Loose Coupling (LC)
Esnek Bağ Tasarım Prensibi

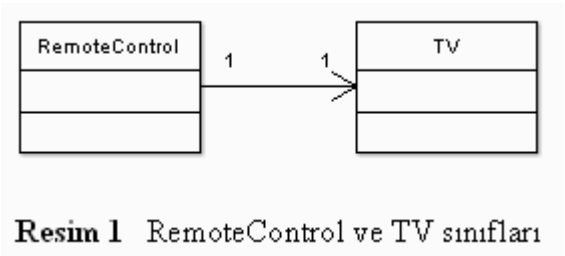
KurumsalJava.com

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

Bir program bünyesinde, tanımlanan görevlerin yerine getirilebilmesi için birden fazla nesne görev alır. Bu nesneler birbirlerinin sundukları hizmetlerden faydalanarak kendi görevlerini yerine getirirler. Bu durumda nesneler arası bağımlılıklar oluşur. Bir nesne kullandığı diğer bir nesne hakkında ne kadar fazla detay bilgiye sahip ise, o nesneye olan bağımlılığı o oranda artar. Oluşan her bağımlılık bir sınıf için dolaylı olarak yapısal değiştirilme rizikosunu artırır, çünkü bağımlı olduğu sınıf üzerinde yapılan her değişiklik kendi yapısında değişikliğe neden olacaktır. Bu durum programın genel olarak kırılgan bir hale gelmesini kolaylaştıracaktır.

Buradan "Eğer bağımlılık varsa, sorun var, bu yüzden bağımlılıkların ortadan kaldırılması gerekmektedir" sonucunu çıkartabiliriz. Nesneye yönelik tarzda design edilmiş bir program içinde bağımlılıkları ortadan kaldırmak imkansızdır, çünkü nesnelerin olduğu yerde interaksiyon ve bağımlılık olmak zorundadır. Bağımlılıkları ortadan kaldıramıyorsak, o zaman onları kontrol altına almamız işimizi kolaylaştıracaktır. Eğer bana soracak olursanız, yazılım disiplininin özü de burada saklıdır: Yazılımcı olarak kullandığımız tüm metotların temelinde bağımlılıkların kontrolü ve yönetilmesi yatmaktadır. İyi bir tasarım oluşturmak için sarf ettiğimiz efor, bağımlılıklara hükmetmek isteyişimizden kaynaklanmaktadır, yani iyi bir tasarım , kontrol edilebilir bağımlılıkları beraberinde getirdiği için iyidir, kendimizi programcı olarak iyi hissetmemizi sağladığı için değil! Biliyoruz ki kötü bir tasarım nesneler arası yüksek derecede bağımlılığa sebep vereceği için programcı olarak hayatımızı zorlaştıracaktır. Bu yüzden sahip olduğumuz tüm teknik yeteneklerimizle bağımlılığa karşı bir savaş veririz. Onu yenmemiz mümkün olmasa da, bize zarar vermeyecek şekilde kontrol altına almamız mümkündür. Bunun yolu da çevik tasarımdan geçmektedir.

Anladığımız kadarıyla bağımlılıkları mantıklı bir çerçevede ortadan kaldırmamız imkansız! Peki bağımlılıkları nasıl kontrol altına alabiliriz? **Esnek bağımlılıklar oluşturarak!** Esnek bağımlılık oluşturmak demek, nesneler arası bağların oluşmasına izin vermek, ama sınıflar üzerinde yapılan yapısal değişikliklerin bağımlı sınıflar üzerinde yapısal değişikliğe sebep vermesini engellemek demektir. Bunu bir örnek vererek açıklayalım.



Resim 1 RemoteControl ve TV sınıfları

RemoteControl (tv uzaktan kumanda aleti) ve TV (televizyon) sınıflarının arasında tek yönlü (uni directional) bir bağ oluşmuştur, çünkü bir RemoteControl nesnesi görevini yerine getirebilmek için bir TV nesnesine ihtiyaç duymaktadır. Nesneler arası bağımlılıkların yönü vardır. Resim 1 de bu bağımlılığın yönünün RemoteControl sınıfından TV sınıfına doğru olduğunu görmekteyiz. Bu tek yönlü bir bağdır ve RemoteControl sınıfı bünyesinde TV tipinde bir sınıf değişkeni barındırarak bağı oluşturur. Sınıflar arası bağlar karşılıklı da (bi directional) olabilir. İki taraflı bağlarda sınıflar, bir sınıf değişkeni aracılığıyla karşılıklı olarak birbirlerine işaret ederler. RemoteControl nesnesi bir TV nesnesi olmadığı sürece işe yaramaz. Bu yüzden bu iki sınıf arasında direk bağlantı oluşturulmuştur. Böyle bir bağımlılık aşağıdaki sorunların oluşmasına sebep vermektedir:

- RemoteControl nesnesi, bir TV nesnesi olmadığı sürece kendi görevini yerine getiremez, bu yüzden tek başına bir işe yaramaz. Mutlaka ve mutlaka, var

olabilmesi için bir TV nesnesine ihtiyaç duymaktadır. Bu durumda RemoteControl sınıfını başka bir alanda kullanmak istediğimizde TV sınıfını da yanına koymak zorundayız. Böyle bir bağımlılık RemoteControl sınıfının tek başına başka bir alanda tekrar kullanılmasını engellemektedir. Buradan söyle bir sonucu çıkartıyoruz: Program parçalarının (sınıflar) tekrar kullanılabilir olabilmeleri için diğer sınıflara olan bağımlılıklarının düşük seviyede olması gerekmektedir. Esnek bağımlılık olmadan bir kere yazılan kodun tekrar kullanımı çok güçtür.

- TV sınıfı bünyesinde meydana gelen yapısal değişiklikler RemoteControl sınıfını doğrudan etkileyecektir. Böyle bir bağımlılık RemoteControl sınıfının yapısal değişikliğe uğrama rizikosunu artırır.
- RemoteControl nesnesi sadece bir TV nesnesini (yani bir televizyonu) kontrol edebilir. Oysaki bir evde uzaktan kumanda edilebilecek başka aletler de olabilir. Böyle bir bağımlılık RemoteControl nesnesini sadece bir TV nesnesiyle beraber çalışmaya mahkum eder. Uzaktan kumanda edilebilen aletler için başka RemoteControl sınıflarının oluşturulması gerekmektedir, örneğin bir CD çalıcıyı kontrol etmek için CDPlayerRemoteControl isminde bir sınıfı oluşturmak zorundayız.

Kod 1 RemoteControl.java

```
package org.cevikjava.design.loosecoupling;

/**
 * Bir televizyonu uzaktan kuman etme
 * aletini simule eden sınıf.
 *
 * @author Oezcan Acar
 *
 */
public class RemoteControl
{
     /**
      * Kontrol edilen televizyon
      */
     private TV tv = new TV();

     /**
      * Televizyonu acmak
      * için kullanilan metot.
      *
      */
     public void tvOn()
     {
          tv.on();
     }


     /**
      * Televizyonu kapatmak
      * için kullanilan metot.
      */
```

```
	public void tvOff()
	{
		tv.off();
	}
}
```

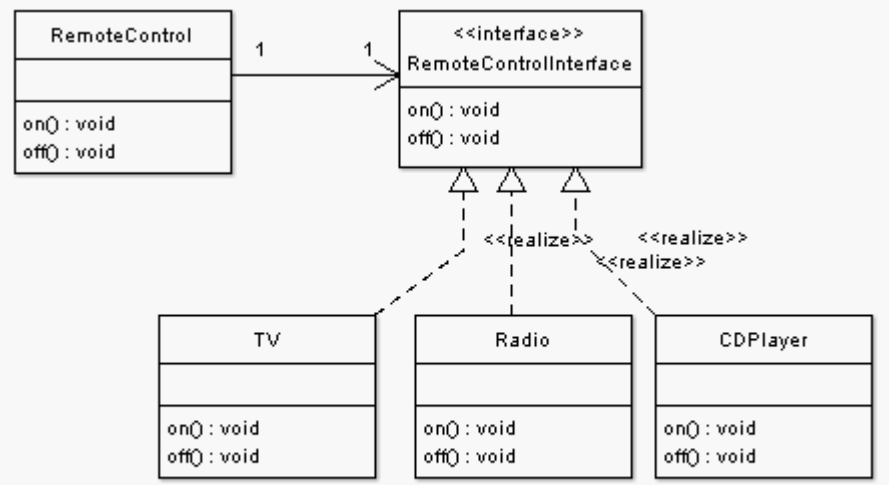
RemoteControl sınıfı bünyesinde TV tipinde bir sınıf değişkeni (tv) barındırdığı için kendisini TV sınıfına bağımlı kılar.

Kod 2 TV.java

```
package org.cevikjava.design.loosecoupling;

/**
 * Bir televizyonu simule eden sınıf.
 *
 * @author Oezcan Acar
 *
 */
public class TV
{
	/**
	 * Televizyonu acmak için
	 * kullanilan metot.
	 *
	 */
	public void on()
	{
		System.out.println("TV acildi.");
	}

	/**
	 * Televizyonu kapatmak için
	 * kullanilan metot.
	 *
	 */
	public void off()
	{
		System.out.println("TV kapandi");
	}
}
```



Resim 2 Esnek bağ için gerekli sınıf yapısı

Nesneler arası kuvvetli bağların, programın bakımı, geliştirilmesi ve kodun tekrar kullanımını negatif yönde etkilediğini gördük. Bu sorunu ortadan kaldırabilmek için sınıfların ilişkileri üzerinde yapısal değişikliğe gitmemiz gerekiyor. Esnek bağ oluşturabilmek için Abstract (soyut) yada Interface[1] sınıflarından faydalanabiliriz. Resim 2 de görüldüğü gibi **RemoteControlInterface** ismini taşıyan bir Interface sınıf ile RemoteControl ve bu sınıfın kontrol etmek istediği aletler arasında esnek bir bağ oluşturuyoruz. Böyle bir yapının esnekliği nereden gelmektedir, bunu yakından inceleyelim.

Kod 3 RemoteControlInterface.java

```
package org.cevikjava.design.loosecoupling.design;

/**
 * RemoteControlInterface sınıfı
 *
 * @author Oezcan Acar
 *
 */
public interface RemoteControlInterface
{
     /**
      * Bu sınıfı implement eden
      * bir aleti acmak için
      * kullanilan metot.
      *
      */
     void on();


     /**
      * Bu sınıfı implement eden
      * bir aleti kapatmak için
```

[1] Interface ve soyut (abstract) sınıflar hakkında detaylı bilgiyi Java Tasarım Şablonları ve Yazılım Mimarileri isimli kitabımdan edinebilirsiniz. (Pusula 183 05/2007 – ISBN = 978-9944-711-14-2)

```
        * kullanilan metot.
        *
        */
      void off();

}
```

Interface sınıflar bünyesinde sadece metot gövdeleri tanımlanır. Alt sınıflar kullanılarak interface sınıfında tanımlanmış olan metotlar implemente edilir. Bu sayede interface sınıfında tanımlanmış metotlar için değişik sınıflarda değişik tarzda implementasyonlar yapmak mümkündür. Herhangi bir sınıf **implements** direktifini kullanarak bir interface sınıfını implemente edebilir. Interface sınıfında tanımlanmış olan tüm metotların alt sınıflarca implemente edilmesi gerekmektedir.

Resim 2 de görüldüğü gibi RemoteControl sınıfı direk TV sınıfı kullanmak yerine, RemoteControlInterface ismini taşıyan interface sınıfı ile beraber çalışmaktadır. Böyle bir yapılanma ile RemoteControl ve bu sınıfın kullanmak istediği somut sınıflar (örneğin TV) arasına bir set çekmiş olduk. RemoteControl sınıfı bu setin arkasında yer alan sınıfları, bunlar RemoteControlInterface sınıfını implemente eden sınıflardır, tanımamaktadır ve tanımak zorunda değildir. RemoteControl sınıfının tanıması gereken tek sınıf RemoteControlInterface sınıfı ve bu interface sınıfının dış dünyaya sunduğu metotlardır. RemoteControl sınıfı böylece dolaylı olarak, RemoteControlInterface sınıfını implemente eden her sınıfı kullanabilir hale gelmektedir. Böylece RemoteControl sınıfının somut sınıflara olan bağımlılığı bir interface sınıf kullanılarak ortadan kaldırılmıştır.

Kod 4 RemoteControl.java

```
package org.cevikjava.design.loosecoupling.design;

/**
 * RemotControlInterface sınıfını
 * implemente eden sınıfları
 * kontrol edebilen sınıf.
 *
 * @author Oezcan Acar
 *
 */
public class RemoteControl
{
      /**
       * Delegasyon islemi için RemoteControlInterface
       * tipinde bir sınıf degiskeni tanimliyoruz.
       * Tüm islemler bu nesnenin metodlarina
       * delege edilir.
       */
      private RemoteControlInterface remote;

      /**
       * Sinif konstuktörü. Bir nesne oluşturma islemi
       * esnasında kullanilacak RemoteControlInterface
```

```
         * implementasyonu parametre olarak verilir.
         *
         * @param _remote RemoteControlInterface
         */
        public RemoteControl(RemoteControlInterface _remote)
        {
             this.remote = _remote;
        }

        /**
         * Aleti acmak
         * için kullanilan metot.
         *
         */
        public void on()
        {
             remote.on();
        }


        /**
         * Aleti kapatmak
         * için kullanilan metot.
         */
        public void off()
        {
             remote.off();
        }

}
```

RemoteControl sınıfını RemoteControlInterface sınıfını kullanacak şekilde değiştiriyoruz. Bu amaçla RemoteControl sınıfında RemoteControlInterface tipinde bir sınıf değişkeni (remote) tanımlıyoruz. Sınıf konstruktörü RemoteControlInterface tipinde bir parametre kabul etmektedir. Böylece RemoteControl sınıfından bir nesne oluştururken, istediğimiz tipte bir RemoteControlInterface implementasyon sınıfı kullanabiliriz.

RemoteControl sınıfı on() ve off() isminde iki metot tanımlamaktadır. Bu metotlar RemoteControlInterface sınıfında yer alan on() ve off() metotları ile karıştırılmamalıdır. RemoteControl sınıfı sahip olduğu metotlara ac() ve kapat() isimlerini de verebilirdi. Bu metotlar içinde delegasyon yöntemiyle sınıf değişkeni olan remote nesnesi kullanılmaktadır. Bu sayede kullanılan RemoteControlInterface implementasyon sınıfının (örneğin TV) on() ve off() metotları devreye girecektir.

Kod 5 TV.java

```
package org.cevikjava.design.loosecoupling.design;

/**
 * Bir televizyonu simule eden sınıf.
```

```
 * RemoteControlInterface sınıfını
 * implemente ederek bir RemoteControlInterface
 * haline gelir.
 *
 * @author Oezcan Acar
 *
 */
public class TV implements RemoteControlInterface
{

     /**
      * Televizyonu acmak için
      * kullanilan metot.
      *
      */
     public void on()
     {
          System.out.println("TV acildi.");
     }

     /**
      * Televizyonu kapatmak için
      * kullanilan metot.
      *
      */
     public void off()
     {
          System.out.println("TV kapandi");
     }
}
```

TV sınıfı RemoteControlInterface sınıfını implemente etmektedir. Bu sebepten dolayı RemoteControlInterface sınıfında tanımlanmış olan on() ve off() metotlarına sahiptir. TV sınıfı RemoteControlInterface sınıfını implemente etmediği sürece RemoteControl sınıfı tarafından kullanılamaz.

Kod 6 Test.java

```
package org.cevikjava.design.loosecoupling.design;

/**
 * Test sınıfı
 *
 * @author Oezcan Acar
 *
 */
public class Test
{
```

```
    public static void main(String[] args)
    {
        RemoteControlInterface rci = new TV();
        RemoteControl control = new RemoteControl(rci);
        control.on();
        control.off();
    }
}
```

Test.main() bünyesinde ilk önce kullanmak istediğimiz alet nesnesini, (TV) ve akabinde RemoteControl nesnesini oluşturuyoruz. RemoteControl konstruktör parametresi olarak bir satır önce oluşturduğumuz TV nesnesini almaktadır. RemoteControl bünyesinde yer alan on() ve off() metotları ile televizyonu açıp, kapatabiliriz. Ekran çıktısı şu şekilde olacaktır:

```
TV acildi.
TV kapandi
```

RemoteControl sınıfının konstruktörü RemoteControlInterface sınıfını implemente eden her sınıfı kabul ettiği için istediğimiz herhangi bir aleti RemoteControl sınıfı ile kontrol edebilir hale geliyoruz.

Oluşturduğumuz yeni tasarımın bize sağladığı avantajlar şöyledir:

- Bir interface sınıf (RemoteControlInterface) kullanarak, RemoteControl ve kontrol etmek istediği aletler (TV, CDPlayer) arasında bir bariyer oluşturduk. RemoteControl sınıfı kontrol etmek istediği aleti tanımak zorunda olmadığı için bu sınıf ve diğerleri arasındaki sıkı bağı çözmüş ve interface sınıf kullanarak daha esnek bir hale getirmiş oluyoruz.
- Bu tarz bir tasarım ile programı gelecekte oluşacak değişiklikleri taşıyabilecek hale getirdik. RemoteControl sınıfı, RemoteControlInterface sınıfını implemente eden her sınıfı kontrol edebilir. RemoteControlInterface sınıfını implemente ederek, sisteme uzaktan kumanda edilebilen yeni aletler ekleyebiliriz.
- RemoteControl sınıfı, RemoteControlInterface sınıfının implemente edildiği başka bir ekosistemde tekrar kullanılabilir hale geldi. Esnek bağımlılık oluşturmak, kodun tekrar kullanımını kolaylaştırmaktadır.

İncelediğimiz örneklerde nesneler arası bağın ortadan kaldırılamayacağını ama esnek bağ oluşturma prensibini uygulayarak kontrol edilebilir bir hale getirilebileceklerini gördük. Esnek bağlar oluşturabilmek için Interface yada soyut sınıflardan yararlanabiliriz.

Usta yazılımcılar tasarım prensiplerine ve tasarım şablonlarına (design pattern) hakim olup,onları doğru yerde kullanmasını bilirler. Eğer şimdiye kadar tasarım prensipleri hakkında bir çalışmanız olmadıysa, sizin için yazılım disiplininde bir üst boyutun kapısını aralamış olduk. Tasarım prensiplerini uygulayarak ve tasarım şablonlarını kullanarak konseptüel daha yüksek seviyede çalışabilirsiniz. Bu bölümde yer alan tasarım prensipleri yazılım sürecine olan bakış açınızı tamamen değiştirecek niteliktedir.

EOF (End Of Fun)
Özcan Acar