Open Closed Principle (OCP)
Açık Kapalı Tasarım Prensibi

KurumsalJava.com

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

Yazılım disiplininde değişmeyen birşey varsa o da değişikliğin kendisidir. Birçok program müşteri gereksinimleri doğrultusunda ilk sürümden sonra değişikliğe uğrar. Bu doğal bir süreçtir ve müşteri programı kullandıkça ya yeni gereksinimlerini yada mevcut fonksiyonlar üzerinde adaptasyonları gerekçe göstererek programın değiştirilmesini talep edecektir.

Tanınmış yazılım ustalarından Ivar Jacobson[1] bu konuda şöyle bir açıklamada bulunmuştur:

"All systems change during their life cycles. This must be born in mind when developing systems are excepted to last longer than the first version."

Şu şekilde tercüme edilebilir:

"Her program görev süresince değişikliğe uğrar. Bu ilk sürümden ötesi düşünülen programların yazılımında göz önünde bulundurulmalıdır."

Değişiklik kaçınılmaz olduğuna göre, bir programı gelecekte yapılması gereken tüm değişiklikleri göz önünde bulundurarak, şimdiden buna hazır bir şekilde geliştirmek mümkün müdür? Öncelikle şunu belirtelim ki gelecekte meydana gelecek değişikliklerin hepsini kestirmemiz mümkün değildir. Ayrıca çevik süreçlerde ilerde belki kullanılabileceğini düşündüğümüz fonksiyonların implementasyonu kesinlikle tabudur (istenilmeyen bir durum anlamında). Bu durum bizi programcı olarak gelecekteki değişiklikleri göz önünde bulundurarak, bir nevi hazırlık yapmamızı engeller. Çevik süreç sadece müşteri tarafından dile getirilmiş, kullanıcı hikayesi haline dönüştürülmüş ve müşteri tarafından öncelik sırası belirlenmiş gereksinimleri göz önünde bulundurur ve implemente eder. Kısacası çevik süreç geleceği düşünmez ve şimdi kendisinden beklenenleri yerine getirir. Böylece müşteri tarafından kabul görmeyecek bir sistemin oluşması engellenmiş olur.

Çevik süreç büyük çapta bir tasarım hazırlığı ile start almaz. Daha ziyade mümkün olan en basit şekilde yazılıma başlanır. Her iterasyon başlangıcında implemente edilmesi gereken kullanıcı hikayeleri seçildikten sonra, mevcut yapının yeni gereksinimlere cevap verip, veremeyeceği incelenir. Büyük bir ihtimalle mevcut yapı yeni kullanıcı hikayelerinin implementasyonu için yeterli olmayacaktır. Bu durumda programcı ekip refactoring yöntemleriyle programın yapısını yeni gereksinimleri kabul edecek şekilde modifike eder. Bu esnada tasarım yapısal değişikliğe uğrar. Bu gerekli bir işlemdir ve yapılmak zorundadır, aksi taktirde bir sonraki iterasyon beraberinde getirdiği değişikliklerle programcı ekibini yazılım esnasında zorlayacaktır.

Çevik süreç gelecekte olabilecek değişikleri göz önünde bulundurmadığına göre, programı bu değişikliklerin olumsuz yan etkilerine karşı nasıl koruyabiliriz? Bunun yolu her zaman olduğu gibi yazılım esnasında uygun tasarım prensiplerini uygulamaktan geçmektedir. Eğer çevik süreç bize gelecekte olabilecek değişikliklere karşı destek sağlamıyorsa, bizimde tedbir alarak çevik süreci, çevik prensiplere ters düşmeden takviye etmemiz gerekiyor. Çevik süreci, çevik tasarım prensiplerini kullanarak istediğimiz bir yapıda, bakımı ve geliştirilmesi kolay program yazılımına destek verecek şekilde takviye edebiliriz.

Tekrar bölüm başında sorduğum soruya dönelim ve sorunun cevabını bulmaya çalışalım. Şu şekilde bir soru sormuştum: "Değişiklik kaçınılmaz olduğuna göre, bir programı gelecekte yapılması gereken tüm değişiklikleri göz önünde bulundurarak, şimdiden buna hazır bir şekilde geliştirmek mümkün müdür?" Bu mümkün değil demiştik. Lakin esnek bir tasarım

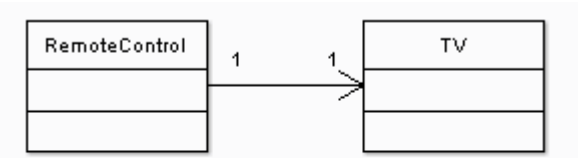
[1] Bakınız: http://en.wikipedia.org/wiki/Ivar_Jacobson

oluşturarak önü açık ve gelecek korkusu olmayan bir program yapısı oluşturabiliriz. Bunu gerçekleştirmek için kullanabileceğimiz prensiplerin başında Open Closed – Açık Kapalı prensibi (OCP) gelmektedir. Bertrand Meyer[2] tarafından geliştirilen bu prensip kısaca şöyle açıklanabilir:

"***Programlar geliştirilmeye açık ama değiştirilmeye kapalı olmalıdır."***

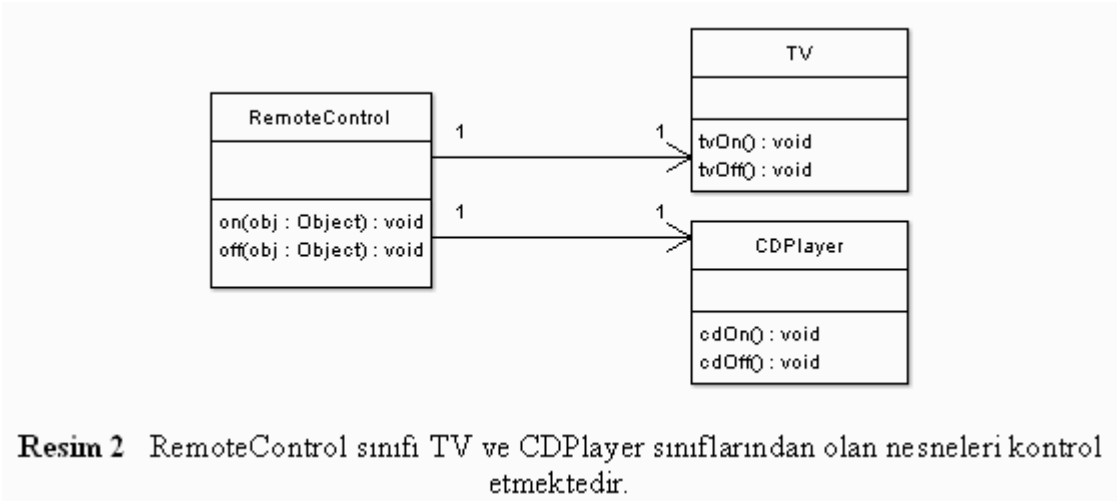
Programı geliştirmek, programa yeni bir davranış biçimi eklemek anlamına gelmektedir. OCP ye göre programlar geliştirmeye açık olmalıdır, yani programı oluşturan modüller yeni davranış biçimlerini sergileyecek şekilde genişletilebilmelidirler. Bir modüle yeni bir davranış biçimi kazandırılarak düşünülen değişiklik sağlanır. Bu yeni kod yazılarak gerçekleştirilir (bu yüzden bu işleme değiştirme değil, genişletme denir), mevcut kodu değiştirerek değil! Eğer kendinizi bir müşteri gereksinimini mevcut kod üzerinde değişiklik yaparken bulursanız, biliniz ki OCP prensibine ters düşüyorsunuz. Kod üzerinde yapılan değişiklik, bir sonraki gereksinimlerinde ayni şekilde implemente edilmesini zorunlu kılacaktır. Bu durum, kodun zaman içinde içinden çıkılmaz ve çok karmaşık bir yapıya dönüşmesini çabuklaştırır.

OCP prensibinin nasıl uygulanabileceğini bir önceki bölümde yer alan RemoteControl – TV örneği üzerinde inceleyelim.



Resim 1 RemoteControl ve TV sınıfları

Resim 1 de görüldüğü gibi RemoteControl sınıfı TV sınıfını kullanarak işlevini yerine getirmektedir. Eğer RemoteControl sınıfını TV haricinde başka bir aleti kontrol etmek için kullanmak istersek, örneğin CDPlayer Resim 2 deki gibi değişiklik yapmamız gerekebilir. Bu noktada esnek bağ prensibini unutarak, resim 2 de yer alan çözümün bizim için yeterli olduğunu düşünelim.



Resim 2 RemoteControl sınıfı TV ve CDPlayer sınıflarından olan nesneleri kontrol etmektedir.

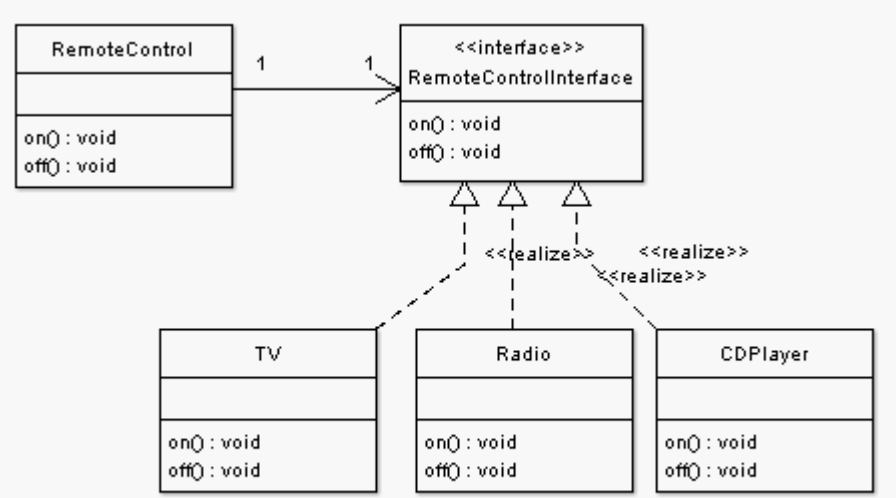
[2] Bakınız: http://en.wikipedia.org/wiki/Bertrand_Meyer

Kod 1 RemoteControl.java

```
package org.cevikjava.design.ocp;

/**
 * TV ve CDPlayer sınıflarından
 * nesneleri kontrol eder.
 *
 * @author Oezcan Acar
 *
 */
public class RemoteControl
{

	/**
	 * Aleti acmak
	 * için kullanilan metot.
	 *
	 */
	public void on(Object obj)
	{
		if(obj instanceof TV)
		{
			((TV)obj).tvOn();
		}
		else if(obj instanceof CDPlayer)
		{
			((CDPlayer)obj).cdOn();
		}
	}


	/**
	 * Aleti kapatmak
	 * için kullanilan metot.
	 */
	public void off(Object obj)
	{
		if(obj instanceof TV)
		{
			((TV)obj).tvOff();
		}
		else if(obj instanceof CDPlayer)
		{
			((CDPlayer)obj).cdOff();
		}

	}
}
```

Bu şekilde oluşturulan bir tasarım OCP prensibine ters düşmektedir, çünkü her yeni eklenen alet için on() ve off() metotlarında değişiklik yapmamız gerekmektedir. OCP böyle bir modifikasyonu kabul etmez. OCP ye göre mevcut çalışır kod kesinlikle değiştirilmemelidir. Onlarca aletin bulunduğu bir sistemde on() ve off() metotlarının ne kadar kontrol edilemez ve bakımı zor bir yapıya bürüneceği çok net olarak bu örnekte görülmektedir.

Bunun yanı sıra RemoteControl sınıfı TV ve CDPlayer gibi sınıflara bağımlı kalacak ve başka bir alanda kullanılması mümkün olmayacaktır. TV ve CDPlayer sınıflar üzerinde yapılan tüm değişiklikler RemoteControl sınıfını doğrudan etkileyecek ve yapısal değişikliğe sebep olacaktır.



Resim 3 Esnek bağ için gerekli sınıf yapısı

Resim 3 de yer alan çözüm OCP ye uygun yapıdadır, çünkü kod üzerinde değişiklik yapmadan programa yeni davranışlar eklemek mümkündür. OCP prensibi, esnek bağ prensibi kullanılarak uygulanabilir. OCP ye uygun RemoteControl sınıfının yapısı şu şekilde olmalıdır:

Kod 2 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();
        }

}
```

on() ve off() metotları sadece RemoteControlInterface tipinde olan bir sınıf değişkeni üzerinde işlem yapmaktadır. Bu sayede if/else yapısı kullanmadan RemoteControlInterface sınıfını implemente etmiş herhangi bir alet üzerinde gerekli işlem yapılabilmektedir.

Bu örnekte on() ve off() metotları değişikliğe kapalı ve tüm program geliştirmeye açıktır, çünkü RemoteControlInterface interface sınıfını implemente ederek sisteme yeni aletleri eklemek mümkündür. Sisteme eklediğimiz her alet için on() ve off() metotları üzerinde değişiklik yapmak zorunluluğu ortadan kalkmaktadır. Uygulanan OCP ve esnek bağ prensibi ile RemoteControl başka bir alanda her tip aleti kontrol edebilecek şekilde kullanılır hale gelmiştir.

Stratejik Kapama (Strategic Closure)

Ne yazık ki bir yazılım sistemini OCP prensibini uygulayarak %100 değişikliklere karşı korumamız imkansızdır. OCP ye uygun olan bir metot müşterinin yeni istekleri doğrultusunda OCP'ye uygun olmayan bir hale gelebilir. Programcı metodu ilk implemente ettiği zaman, gelecekte olabilecek değişiklikler hakkında fikir sahibi olmayabilir. Metot OCP uyumlu implemente edilmiş olsa bile, bu metodun her zaman OCP uyumlu kalabileceği anlamına gelmez.

Eğer kapama tam sağlanamıyorsa, kapamanın stratejik olarak implemente edilmesi gerekir. Programcı implementasyon öncesi meydana gelebilecek değişiklikleri kestirerek, implemente ettiği metotların kapalılık oranını yükseltmelidir. Bu tecrübe gerektiren stratejik bir karardır.

Programcı her zaman ne gibi değişikliklerin olabileceğini kestiremeyebilir. Bu durumda konu hakkında araştırma yaparak, oluşabilecek değişiklikleri tespit edebilir. Eğer olabilecek değişikliklerin tespiti mümkün değilse, beklenen değişiklikler meydana gelene kadar beklenir ve implementasyon yeni değişiklikleri de yansıtacak şekilde OCP uyumlu hale getirilir.

EOF (End Of Fun)
Özcan Acar