Tasarım Örüntüleri
Proxy

Örüntülerin Temel Prensipleri

- GoF tasarım örüntülerinin altında yatan temel prensipler
  - Encapsulation
  - Composition
  - Abstract Data Types

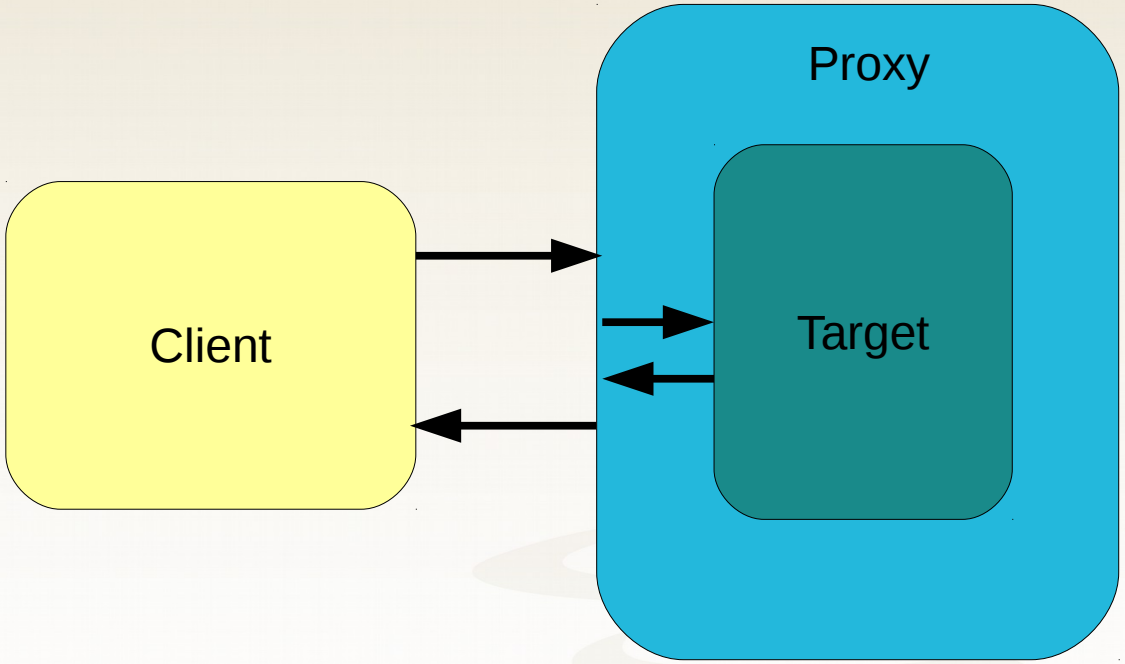
Proxy Örüntüsü

- Proxy
  - **Davranış encapsulation**'ı yapan diğer bir örüntü
  - **Aspect Oriented Programlama**'nın da temeli
  - Pek çok **altyapısal (middleware) problemin çözümü**nde yer alır
  - **Spring** Application Framework tarafından yoğun biçimde kullanılır

Proxy Örüntüsü

- Bazı durumlarda **nesnenin hemen yaratılması** mümkün olmayabilir
- Ya da yaratılmasını **ihtiyaç anına kadar ötelemek** daha verimli olabilir
- Nesnelerin belirli **metotlarını çalıştırmadan önce veya çalıştırdıktan sonra ilave işlemler**in yapılması gerekebilir

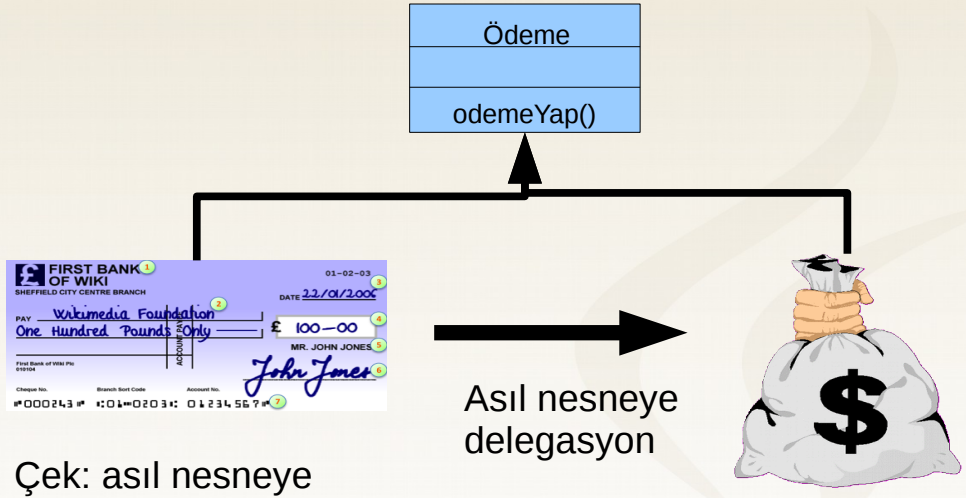
Proxy Örüntüsü



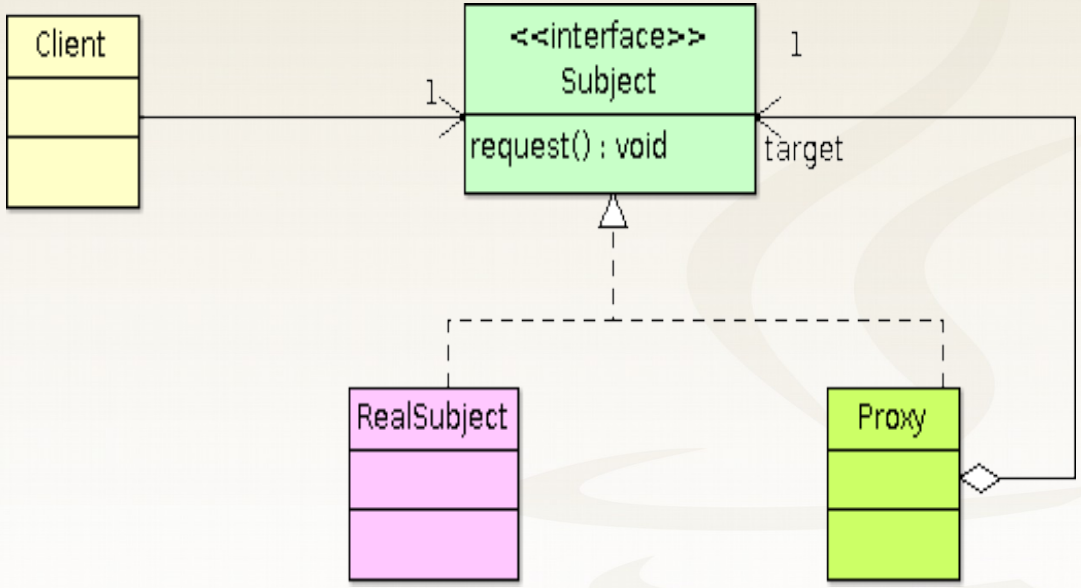
Proxy Türleri

- **Remote Proxy:** Farklı bir adres space'indeki nesnenin lokal representasyonunu oluşturur
- **Virtual Proxy:** Yaratılması maliyetli nesneyi gerçekten ihtiyaç duyulduğunda yaratır
- **Protection Proxy:** Asıl nesneye erişimi denetler
- **Smart Reference:** Asıl nesneye erişim sağlamanın yanı sıra ilave işlemlerde gerçekleştirir

Gerçek Dünyadan Bir Metafor



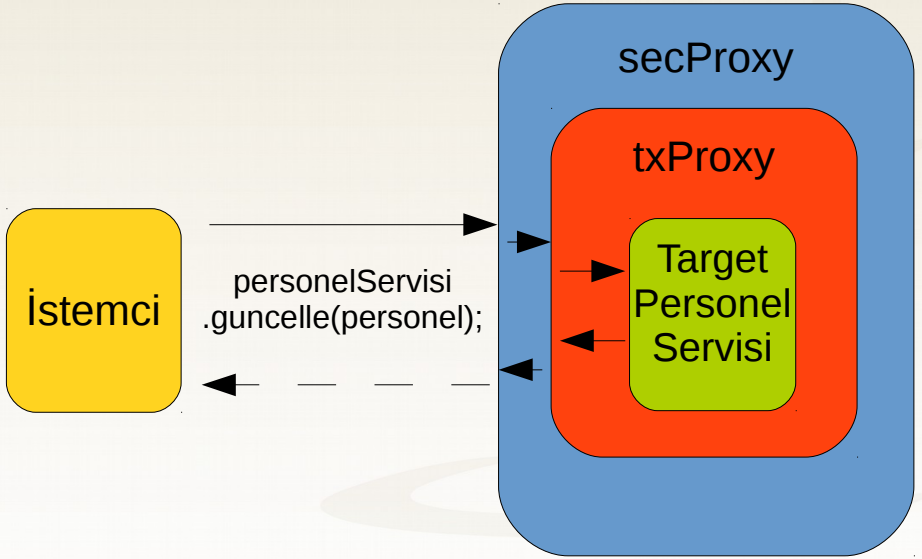
Proxy Sınıf Diagramı



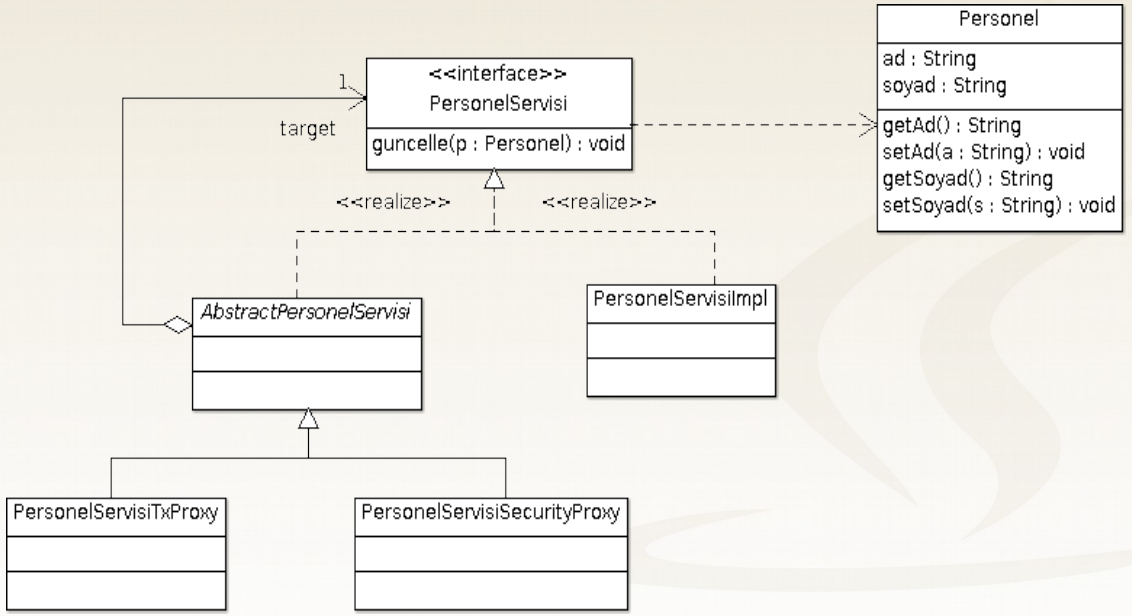
Örnek Problem

- Personel bilgilerini güncelleyen PersonelServisi isimli bir arayüz ve bunun bir gerçekleştirimi olsun
- Bu sınıf içerisinde personel güncellemesi yapılırken TX yönetiminin de yapılması istenmektedir
- Ayrıca personelin sadece kendi bilgilerini güncellemesi için güncelleme işlemi öncesinde bir yetki kontrolü yapılması da istenmektedir
- Transaction yönetimi ve yetkilendirme işlemleri, personel güncelleme davranışı üzerine sonradan konfigüratif ve uygulama geliştiricilerin isteğine bağılı biçimde eklenebilmelidir

PersonelServisi ve Proxy Zinciri



Örnek Problem Sınıf Diagramı



Örnek Problem

```
public interface PersonelServisi {
     public void guncelle(Personel personel);
}
```

```
public class PersonelServisiImpl implements PersonelServisi {

     @Override
     public void guncelle(Personel personel) {
          System.out.println(
               "Perform personel update here...");
     }

}
```

Örnek Problem

```
public abstract class AbstractPersonelServisi
                                implements PersonelServisi {
     protected PersonelServisi target;

     public AbstractPersonelServisi(PersonelServisi target) {
          super();
          this.target = target;
     }
}
```

Örnek Problem

```
public class PersonelServisiTransactionProxy
                    extends AbstractPersonelServisi {

     public PersonelServisiTransactionProxy(PersonelServisi target) {
          super(target);
     }

     @Override
     public void guncelle(Personel personel) {
          try {
               System.out.println("begin transaction here");
               target.guncelle(personel);
               System.out.println("commit transaction");
          } catch (Exception ex) {
               System.out.println("rollback transaction");
               throw ex;
          }
     }
}
```

Örnek Problem

```
public class PersonelServisiSecurityProxy
                    extends AbstractPersonelServisi {

     public PersonelServisiSecurityProxy(PersonelServisi target) {
          super(target);
     }

     @Override
     public void guncelle(Personel personel) {
          System.out.println("perform security check, "
                    + "then allow target method execution");
          target.guncelle(personel);
     }
}
```

Örnek Problem

```
public class Client {
     public static void main(String[] args) {

          PersonelServisi ps = new PersonelServisiSecurityProxy(
                                    new PersonelServisiTransactionProxy(
                                         new PersonelServisiImpl()));

          Personel personel = new Personel();

          ps.guncelle(personel);
     }
}
```

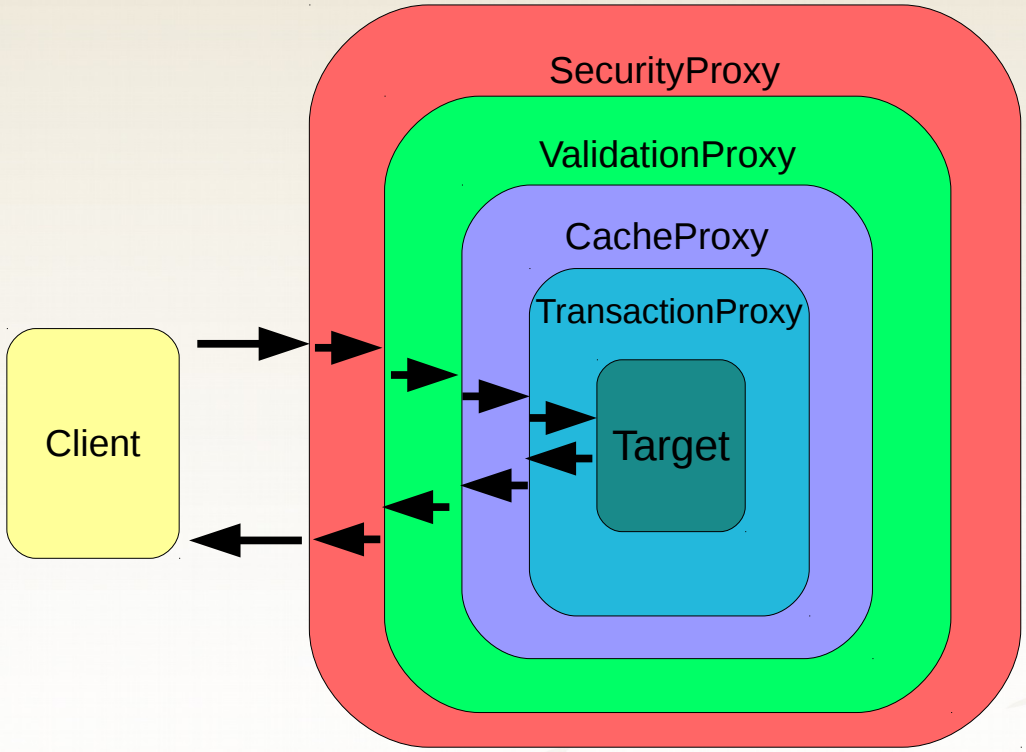
Spring İçerisinde Proxy Örüntüsünün Kullanım Yerleri

- Transaction yönetimi
- Bean scope kabiliyeti (request ve session scope bean'ler)
- Aspect oriented programlama altyapısı (Spring AOP)
- Metot düzeyinde validasyon ve caching
- Bean hot-swapping
- Remoting
- Spring security'de metot düzeyinde yetkilendirme

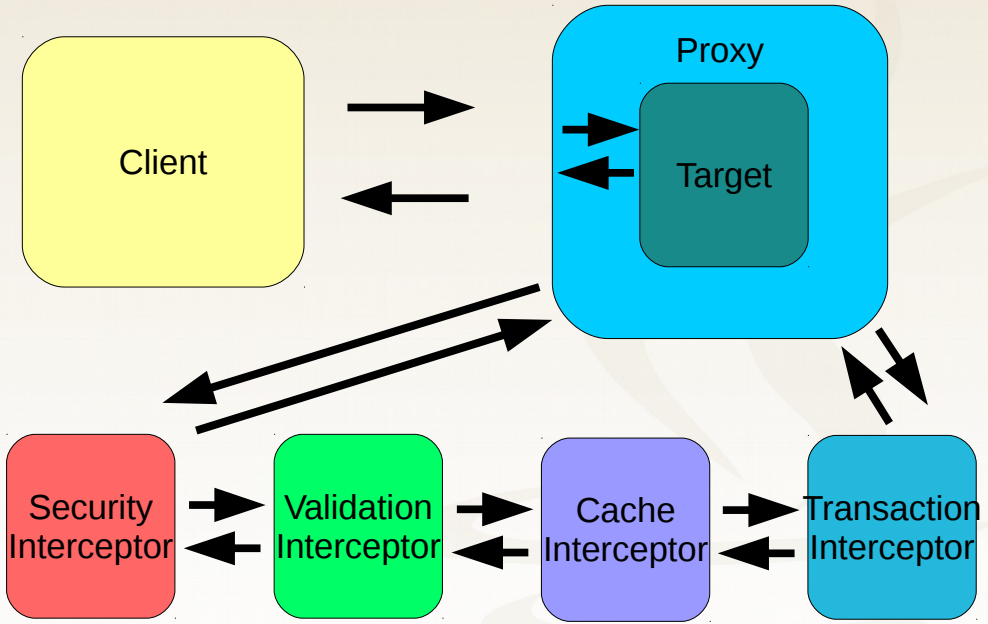
Dinamik Proxy Oluşturma Yöntemleri

- **Interface Proxy**
  - Proxy sınıf üretmek için hedef nesnenin sahip olduğu arayüzlerden birisi kullanılır
  - JDK proxy olarak da bilinir, JDK API'sinde mevcuttur
- **Class Proxy**
  - Proxy sınıf hedef nesnenin ait olduğu sınıf extend edilerek yaratılır
  - CGLIB proxy olarak da bilinir, CGLIB, Javassist gibi kütüphaneler kullanılarak gerçekleştirilir

Spring ve Proxy Nesne Zinciri



Spring Proxy ve MethodInterceptor Zinciri



ProxyFactoryBean

```
<bean id="personelServisiProxy"
          class="org.springframework.aop.framework.ProxyFactoryBean">
     <property name="interfaces">
          <array>
               <value>
                    com.javaegitimleri.PersonelServisi
               </value>
          </array>
     </property>
     <property name="target" ref="targetPersonelServisi"/>
     <property name="interceptorNames">
          <value>securityInterceptor,transactionInterceptor</value>
     </property>
</bean>

<bean id="targetPersonelServisi"
                    class="com.javaegitimleri.PersonelServisiImpl"/>

<bean id="securityInterceptor"
                    class="com.javaegitimleri.SecurityInterceptor"/>

<bean id="transactionInterceptor"
                    class="com.javaegitimleri.TransactionInterceptor"/>
```

MethodInterceptor

```
public class LoggingInterceptor implements MethodInterceptor {

     @Override
     public Object invoke(MethodInvocation invocation) throws
Throwable {
          try {
               System.out.println("method entered");
               Object result = invocation.proceed();
               System.out.println("method executed successfully,
returning result :" + result);
               return result;
          } catch (Throwable t) {
               System.out.println("exception occurred :" + t);
               throw t;
          } finally {
               System.out.println("method exited");
          }
     }
}
```

Proxy Örüntüsünün Sonuçları

- İlave kabiliyetlerin veya her durumda işletilmesi uygun olmayan **davranışların tek bir sınıf içerisinde birikmesi**nin önüne geçilir
- İlave davranışlar **farklı tipte nesneler üzerinde** uygulanabilir
- Böylece bu **davranışların yeniden kullanılabilirliği** mümkün hale gelir

İletişim

JAVA Eğitimleri

Kurumsal Java Eğitimleri

www.java-egitimleri.com

info@java-egitimleri.com

@javaegitimleri

youtube.com/c/
KurumsalJavaEğitimleri