Vekil(Proxy) Tasarım Deseni

Nedir?

Proxy tasarım deseni **structural** tasarım desenlerinden biridir. Bir nesneye erişimi kontrol etmek için, **proxy** nesne kullanır. Bu nesne erişilecek nesneyi kontrol eder. Erişilecek nesne remote, yaratılması pahalı olan veya erişim için yetki istiyen gibi türlere sahip olabilir.

Ne zaman Kullanılır?

1. Remote nesne kullanmak istediğimizde kullanılır.(**Remote** proxy, farklı adres uzayında bulunan bir nesneye lokal temsilci sağlar)
2. Masraflı nesnelerin ihtiyaç duyulduğunda yaratılmasını sağlamak için(Örneğin resim yüklenmeden önce yükleniyor yazısı göstermek için. Bazı resimlerin boyutu büyük olduğu için yüklenmesi geç olabilmektedir. Burada belirtilen masraflı ifadesi büyük resim nesnesini temsil eder) kullanılır. Bu tarz proxy'lere **virtual proxy** denir.
3. Client'ın yetkisine göre işlem yapılıp yapılmamasını belirlemek için kullanılır. (**Protection** proxy)

Not: Üç temel **proxy** türü vardır. Bunlar: **Remote, Virtual** ve **Protection** proxy'dir.

Nasıl Kullanılır?

Öncelikle türü **interface** veya **abstract** olan bir **Subject** sınıfı oluşturulur. Bu sınıftan türeyecek veya sınıfı(interface olarak tanımlanırsa) implement edecek **Proxy** ve **RealSubject** sınıfları yaratılır. Bu sınıflar aynı sınıftan türediği için, ortak metod(lar)a sahip olurlar. **Proxy** sınıfına, **Subject** türüne sahip değişken eklenir. **Proxy** ve **RealSubject** aynı sınıftan türediği için **türleri aynı** olur. Bu sayede **Proxy** sınıfı içerisindeki bir metod içerisinden **RealSubject** sınıfındaki metod çağrılır. **Proxy** sınıfında bulunan metoda ise **Client** sınıfı tarafından erişilir. **Client** sınıfı **RealSubject** sınıfına **direkt erişmez,** bunun yerine **Proxy** sınıfı **aracılığı** ile erişir. **Proxy** sınıfı bu özelliği ile **erişimi kontrol eden** sınıf olarak adlandırılır.Ayrıca **RealSubject** sınıfından nesne yaratılma işlemi **Proxy** sınıfında yapılır.

Gerekenler

Proxy sınıfı. **RealSubject** sınıfına erişimi kontrol eder ve düzenler. **Client** sınıfı tarafından erişilir.
Türü **interface** veya **abstract** olan **Subject** sınıfı. **Proxy** ve **RealSubject** sınıfları bu interface'den implement olurlar.
Real Subject: Somut **Subject** sınıfı.
Client sınıfı. **Proxy** sınıfı aracılığı ile **RealSubject** nesnesine erişir. Direkt erişmez.

Örnek Kullanım Alanları

1. java.lang.reflect.Proxy sınıfı
2. java.rmi paketinin tamamı kullanır.

Örnek Uygulama

Proxy ile kullanıcıya göre yetkilendirme işlemi yapılması uygulamasıdır.

Not: Bu uygulamamızda **protection proxy** kullanılmıştır.

CommandExecutor Interface

```
/**
 * Subject interface
 */
public interface CommandExecutor {
    public void runCommand(String cmd) throws Exception;
}
```

Proxy Sınıfı

```
/**
 * Proxy sinifi
 */
public class CommandExecutorProxy implements CommandExecutor {

    private boolean isAdmin;
    private CommandExecutor executor;

    public CommandExecutorProxy(String user, String pwd){
        if("Pankaj".equals(user) && "J@urnalD$v".equals(pwd)){
            isAdmin=true;
        }
        //Goruldugu gibi proxy sinifinda Real Subject nesnesi yaratilir.
        executor = new CommandExecutorImpl();
    }

    @Override
    public void runCommand(String cmd) throws Exception {
        if(isAdmin){
            executor.runCommand(cmd);
        }else{
            if(cmd.trim().startsWith("rm")){
                throw new Exception("rm command is not allowed for non-admin users.");
            }else{
```

```
                executor.runCommand(cmd);
            }
        }
    }
}
```

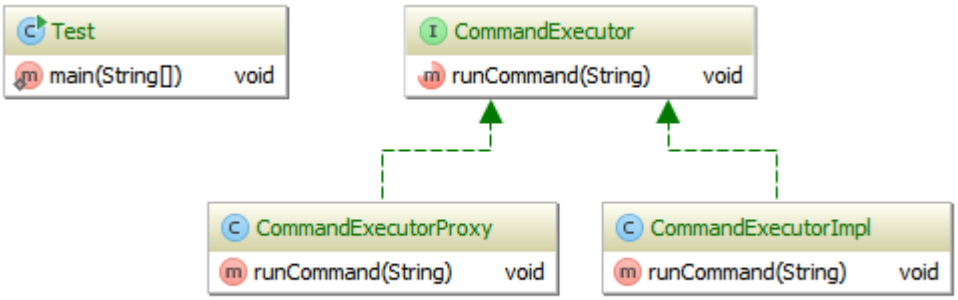
RealSubject Sınıfı

```
/**
 * Real Subject sinifi
 */
public class CommandExecutorImpl implements CommandExecutor {
    @Override
    public void runCommand(String cmd) throws IOException {
        //some heavy implementation
        Runtime.getRuntime().exec(cmd);
        System.out.println("'" + cmd + "' command executed.");
    }

}
```

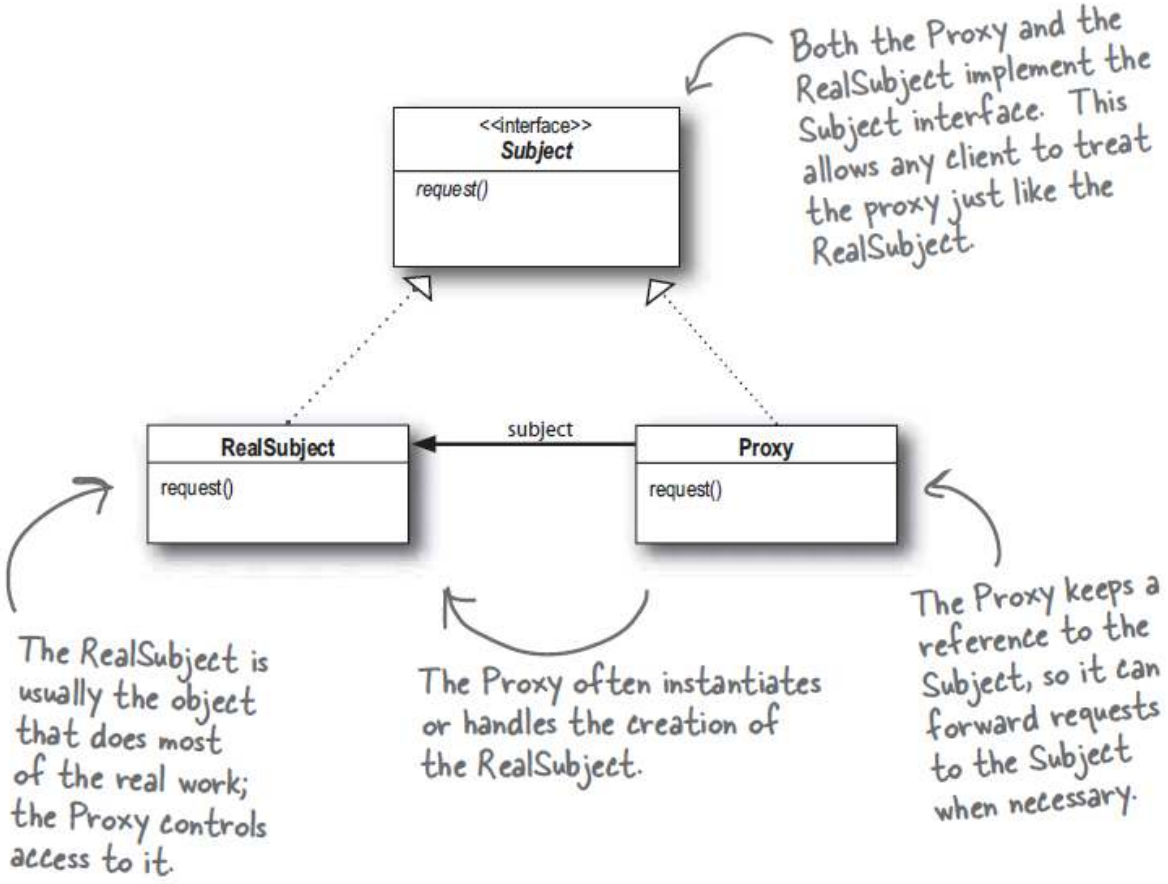
Client Sınıfı

```
/**
 * Client sinifi
 */
public class Test {
    public static void main(String[] args){
        CommandExecutor executor = new CommandExecutorProxy("Myedn", "wrong_pwd");
        try {
            executor.runCommand("ls -ltr");
            executor.runCommand(" rm -rf abc.pdf");
        } catch (Exception e) {
            System.out.println("Exception Message::"+e.getMessage());
        }

    }
}
```

Uygulamamızın UML Diagramı



Proxy Tasarım Deseninin Şematik Gösterimi


Both the Proxy and the RealSubject implement the Subject interface. This allows any client to treat the proxy just like the RealSubject.
<<interface>>
Subject
request()
RealSubject
request()
subject
Proxy
request()
The RealSubject is usually the object that does most of the real work; the Proxy controls access to it.
The Proxy often instantiates or handles the creation of the RealSubject.
The Proxy keeps a reference to the Subject, so it can forward requests to the Subject when necessary.