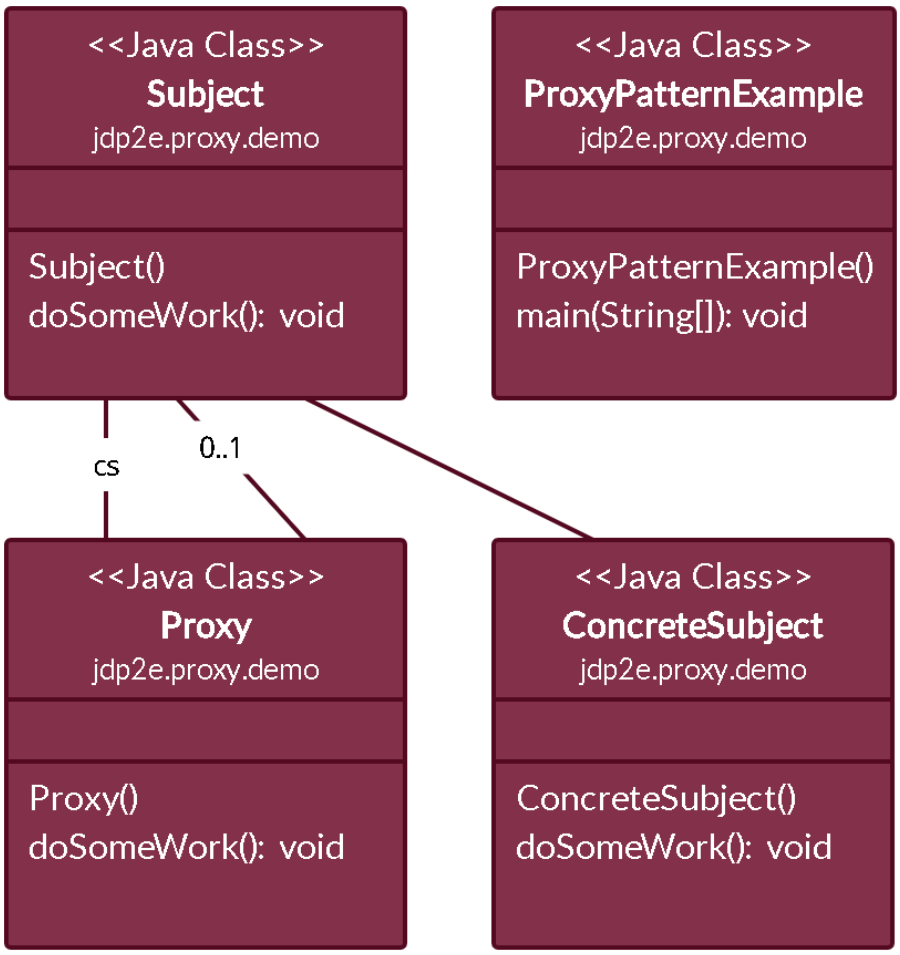
Proxy Tasarım Deseni

Türkçe' deki karşılığı **temsilci** olan Proxy, oluşturulan sınıf üzerinden başka bir nesneyi temsil etmek amacıyla oluşturulmuş bir tasarım desenidir. Bu sayede asıl sınıfta değişiklik yapmadan gereken işlemler gerçekleştirilebilir. Güvenlik, zaman maliyeti, gibi durumlarda daima asıl nesneye erişmek mümkün olmayabilir. Bu durumda Proxy Tasarım Deseni bize yardımcı olur.

Bir sınıfta yoklama alınırken okula gelmeyen bir öğrencinin sesini en yakın arkadaşı taklit edebilir. Programlama dünyasında bu olayın taklidi olan olaylar yaşanabilir. Bazı durumlarda nesnenin birden fazla örneğini oluşturup kullanmamız gerekebilir. Bu sayede ihtiyaç duyulduğunda asıl nesneyi temsil etmek üzere Proxy nesneleri oluşturulabilir. Banka ATM'lerinde bu sistem, banka bilgilerini tutabilmek için Proxy nesnelerini tutmak üzere Proxy Tasarım Deseni'ni kullanır.



Şekil 1

Şekil 1'de gösterilen sınıf diagramında **Subject**, **ProxyPatternExample**, **Proxy** ve **ConcreteSubject** java sınıfları oluşturulmuştur. Subject sınıfındaki doSomeWork() metodu oluşturulmuştur. Proxy ve ConcreteSubject sınıfları

Subject sınıfının mirasını alır (extends eder). Proxy ve ConcreteSubject Sınıfları, doSomeWork() metodunu çağırmaya çalışacaklardır.

1. UYGULAMA

```
abstract class Subject
{
	public abstract void doSomeWork();
}
```

Soyut **Subject** sınıfı ve içerisinde doSomeWork() metodu oluşturduk.

```
class ConcreteSubject extends Subject
{
	@Override
	public void doSomeWork()
	{
	System.out.println("doSomeWork() inside ConcreteSubject is invoked.");
	}
}
```

ConcreteSubject sınıfı **Subject** sınıfını extends ederek **doSomeWork()** metodunu aktardık ve metodu bu sınıf içerisinde çağırdık. İşlemin tamamlandığını belirtmek için ekrana bir çıktı verdik.

```
class Proxy extends Subject
{
	static Subject cs;
	@Override
	public void doSomeWork()
	{
	System.out.println("Proxy call happening now...");

	if (cs == null)
	{
	cs = new ConcreteSubject();
	}
	cs.doSomeWork();
	}
}
```

Proxy sınıfında tekrardan **Subject** sınıfını extends ederek **doSomeWork()** metodunu aktardık ve sınıf içerisinde çağırdık. İşlemin gerçekleştiğini teyit etmek için ekrana çıktı verdik. Subject'te oluşturduğumuz **cs** nesnesinin doluluğunu control ettikten sonra nesneyi çağırdık. Eğer nesne yok ise yeni nesne oluşturduk ve çağırdık.

```
public class ProxyPatternExample
{
	public static void main(String[] args) {
		System.out.println("***Proxy Pattern Demo***\n");
		Proxy px = new Proxy();
		px.doSomeWork();
	}
```

}
Son olarak oluşturduğumuz **ProxyPatternExample** sınıfında ana sınıfı oluşturduk ve burada **px** adında bir Proxy nsenesi oluşturarak Proxy' ı control ettik.

Son durumda ekrana getirdiğimiz çıktı şu şekildedir:

```
***Proxy Pattern Demo***

Proxy call happening now...
doSomeWork() inside ConcreteSubject is invoked.
```

PROXY ÇEŞİTLERİ

1. **Remote Proxy:** Uzaktaki bir nesneye cihazda local temsilci sağlayarak gerekli kontrollerin yapılması gerektiği durumlarda kullanılan proxy çeşididir.

2. **Virtual Proxy:** Genellikle kullanımı maliyetli, üretimi maliyetli ve zor olan nesneleri oluşturmak veya bu nesneleri kullanmak amacı ile kullanılan proxy çeşididir. Örneğin yüksek boyutlu bir resmiin boyutundan dolayı geç yüklenmesi durumunda verilen 'yükleniyor' mesajı ve işlem bittikten sonra resmin gösterilmesini düşünebiliriz.

3. **Protection Proxy:** İstemcinin belirli durumlarda sınıfı çalıştırması gerektiği zaman kullanılır. Örneğin farklı erişim haklarını kontrol ederken kullanılır.

4. **Smart Reference:** Bir nesneye işlemci eriştiğinde, ek temizlik çalışması yapar. Belirli bir anda nesneye yapılan referansları sayar.

2. UYGULAMA

```
abstract class Subject
{
 public abstract void doSomeWork();
}
```

Soyut **Subject** sınıfı ve içerisinde doSomeWork() metodu oluşturduk.

```
class ConcreteSubject extends Subject
{
	@Override
	public void doSomeWork()
	{
	System.out.println("doSomeWork() inside ConcreteSubject is invoked.");
	}
}
```

ConcreteSubject sınıfı **Subject** sınıfını extends ederek **doSomeWork()** metodunu aktardık ve metodu bu sınıf içerisinde çağırdık. İşlemin tamamlandığını belirtmek için ekrana bir çıktı verdik.

```
class Proxy extends Subject
{
```

```
	static Subject cs;
	static int count=0;
	public Proxy()
	{
		cs = new ConcreteSubject();
		count ++;
	}
	@Override
	public void doSomeWork()
	{
		System.out.println("Proxy call happening now...");

		cs.doSomeWork();
	}
}
```

Proxy sınıfında tekrardan **Subject** sınıfını extends ederek **doSomeWork()** metodunu aktardık ve sınıf içerisinde çağırdık. İşlemin gerçekleştiğini teyit etmek için ekrana çıktı verdik. Subject'te oluşturduğumuz **cs** nesnesinin doluluğunu control ettikten sonra nesneyi çağırdık. Eğer nesne yok ise yeni nesne oluşturduk ve çağırdık. Buna ek olarak Proxy'yi kaç kere oluşturduğumuzu control etmek için **count** adında bir değişken ekledik. Bu değişkeni 0'a eşitledik. Her Proxy() metodu çağırıldığında count bir artarak kaç kere Proxy() metodunun çağırıldığını kontrol edecektir.

```
public class ProxyPatternAlternate {
	public static void main(String[] args)
	{
		System.out.println("***Proxy Pattern Demo without lazy
instantiation***\n");
		Proxy px = new Proxy();
		px.doSomeWork();

		Proxy px2 = new Proxy();
		px2.doSomeWork();
		System.out.println("Instance Count="+Proxy.count);
	}
}
```

Ana kısımda bir adet px adında nesne oluşturmuş ve bir çıktı elde etmiştik. Daha sonra ise aynı şekilde ikinci nesneyi oluşturuyoruz ve bu nesnede Proxy.count komutu ile sayaçdaki bilgiyi ekrana yazdırıyoruz.

Son durumda ekrana getirdiğimiz çıktı şu şekildedir:

```
***Proxy Pattern Demo without lazy instantiation***

Proxy call happening now...
doSomeWork() inside ConcreteSubject is invoked
Proxy call happening now...
doSomeWork() inside ConcreteSubject is invoked
Instance Count=2
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

Arraylist ve List kullanabilmek için kütüphaneleri .java uzantılı dosyamıza dahil ettik.

```
abstract class Subject
{
	public abstract void doSomeWork();
}
```

Soyut **Subject** sınıfı ve içerisinde doSomeWork() metodu oluşturduk.

```
class ConcreteSubject extends Subject
{
	@Override
	public void doSomeWork()
	{
		System.out.println("doSomeWork() inside ConcreteSubject is invoked.");
	}
}
```

ConcreteSubject sınıfı **Subject** sınıfını extends ederek **doSomeWork()** metodunu aktardık ve metodu bu sınıf içerisinde çağırdık. İşlemin tamamlandığını belirtmek için ekrana bir çıktı verdik.

```
class ModifiedProxy extends Subject
{
	/*Chapter 1*/
	static Subject cs;
	String currentUser;
	List<String> registeredUsers;

	/*Chapter 2*/
	public ModifiedProxy(String currentUser)
	{
		registeredUsers = new ArrayList<String>();
		registeredUsers.add("Admin");
		registeredUsers.add("Rohit");
		registeredUsers.add("Sam");
		this.currentUser = currentUser;
	}

	/*Chapter 3*/
	@Override
	public void doSomeWork()
	{
		System.out.println("\n Proxy call happening now...");
		System.out.println(currentUser+" wants to invoke a proxy method.");
		if (registeredUsers.contains(currentUser))
		{

			if (cs == null)
			{
				cs = new ConcreteSubject();
			}
			cs.doSomeWork();
```

```
            }
            else
            {
                  System.out.println(currentUser+  , u don’t have access rights.");
            }
      }
}
```

Yukarıdaki kod parçasında ModifiedProxy adında yeni bir sınıf oluşturduk. Kod satırları uzun olduğundan okuyucuya karmaşıklık yaratmaması ve anlaşılır olabilmesi adına kodu ‘Chapter 1’, ‘Chapter 2’ ve ‘Chapter 3’ olarak böldük.

Chapter 1 – Chapter 2 arasında bir adet **currentUser** adında String veri tipli bir değişken oluşturduk. Ardından kullanıcı kayıtılarını tutmak için **registeredUsers** adında yine String veri tipinde liste oluşturduk.

Chapter 2 – Chapter 3 arasında ModifiedProxy() adında bir metod oluşturduk ve bu metotta şimdiki kullanıcıyı almak için currentUser argümanın bu metoda verdik. ArrayList tipinde yeni bir kayıt nesnesi oluşturarak içerisine “Admin”, “Rohit”, “Sam” isimlerinde kullanıcı kaydına üç adet yeni kullanıcı ekledik.

Chapter 3’ den sonra doSomeWork() isimli metotta eğer giriş yapmak istediğimiz kullanıcı kayıtlı kullanıcılarda varsa giriş yaparak somut nesnenin varlığına bağlı olarak bir nesne oluşturduk. Eğer bu nesne yoksa hata mesajı döndürdük.

```
public class ModifiedProxyPatternExample
{
      public static void main(String[] args)
      {
            System.out.println("***Modified Proxy Pattern Demo***\n");
            ModifiedProxy px1 = new ModifiedProxy("Admin");
            px1.doSomeWork();
            ModifiedProxy px2 = new ModifiedProxy("Robin");
            px2.doSomeWork();
      }
}
```

ModifiedProxyPatternExample sınıfında px1 adında bir adet “Admin” adına sahip, bir adet “Robin” adına sahip iki adet nesne oluşturduk. Bu nesnelere bağlı olarak doSomeWork() metodunu her bir nesne için çağırdık.

Son durumda ekrana getirdiğimiz çıktı şu şekildedir:

```
Proxy call happening now...
Admin wants to invoke a proxy method.
doSomeWork() inside ConcreteSubject is invoked.

 Proxy call happening now...
Robin wants to invoke a proxy method.
Sorry Robin , you do not have access rights.
```