir()* metodu bünyesinde dokümana bir satır eklenir. satirNo doküman içinde hangi satırın kullanılacağına işaret eder.

```
package org.javatasarim.pattern.flyweight;

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

	public static void main(String[] args)
	{
		/**
		 * Dokümanin ilk satiri
		 */
		String ilkSatir ="Bu ilk satirdir";

		/**
		 * Dokümanin ikinci satiri
		 */
		String ikinciSatir ="Bu ikinci satirdir";

		/**
		 * Dokümana ilk satiri ekliyoruz
		 */
		HarfManager.instance().addSatir(ilkSatir, 1);

		/**
		 * Dokümana ikinci satiri ekliyoruz.
		 */
		HarfManager.instance().addSatir(ikinciSatir, 2);

		HarfManager.getDokument();
	}
}
```

Test programı ile ekran çıktısı şöyle olacaktır:

```
Bu ilk satirdir
Bu ikinci satirdir
Olusturulan harf nesnesi adedi: 13
```

ilkSatir ve ikinciSatir String'leri ile dokümanın birinci ve ikinci satırları tanımlanır. **HarfManager** sınıfının sahip olduğu *addSatir()* metodu ile bu satırlar **HarfManager** sınıfı bünyesinde tanımlanmış olan dokümana (document değişkeni) eklenir. **HarfManager**.*getDokument()* metodu ile iki boyutlu array (harfArray2D değişkeni) içinde

yer alan satırlar ekranda görüntülenir. İki satır için toplam 29 harf kullanıldı. harfList listesinde 13 harf bulunmaktadır. Örneğin i harfi 8 değişik yerde kullanılmıştır. Flyweight tasarım şablonu uygulandığı için 8 i harfi nesnesi oluşturmak yerine, sadece 1 kere oluşturarak, 8 değişik yerde kullanmış olduk.

Flyweight tasarım şablonu ne zaman kullanılır?

- Aplikasyon için kullanılan nesne adedini signifikant bir şekilde düşürmek için Flyweight tasarım şablonu kullanılır. Nesne adedinin düşmesi ile performans iyileşir ve daha az hafıza alanı kullanılır.

İlişkili tasarım şablonları:

- State ve Strategy tasarım şablonlarında kullanılan nesnelerin Flyweight olarak implemente edilmelerinde fayda vardır.

EOF (End Of Fun)
Özcan Acar