

Spring Application Framework Overview 3



Proxy Örüntüsü

- Bazı durumlarda **nesnelerin hemen yaratılması maliyetli** olabilir, yada o anda yaratılmaları **uygun olmayabilir**
- Ya da bazı nesnelere erişmeden evvel veya erişimden sonra **ilave bazı işlemlerin** yapılması gerekebilir

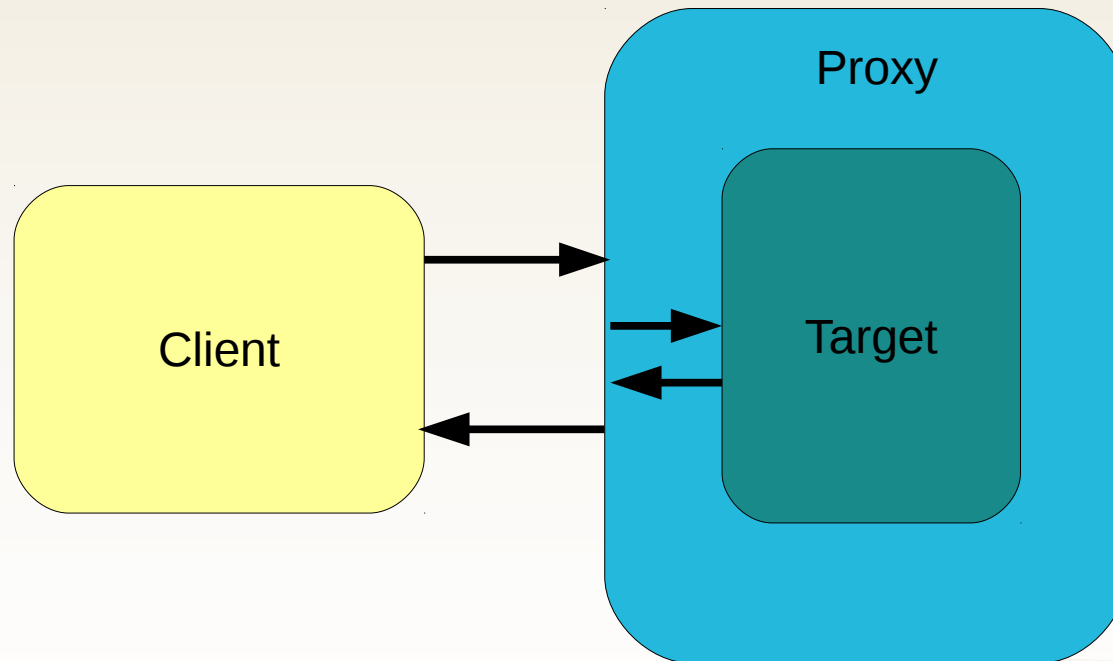
Proxy Örüntüsü

- Asıl nesnenin yaratılmasını ihtiyaç anına kadar erteleyen, asıl nesneden önce veya sonra devreye giren, **asıl nesne yerine kullanılabilen bir nesne** yaratılır
- Vekil nesne asıl nesneye **erişimi dolaylı** hale getirir

Proxy Örüntüsü

Proxy, target nesne ile aynı tipte olup, client ile target nesnenin arasına girer

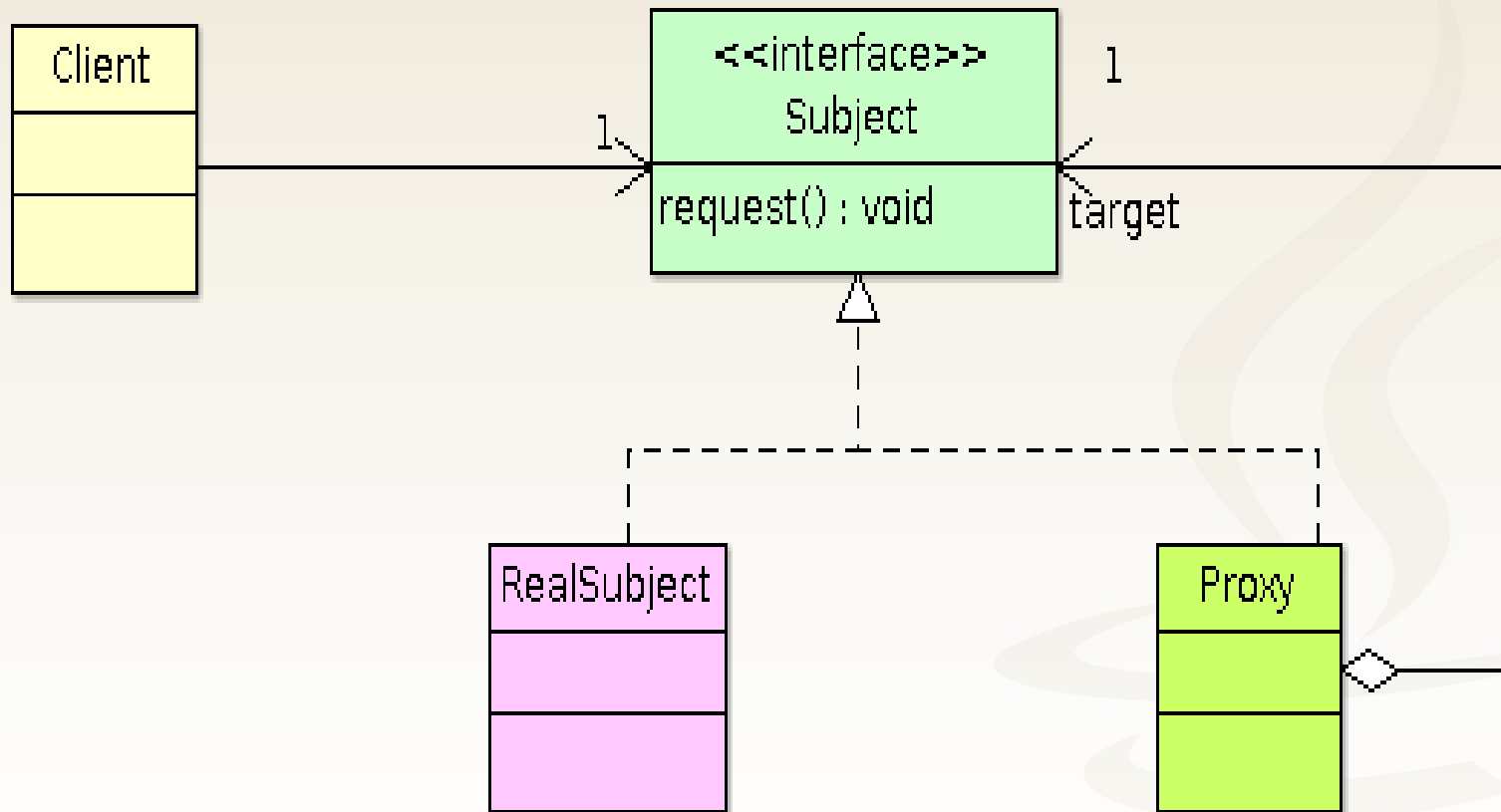
Client proxy nesne ile konuştuğunun farkında değildir



Client'ın target nesne üzerindeki metot çağrıları öncelikle proxy nesneye erişir

Proxy metot çağrısından önce veya sonra bir takım işlemler gerçekleştirebilir

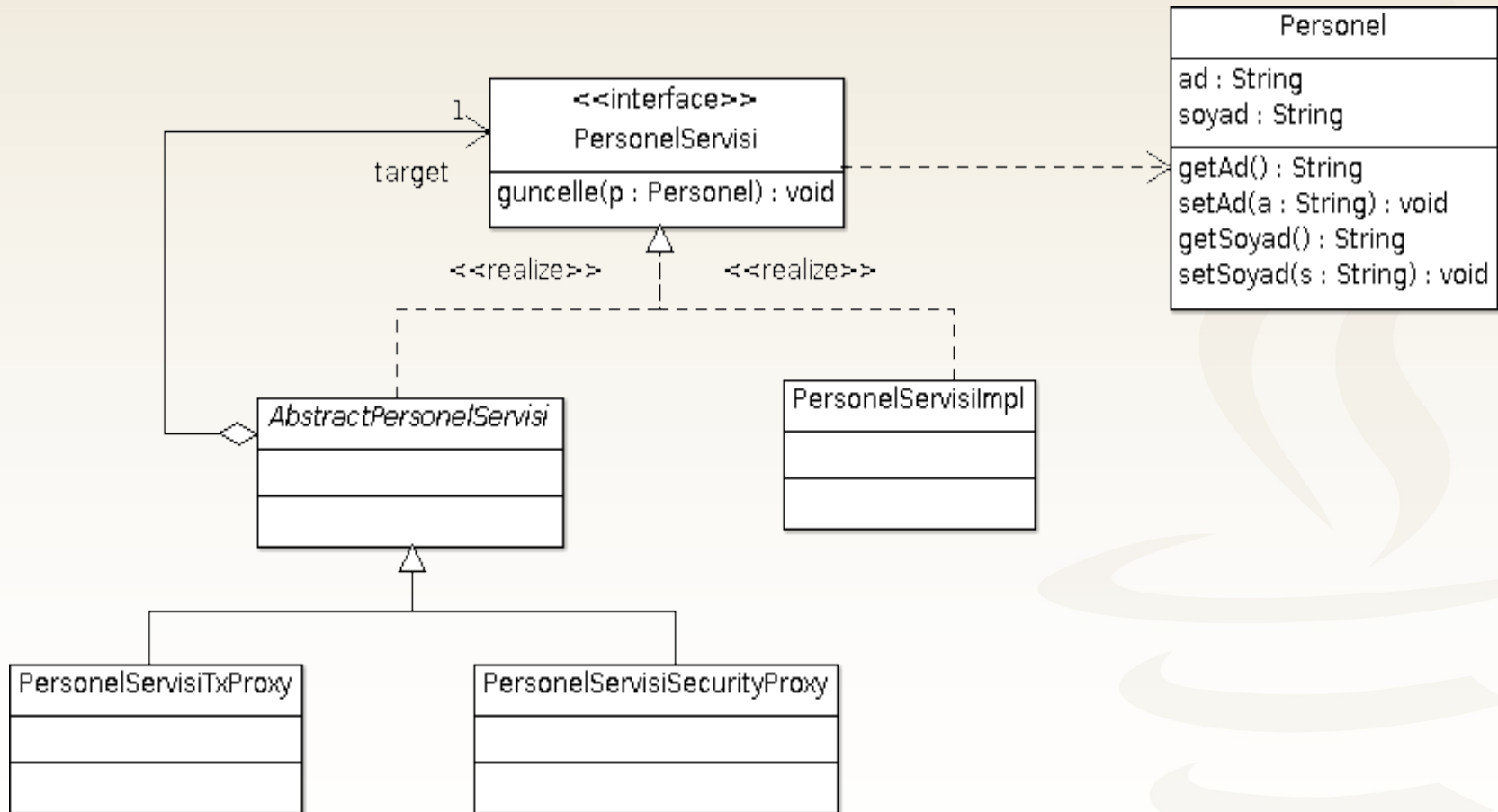
Proxy Sınıf Diagramı



Örnek Problem: Proxy

- Personel bilgilerini güncelleyen PersonelServisi isimli bir sınıf vardır
- Bu sınıf içerisinde personel güncellemesi yapılırken TX yönetiminin de yapılması istenmektedir
- Ayrıca personelin sadece kendi bilgilerini güncellemesi için de yetki kontrolü yapılmalıdır
- Transaction yönetimi ve yetkilendirme işlemlerinin istemci kodu tarafından bilinmesi istenmemektedir
- Bu davranışlar personel güncelleme davranışı üzerine sonradan konfigüratif ve uygulama geliştiricilerin isteğine bağlı biçimde eklenebilmelidir

Örnek Problem: Proxy



PersonelServisiTxProxy

```
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;
        }
    }
}
```


PersonelServisiSecProxy

```
public class PersonelServisiSecurityProxy
    extends AbstractPersonelServisi {

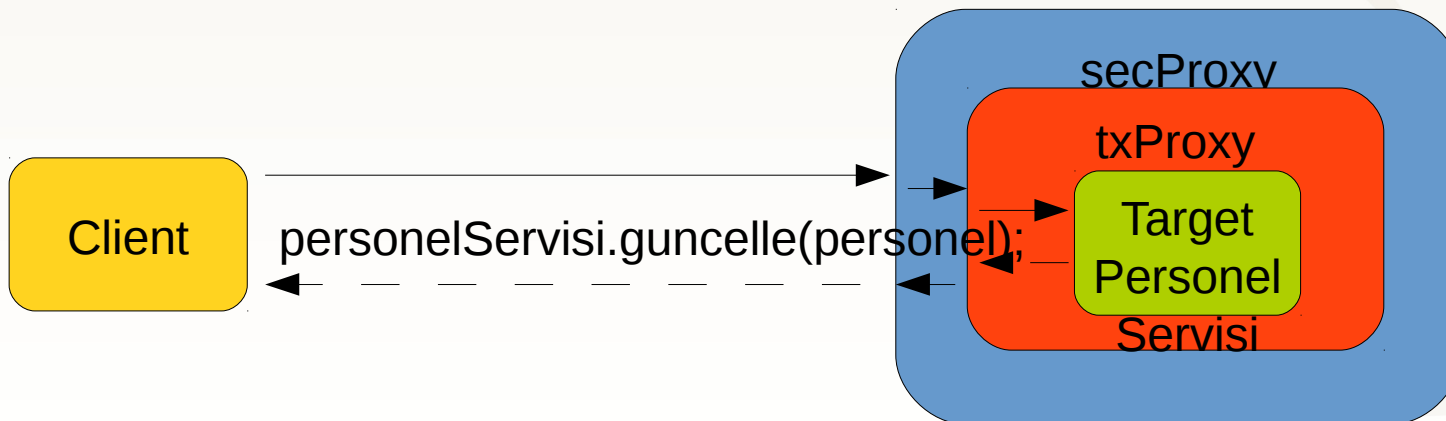
    public PersonelServisiSecurityProxy(PersonelServisi target) {
        super(target);
    }

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

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

Client'in Proxy Nesneler ile Etkileşimi

```
public class Client {  
    public static void main(String[] args) {  
  
        PersonelServisi ps = new PersonelServisiSecurityProxy(  
                                new PersonelServisiTransactionProxy(  
                                    new PersonelServisiImpl()));  
  
        Personel personel = new Personel();  
  
        ps.guncelle(personel);  
    }  
}
```



Dinamik Proxy Oluşturma Yöntemleri

- Spring, Hibernate gibi framework'ler proxy nesneler oluşturmak için **dinamik proxy sınıfları** üretirler
- Dinamik proxy sınıfı üretmek için **iki yol** vardır
 - Interface proxy
 - Class proxy

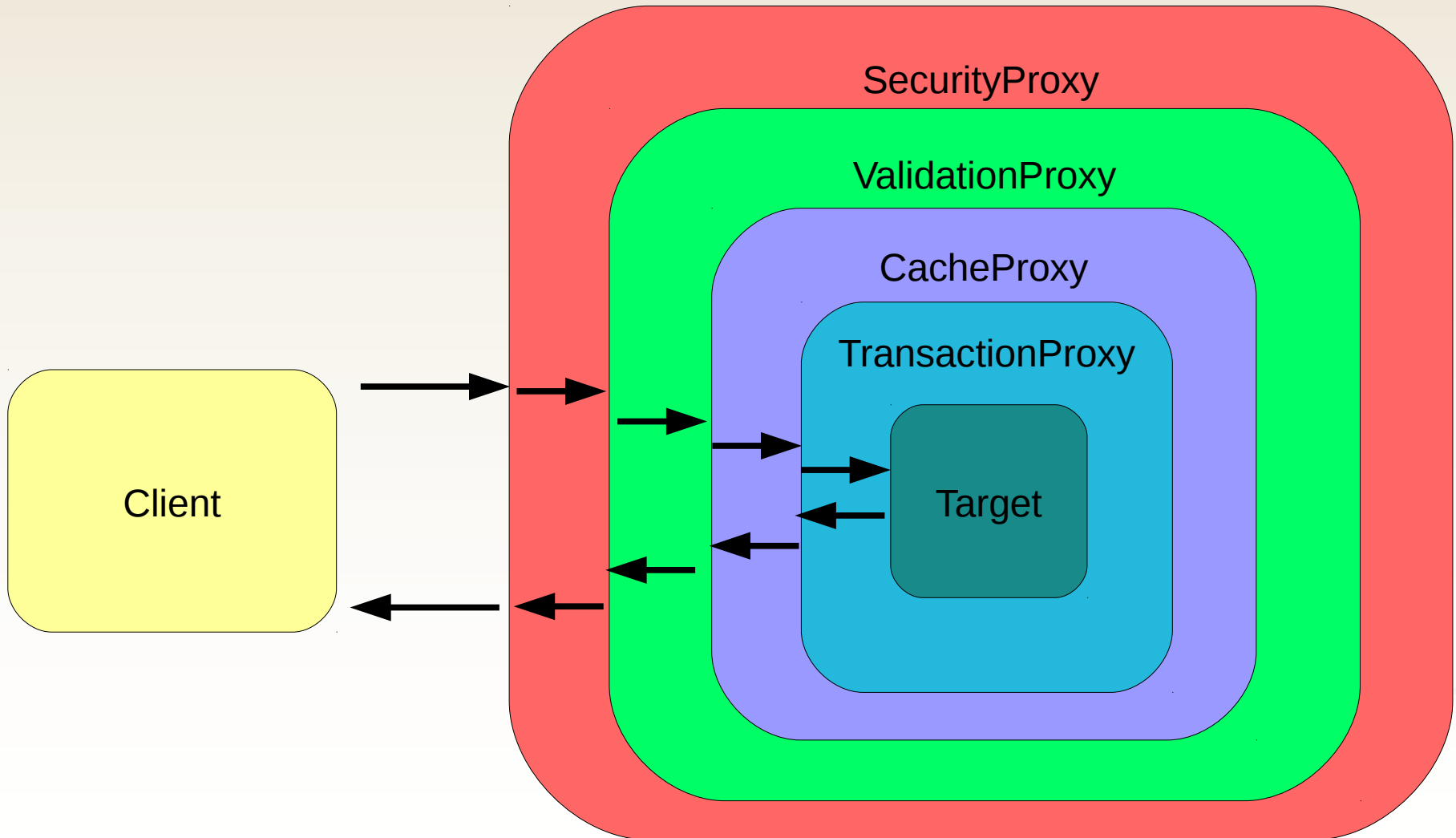
Spring İçerisinde Proxy Örüntüsünün Kullanımı

- Spring bu kabiliyetleri hayata geçirmek için genellikle uygulama geliştiricilerden habersiz **otomatik olarak proxy oluşturma** işini gerçekleştirir
- Diğer bean'lere de **bağımlılık olarak proxy nesne enjekte** edilir
- Diğer bean'ler proxy ile çalıştıklarının farkında **değillerdir**

Spring İçerisinde Proxy Örüntüsünün Kullanımı

- Spring Application Framework'ün **pek çok kabiliyeti** proxy örüntüsü üzerine kuruludur
 - Transaction yönetimi
 - Bean scope kabiliyeti (request ve session scope bean'ler)
 - Aspect oriented programlama altyapısı (Spring AOP)
 - Metot düzeyinde validasyon ve caching
 - Remoting
 - Spring security'de metot düzeyinde yetkilendirme

Spring ve İç İçe Proxy Nesne Zinciri



Spring ile Veri Erişimi'nin Özellikleri

- Spring değişik teknolojiler için kullanımı **kolay ve standart bir veri erişim** desteği sağlar
 - JDBC, JPA, Hibernate vb desteklenir
- Farklı veri erişim teknolojileri aynı anda kullanılabilir
- **Kapsamlı ve transparan** bir transaction yönetim altyapısına sahiptir
- Veri erişim teknolojilerinin exception hiyerarşilerini **standart bir exception hiyerarşisine** çevirir

JDBC API ile Veri Erişimi

- JDBC API kullanarak gerçekleştirilen veri tabanı işlemlerinde yazılan kod blokları genel olarak birbirlerine **benzer bir akışa** sahiptir
- Hepsinde veritabanı bağlantısı oluşturma, SQL'i çalıştırma, dönen sonuçlar üzerinde işlem yapma, TX varsa commit/rollback yapma, hataları ele alma ve bağlantıyı kapatma gibi **işlemler standarttır**
- **Değişen kısımlar** SQL, parametreler ve dönen sonucu işleyen kod bloğu olur

JDBC API ile Veri Erişimi

- Veritabanı bağlantı parametrelerinin belirtilmesi ve bağlantının kurulması
- *SQL sorgusunun oluşturulması*
- Oluşturulan Statement'ın derlenip çalıştırılması
- Dönen ResultSet üzerinde işlem yapan bir döngünün kurulması
- *Bu döngü içerisinde her bir kaydın işlenmesi*
- Meydana gelebilecek hataların ele alınması
- Transaction'ın sonlandırılması ve veritabanı bağlantısının kapatılması

Sadece kırmızı font ile işaretlenen kısımlar değişkenlik gösterir, diğer adımlar bütün persistence işlemlerinde standarttır

JDBC API ile Veri Erişimine Örnek

```
public Collection<Document> findDocuments() {  
    Connection c = null;  
    Statement stmt = null;  
    try {  
        c = DriverManager.getConnection(  
            "jdbc:h2:tcp://localhost/~test","sa", "");  
        c.setAutoCommit(false);  
        stmt = c.createStatement();  
        ResultSet rs = stmt  
            .executeQuery("select * from T_DOCUMENT");  
        Collection<Document> result = new ArrayList<Document>();  
        while (rs.next()) {  
            Document doc = new Document();  
            doc.setName(rs.getString("doc_name"));  
            doc.setType(rs.getInt("doc_type"));  
            result.add(doc);  
        }  
    }  
}
```

JDBC API ile Veri Erişimine Örnek

```
c.commit();  
return result;  
} catch (SQLException ex) {  
    try {  
        c.rollback();  
    } catch (Exception e) {}  
    throw new DataRetrievalFailureException(  
        "Cannot execute query", ex);  
} finally {  
    try {  
        stmt.close();  
    } catch (Exception e) {}  
    try {  
        c.close();  
    } catch (Exception e) {}  
}
```

Spring Üzerinden JDBC ile Veri Erişimi

- Spring veri erişiminde bu tekrarlayan kısımları ortadan kaldırmak için **Template Method örüntüsü tabanlı** bir kabiliyet sunar
- **JdbcTemplate** merkez sınıftır
- Utility veya helper sınıflarına benzetilebilir
- JdbcTemplate sayesinde Template Method tarafından dikte edilen **standart bir kullanım şekli** kod geneline hakim olur

JdbcTemplate ile Veri Erişimi

```
public Collection<Document> findDocuments() {  
    Collection<Document> result = jdbcTemplate.query(  
        "select * from T_DOCUMENT",  
        new RowMapper<Document>() {  
  
        @Override  
        public Document mapRow(ResultSet rs, int rowNum)  
            throws SQLException {  
            Document doc = new Document();  
            doc.setName(rs.getString("doc_name"));  
            doc.setType(rs.getInt("doc_type"));  
            return doc;  
        }  
    }  
);  
return result;  
}
```

sorgu

callback

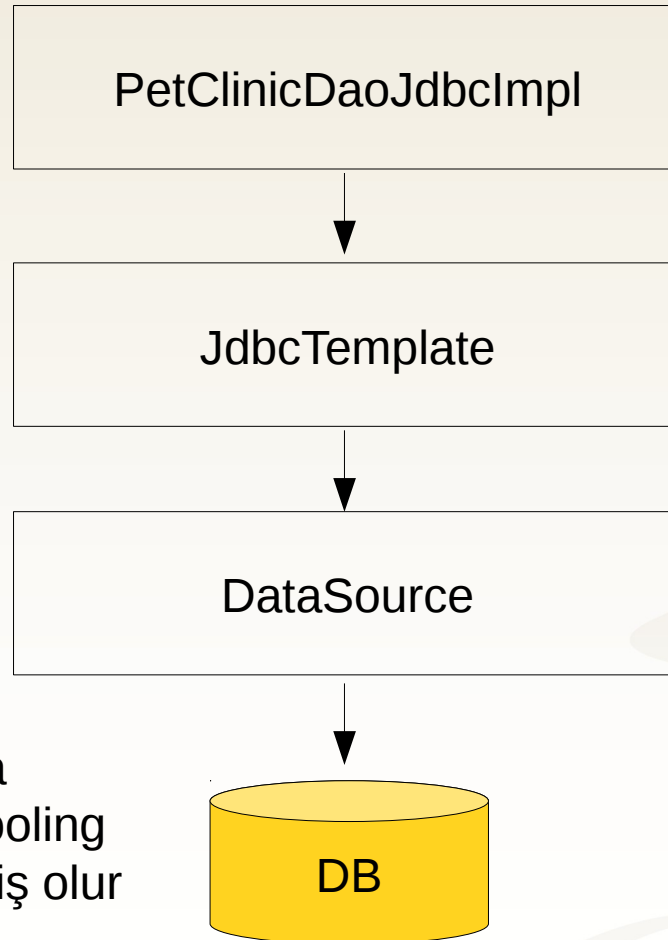
JdbcTemplate Konfigürasyonu

Uygulama tarafında
JdbcTemplate ile
çalışırken sadece
Callback yazmak yeterlidir

Spring veritabanı
bağlantılarını **DataSource**
nesnesinden alır

DataSource
connection factory'dir

DataSource'un kendi başına
yönetilmesi ile connection pooling
vs. uygulamadan izole edilmiş olur



JDBC ile veri erişimi
JdbcTemplate üzerine
kurulmuştur

Çalışması için
DataSource nesnesine
ihtiyaç vardır

Thread safe'dir, birden
fazla bean tarafından
erişilebilir

DataSource Tanım Örneği

```
<beans...>
```

```
    <bean id="dataSource"  
class=  
"org.springframework.jdbc.datasource.DriverManagerDataSource">
```

```
    <property name="driverClassName"  
value="org.h2.Driver"/>
```

```
    <property name="url"  
value="jdbc:h2:tcp://localhost/~/test"/>
```

```
    <property name="username"  
value="sa"/>
```

```
    <property name="password"  
value=""/>
```

```
    </bean>
```

```
</beans>
```

DataSource Tanım Örneği

```
<beans...>
```

```
<jee:jndi-lookup id="dataSource"  
    jndi-name= "java:comp/env/jdbc/DS" />
```

```
</beans>
```


JdbcTemplate Kullanım Örnekleri

```
Collection<Document> result = jdbcTemplate.query(  
    "select * from T_DOCUMENT",  
    new RowMapper<Document>() {  
  
        @Override  
        public Document mapRow(ResultSet rs, int  
rowNum)  
            throws SQLException {  
            Document doc = new Document();  
            doc.setName(rs.getString("doc_name"));  
            doc.setType(rs.getInt("doc_type"));  
            return doc;  
        }  
    }  
);
```

JdbcTemplate Kullanım Örnekleri

```
String result = jdbcTemplate.queryForObject("select  
last_name from persons where id = ?", new Object[]{1212L},  
String.class);
```

```
List<String> result = jdbcTemplate.queryForList("select  
last_name from persons", String.class);
```

```
Map<String, Object> result =  
jdbcTemplate.queryForMap("select last_name, first_name from  
persons where id = ?", 1212L);
```

```
List<Map> result = jdbcTemplate.queryForList("select * from  
persons");
```

JdbcTemplate Kullanım Örnekleri

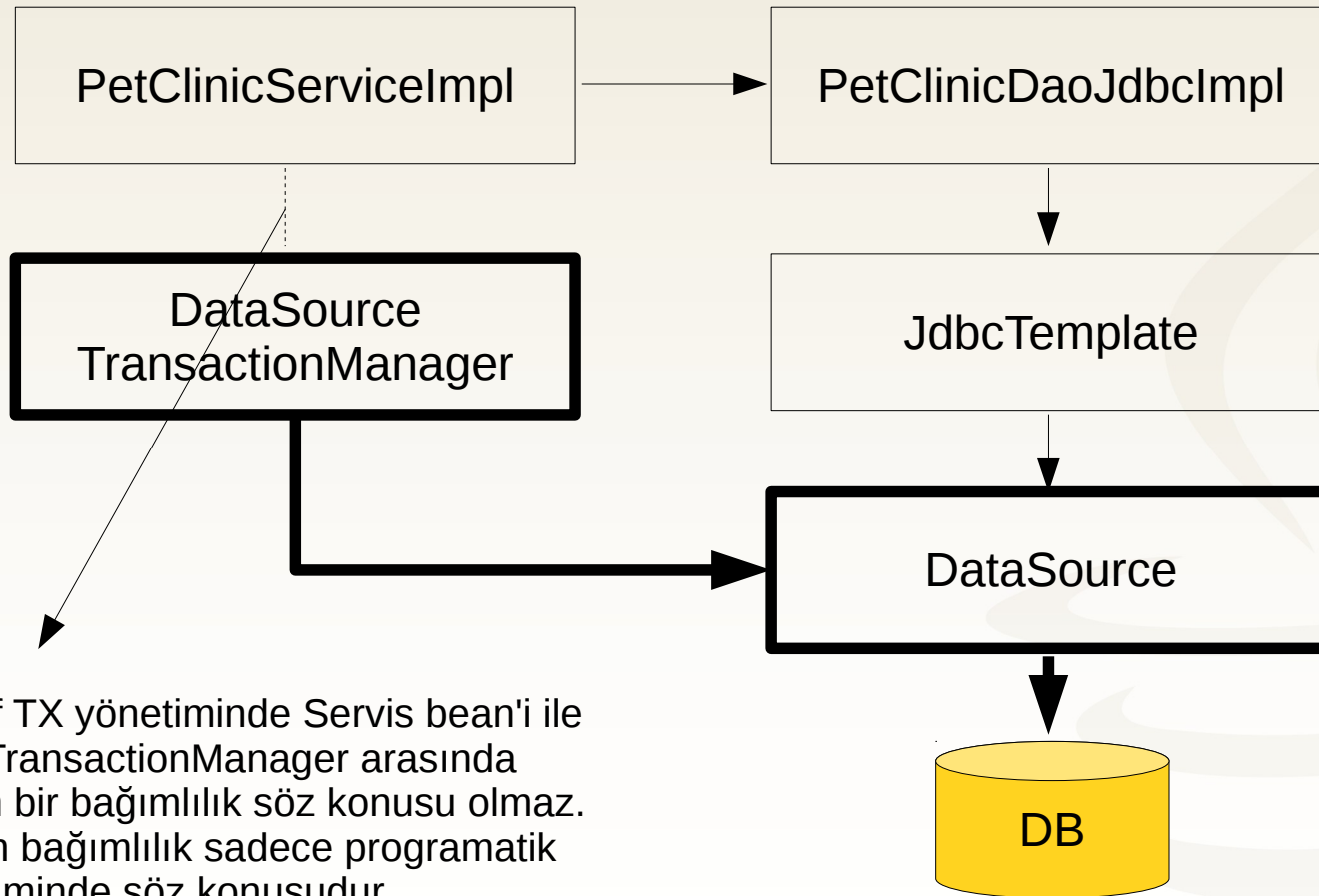
```
int insertCount = jdbcTemplate.update(  
    "insert into persons (first_name, last_name) values (?, ?)",  
    "Ali", "Yücel");
```

```
int updateCount = jdbcTemplate.update(  
    "update persons set last_name = ? where id = ?", "Güçlü", 1L);
```

```
int deleteCount = jdbcTemplate.update(  
    "delete from persons where id = ?", 1L);
```

```
int result = jdbcTemplate.update(  
    "call SUPPORT.REFRESH_PERSON_SUMMARY(?)", 1L);
```

PlatformTransactionManager Konfigürasyonu - JDBC



Dekleratif TX yönetiminde Servis bean'i ile PlatformTransactionManager arasında doğrudan bir bağımlılık söz konusu olmaz. Doğrudan bağımlılık sadece programatik TX yönetiminde söz konusudur.

PlatformTransactionManager Konfigürasyonu - JDBC

```
<bean id="transactionManager"  
class="org.springframework.jdbc.datasource.DataSourceTr  
ansactionManager">  
    <property name="dataSource" ref="dataSource" />  
</bean>
```

```
<bean id="dataSource"  
class="org.springframework.jdbc.datasource.DriverManage  
rDataSource">  
    <property name="driverClassName" value="$  
{jdbc.driverClassName}" />  
    <property name="url" value="{jdbc.url}" />  
    <property name="username" value="{jdbc.username}" />  
    <property name="password" value="{jdbc.password}" />  
</bean>
```

Dekleratif Transaction Yönetimi

- Uygulama içinde TX yönetimi ile ilgili **kod yazılmaz**
- Servis metot çağrısı geldiği vakit Spring Container tarafından **yeni bir TX başlatılır**
- Metot başarılı sonlandığı vakit **TX commit** edilir
- **Sınıf veya metot düzeyinde** TX yönetimi yapılabilir
- En sık **tercih edilen yöntemdir**

Dekleratif Transaction Yönetimi ve Rollback

- Transactional bir metot içerisinde bir exception meydana geldiğinde **exception türüne** bakılır
- Default olarak runtime exception'larda **TX rollback** edilir
- Checked exception'larda ise **TX commit** edilir
- Ancak çoğunlukla **bu davranış değiştirilir**

Dekleratif Transaction Yönetimi

- Dekleratif TX yönetimi **@Transactional** anotasyonu ile yapılır
- **Sınıf veya metot düzeyinde** kullanılabilir
- Sadece **public metotlarda** kullanılmalıdır
- **<tx:annotation-driven/>** elemanı **@Transactional** anotasyonlarını devreye sokar

@Transactional ile Dekleratif TX Yönetimi

```
@Transactional
public class DefaultFooService implements FooService {

    public Foo getFoo(String fooName) {
        // ...
    }

    public void updateFoo(Foo foo) {
        // ...
    }
}
```

@Transactional ile Dekleratif TX Yönetimi

```
public class DefaultFooService implements FooService {  
  
    public Foo getFoo(String fooName) {  
        // ...  
    }  
  
    @Transactional  
    public void updateFoo(Foo foo) {  
        // ...  
    }  
}
```

@Transactional ile Dekleratif TX Yönetimi

```
<bean id="fooService"  
class="x.y.service.DefaultFooService"/>
```

```
<tx:annotation-driven/>
```

```
<bean id="transactionManager"  
class="org.springframework.jdbc.datasource.DataSource  
ceTransactionManager">
```

```
    <property name="dataSource" ref="dataSource"/>
```

```
</bean>
```

@Transactional Default Değerleri

- **@Transactional** anotasyonundaki attribute'ların default değerleri:
 - **Propagation** REQUIRED
 - **Isolation** DEFAULT
 - **Transaction** read/write
 - **Timeout** sistem default
 - **Rollback** Herhangi bir RuntimeException

@Transactional Default Değerleri

```
@Transactional(rollbackFor = Exception.class)
public class DefaultFooService implements FooService {

    @Transactional(readOnly = true)
    public Foo getFoo(String fooName) {
        // ...
    }

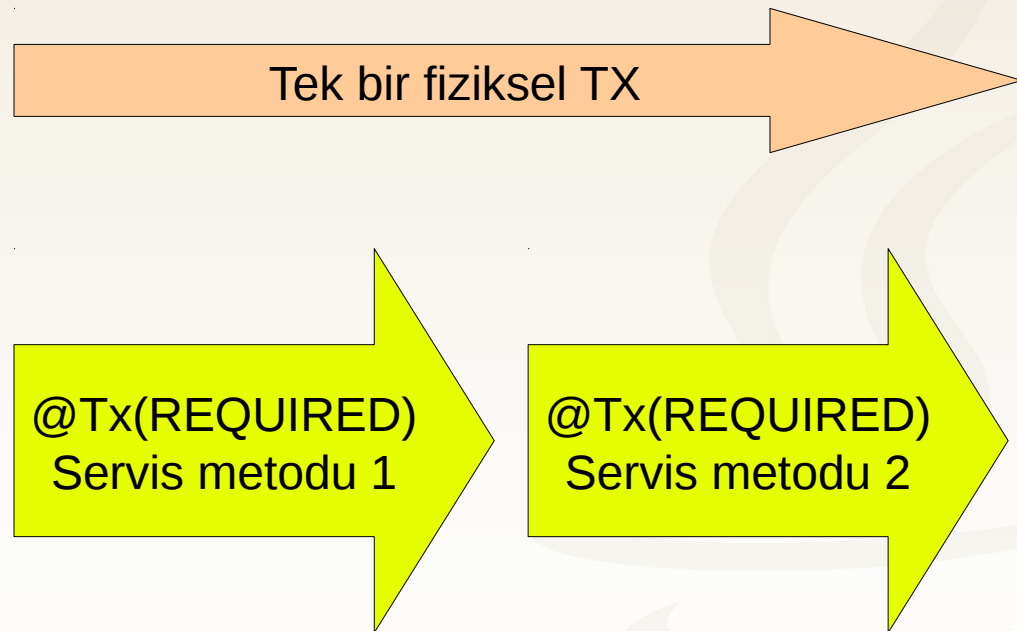
    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void updateFoo(Foo foo) {
        // ...
    }
}
```

Transaction Propagation: PROPAGATION_REQUIRED

Her bir servis metodu için
ayrı mantıksal TX'ler yaratılır

İçteki TX **setRollbackOnly**
yaparsa bütün **diğer TX'ler**
etkilenir

Dıştaki commit yapsa bile
UnexpectedRollback
Exception fırlatılır

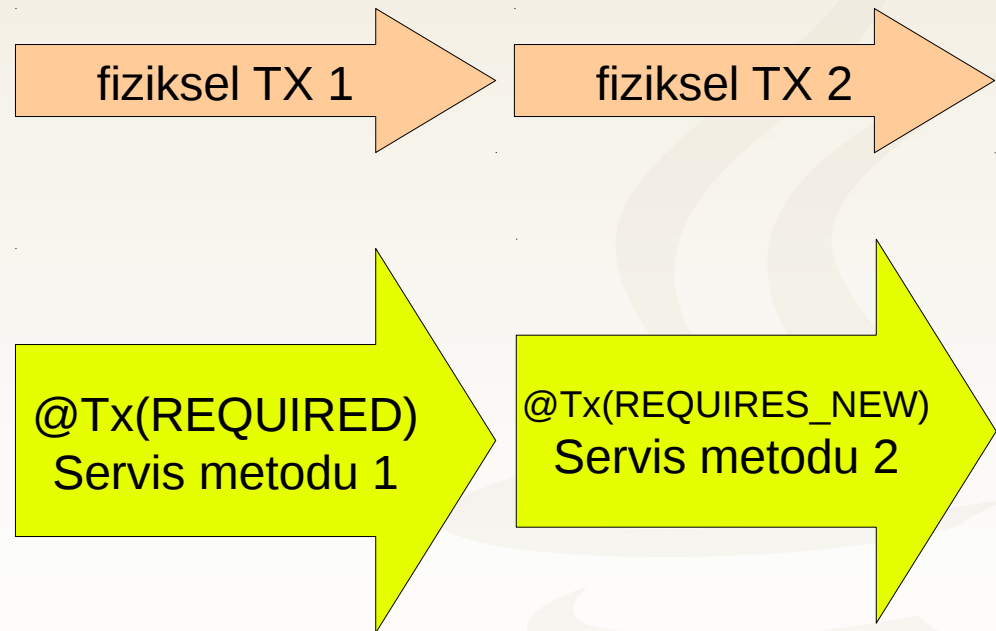


Transaction Propagation: PROPAGATION_REQUIRES_NEW

Birbirinden **bağımsız fiziksel TX'ler** vardır

İkinci servis metodu çalışırken
İlk servis metodunun TX'i
Suspend edilir

Herbirisi kendi başına
commit/rollback yapılabilir



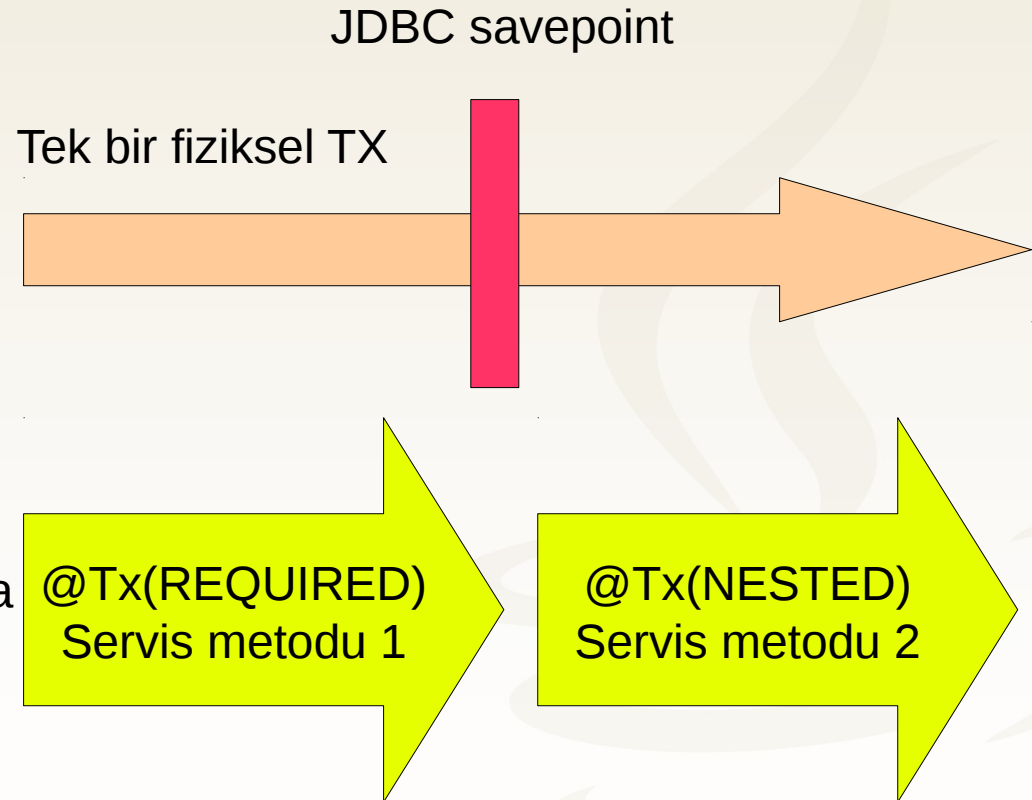
Transaction Propagation: PROPAGATION_NESTED

Birden fazla savepoint vardır

inner TX kendi içinde **rollback**
Yapabilir

Sadece Savepoint'e kadar yapılan
işlemler rollback olur

Sadece **JDBC**'de yani
DataSourceTransactionManager'da
anlamlıdır, Hibernate desteklemez



Spring TestContext Framework Konfigürasyonu

ApplicationContext yüklenmesi
Dependency injection
Transactional test desteği aktive olur

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class FooTests {
    // ...
}
```


Default: classpath:/com/example/FooTests-context.xml
locations attribute ile farklı dosyalar belirtilebilir

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("/appContext.xml")
public class BarTests {
    // ...
}
```

Entegrasyon Testleri ve ApplicationContext Yönetimi

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("/base-context.xml")
public class BaseTest {
    // ...
}
```

```
@ContextConfiguration("/extended-context.xml")
public class ExtendedTest extends BaseTest {
    // ...
}
```



ExtendedTest sınıfı BaseTest sınıfından türediği için bu sınıftaki test metotları için yaratılacak olan ApplicationContext base-context.xml ve extended-context.xml dosyaları yüklenerek oluşturulacaktır

Entegrasyon Testleri ve Dependency Injection

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class SpringTests {

    @Autowired
    private FooService fooService;

    ...
}
```

ApplicationContext Yönetimi ve ÖnBellekleme

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("/appcontext/beans-*.xml")
public class SpringTests {
```

```
    @Autowired
    private ApplicationContext applicationContext;
```

```
    @Test
    @DirtiesContext
    public void testMethod1() {
    }
```

Bu test metodu çalıştıktan sonra ApplicationContext bir sonraki test metodu çalıştırılmadan önce yeniden yaratılacaktır

```
    @Test
    public void testMethod2() {
    }
```

```
}
```

Testleri Ortama Göre Çalıştırmak

- **@IfProfileValue** anotasyonu ile belirli bir ortam veya sistem değişkeninin değerine göre testler enable/disable edilebilir
- Sınıf veya metot düzeyinde kullanılabilir

```
@IfProfileValue(name="targetPlatform", value="test")  
@Test  
public void testMethod() {  
    // ...  
}
```

Test metodu eğer ifade true olarak evaluate ediyor ise çalıştırılır. Sınıf düzeyinde kullanılırsa o sınıftaki hiçbir test metodu çalıştırılmayacaktır

Entegrasyon Testleri ve Transaction Yönetimi

```
@ContextConfiguration
@Transactional
public class TransactionalTests {
    @Test
    public void testWithRollback() {
        // ...
    }

    @Rollback(false)
    @Test
    public void testWithoutRollback() {
        // ...
    }
}
```

İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- info@java-egitimleri.com

