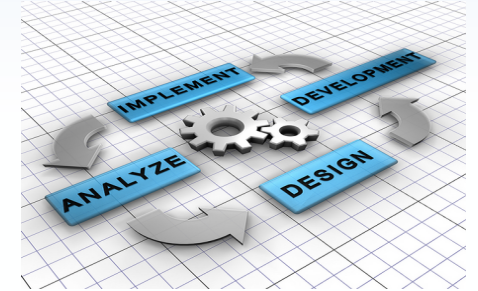
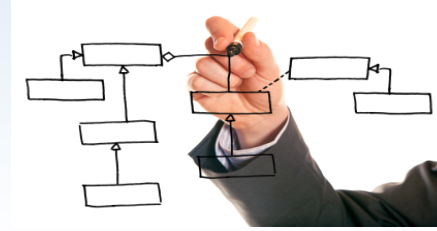


Dinamik Proxy Tabanlı View Model API

Kenan Sevindik Kimdir?

- 15 yıllık kurumsal uygulama geliştirme deneyimi var
- Çeşitli projelerin mimarilerinin oluşturulmasında görev aldı
- Spring, Spring Security, Hibernate, Vaadin gibi kurumsal Java teknolojilerinde kapsamlı bilgi birikimi ve deneyime sahip



Kenan Sevindik Kimdir?

- **Beginning Spring** kitabının yazarlarından
- 2011 yılında **Harezmi Bilişim Çözümleri** kurdu
- Harezmi neler yapıyor?
 - Kurumsal uygulama geliştirme faaliyetleri yürütüyor
 - Danışmanlık ve koçluk hizmetleri sunuyor
 - Kurumsal Java Eğitimleri adı altında eğitimler düzenliyor



Problem

Persistent domain nesnelerinin
doğrudan UI'a bind edilmesi
veya
UI katmanında kullanılması
sağlıklı sonuçlar vermez!



Senaryo 1

- **Owner** ve sahip olduğu **Pet**'lerin görüntülendiği ve yönetildiği klasik bir **master-detay** ekranı olsun

Owner List View

<input type="checkbox"/>	First Name	Last Name	E-Mail
<input type="checkbox"/>	Ali	Güç	ali@example.com
<input checked="" type="checkbox"/>	Veli	Doğru	veli@test.com
<input type="checkbox"/>	Cengiz	Çetin	cengiz@gmail.com
<input type="checkbox"/>	Ayşe	Us	ayse@yahoo.com

Add OwnerRemove OwnersEdit Owner

Owner Detail Tab View

Owner Detail

Owner Pets

First Name

Veli

Last Name

Doğru

E-Mail

veli@test.com.tr

Save Changes

Cancel

Kullanıcı bir grup Owner'ı listeler, listeden bir Owner kaydını seçerek detay ekranına geçer

Seçilen Owner kaydının bir takım alanları üzerinde değişiklikler yapar

Senaryo 1

Owner Pets Tab View

Owner Detail

Owner Pets

<input type="checkbox"/>	Name	Birth Date
<input type="checkbox"/>	Karabaş	01.01.2010
<input checked="" type="checkbox"/>	Cangöz	10.12.2015

Add Pet

Remove Pets

Edit Pet

Edit Pet Dialog

Edit Pet

Name

Cingöz

Birth Date

10.12.2015

Save Changes

Cancel

Daha sonra Owner Detail tab'ından Owner Pets tabına geçerek burada bir Pet kaydını güncellemeye başlar

Pet kaydı üzerinde değişiklikler yaptıktan sonra "Save Changes" butonuna tıkladığında hem Pet hem de Owner nesnesindeki değişiklikler DB'ye yansıtılacaktır

Senaryo 1: Değerlendirme

- Hem Owner hem Pet üzerinde yapılan **değişiklikler DB'ye topluca yansıtılmış** oldu
- Kullanıcının Pet üzerindeki değişiklikleri iptal edip **sadece Owner'da yaptığı değişikliği kaydetme** imkanı yoktu
- Ya da Owner üzerindeki değişikliği iptal edip **sadece Pet üzerindeki değişikliği kaydetme** şansı olmadı

Senaryo 1: Değerlendirme

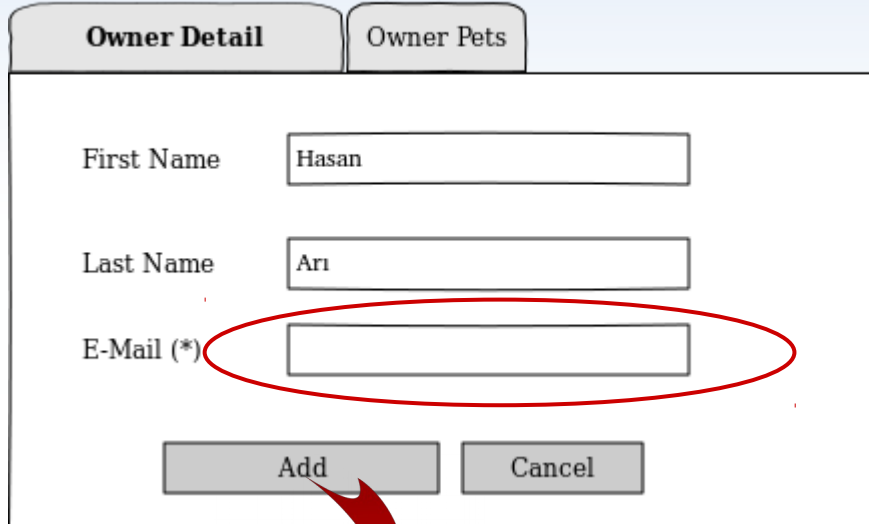
- Değişiklikleri geri alabilmek için property değerlerinin bir yerde saklanıp kullanıcı işleminden vazgeçtiğinde **eski değere dönmeye imkan sağlanması** gerekir

Senaryo 1 Türevleri

- Bu senaryoya benzer durum Owner'ın pets collection'ına **istenmeyen yeni bir Pet kaydının eklenmesi**
- ya da **mevcut Pet kaydının yanlışlıkla silinmesi** şeklinde de karşımıza çıkabilir
- Kullanıcının Owner'ın **pets collection'**ı üzerinde yaptığı işlemde vazgeçmesi durumunda, pets collection'ı üzerinde yapılan **ekleme veya çıkarma işlemlerinin geri alınması şarttır!**

Senaryo 2

Owner Detail Tab View



The form has two tabs: "Owner Detail" (selected) and "Owner Pets". It contains three input fields: "First Name" with the value "Hasan", "Last Name" with the value "Arı", and "E-Mail (*)" which is empty and circled in red. Below the fields are two buttons: "Add" and "Cancel". A red arrow points from the "Add" button to the "Error Dialog" below.

Error Dialog



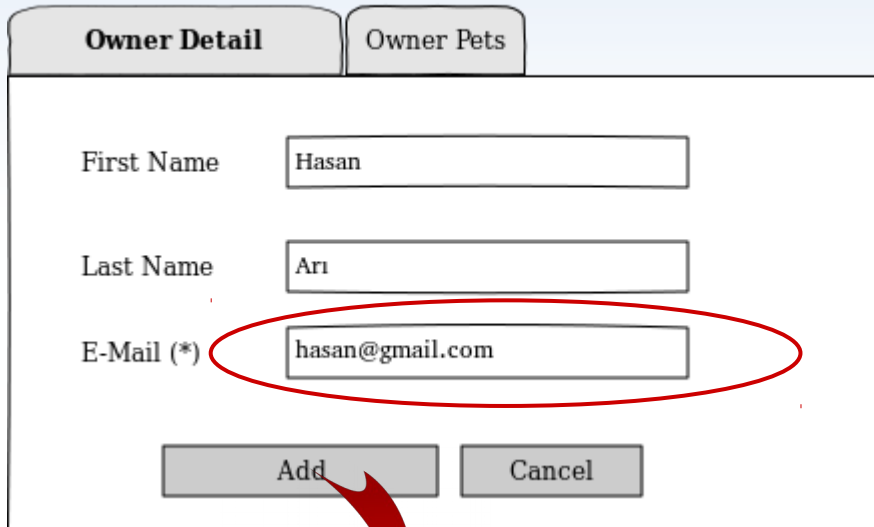
The dialog box has a title bar "Add New Pet Error" with standard window controls. The main text reads: "You need to provide a valid e-mail to add new Owner". At the bottom is a "Close" button.

Kullanıcı yeni bir Owner kaydı oluşturmak için detay ekranında Owner ile ilgili bilgileri girer ve "Add" butonuna tıklar

Ancak iş katmanında kullanıcının girdiği verinin eksik, yanlış veya iş kurallarına tam olarak uygun olmamasından kaynaklanan bir hata meydana gelir

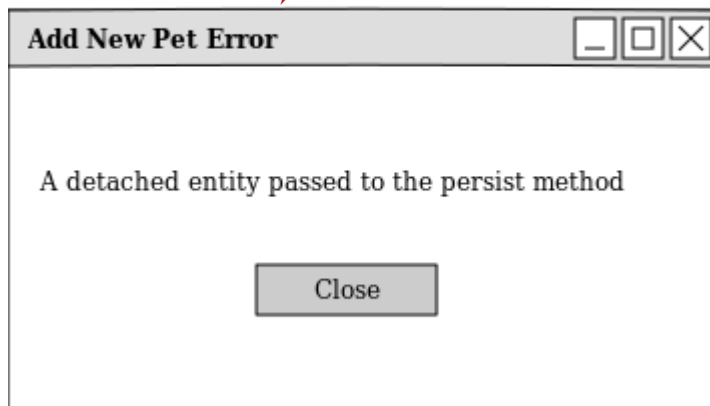
Senaryo 2

Owner Detail Tab View



The form has two tabs: "Owner Detail" (selected) and "Owner Pets". It contains three input fields: "First Name" with the value "Hasan", "Last Name" with the value "Arı", and "E-Mail (*)" with the value "hasan@gmail.com". The "E-Mail (*)" field is circled in red. Below the fields are two buttons: "Add" and "Cancel". A red arrow points from the "Add" button to the "Error Dialog" below.

Error Dialog



The error dialog has a title bar "Add New Pet Error" with standard window controls. The message inside reads: "A detached entity passed to the persist method". There is a "Close" button at the bottom.

Kullanıcı hatasını düzeltir ve tekrar "Add" butonuna tıklar

Ancak Owner domain nesnesinin state'i ilk Kayıt denemesi sırasında değiştiği için ikinci kez aynı domain nesnesi persist edilmeye çalışıldığı vakit persistence katmanında hata meydana gelir

Senaryo 2: Değerlendirme

- Persistence katmanı, “transient” state'deki Owner nesnesi persist edilmeye çalışıldığı vakit bu nesnenin **identifier property'sine bir PK değeri** atar
- Ancak iş katmanında meydana gelen bir hatadan dolayı **transaction rollback** olur ve kayıt DB'ye eklenemez
- İkinci denemede ise persistence katmanı bu Owner nesnesinin **identifier değeri mevcut olduğu için** onu “detached” state'de kabul eder ve kaydetme işlemi yine başarısız olur

Senaryo 2: Değerlendirme

- Domain nesnesinin state'i **transaction rollback sonrasında ilk haline geri döndürülmelidir**

Senaryo 3

- Owner – Pets ilişkisinin **lazy** biçimde yönetildiği bir senaryo olduğunu farz edelim

Owner List View

<input type="checkbox"/>	First Name	Last Name	E-Mail
<input type="checkbox"/>	Ali	Güç	ali@example.com
<input checked="" type="checkbox"/>	Veli	Doğru	veli@test.com
<input type="checkbox"/>	Cengiz	Çetin	cengiz@gmail.com
<input type="checkbox"/>	Ayşe	Us	ayse@yahoo.com

Add Owner

Remove Owners

Edit Owner

Owner Detail Tab View

Owner Detail

Owner Pets

First Name

Veli

Last Name

Doğru

E-Mail

veli@test.com.tr

Save Changes

Cancel

Kullanıcı bir grup Owner'ı listeler, listeden bir Owner kaydını seçerek detay ekranına geçer

Seçilen Owner kaydının bir takım alanları üzerinde değişiklikler yapar

Senaryo 3

Owner Pets Tab View

Owner Detail

Owner Pets

<input type="checkbox"/>	Name	Birth Date
<input type="checkbox"/>	Karabaş	01.01.2010
<input type="checkbox"/>	Cingöz	10.12.2015

Add Pet

Remove Pets

Edit Pet

Owner'ın sahip olduğu Pet kayıtlarını listelemek için “Owner Pets” tabına geçildiğinde lazy pets collection'ı yüklemek için detached Owner nesnesi persistence context'e re-attach edildiği vakit Owner üzerinde yapılan değişiklikler de yan etki olarak DB'ye yansıtılacaktır

Senaryo 3: Değerlendirme

- Lazy bir ilişkinin initialize edilmesi kendi başına bir işlem olarak ele alınabilmelidir
- Daha önce detached nesnede yapılan state **değişikliklerinin DB'ye re-attachmant sırasında yansıtılmaması** gerekmektedir

Senaryo 4

Owner List View

<input type="checkbox"/>	Full Name	E-Mail
<input checked="" type="checkbox"/>	Ali Güç	ali@example.com
<input type="checkbox"/>	Veli Doğru	veli@test.com
<input checked="" type="checkbox"/>	Cengiz Çetin	cengiz@gmail.com
<input type="checkbox"/>	Ayşe Us	ayse@yahoo.com

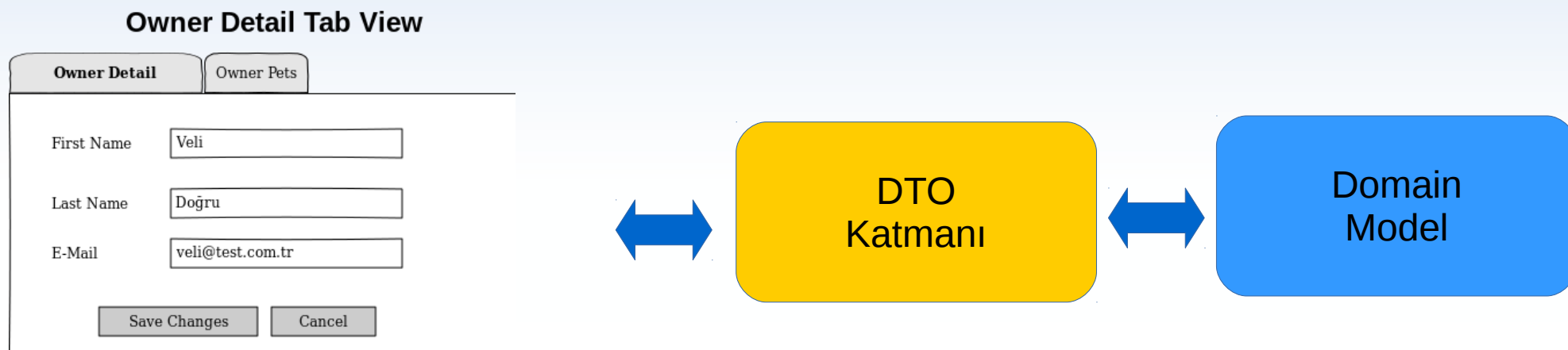
Owner kayıtlarının listelendiği ekranda tabloda hangi kayıtların seçildiği bilgisinin bir yerde takip edilmesi gerekebilir. Bunun için en pratik yer tabloya bind edilen Owner domain nesnelerinin kendileridir. Owner sınıfına selected isimli bir property eklenir. Bu property'nin görevi ekrandaki selection'ları takip etmektir. İş mantığı ile ilgisi yoktur.

Yine kullanıcının talepleri doğrultusunda firstName ve lastName bilgilerini ayrı ayrı görüntülemek yerine ekranda fullName şeklinde bir alanda beraber göstermek istenebilir. Bunun için de en pratik yol yine Owner sınıfına getFullName() isimli bir metot eklemektir. Bu metot içerisinde firstName ve lastName birleştirilip dönlür. Bu metodun da iş mantığı ile bir ilgisi yoktur.

Senaryo 4: Değerlendirme

- Domain sınıfına iş mantığı ile herhangi bir ilgisi olmayan **property ve metotlar** eklenmiş oldu
- Bu domain model farklı uygulamalarda kullanılmak için **yeniden kullanılabilir bir bileşen** olarak tasarlanabilir
- Bu durumda her bir uygulamanın UI gereksinimlerine göre bu tür eklemelerin yapılması **domain model'i kirletmiş** olacaktır

Çözüm !: DTO Katmanı



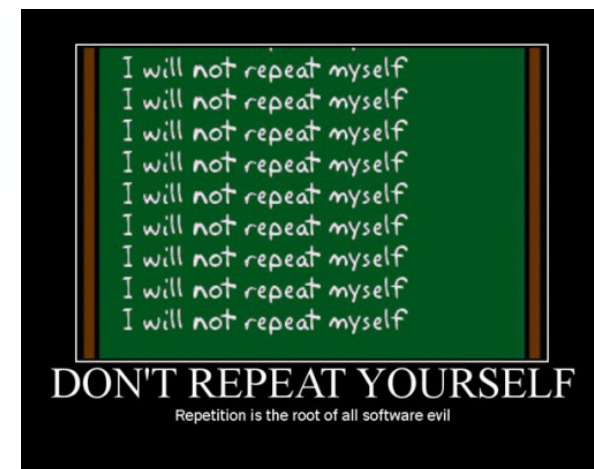
- UI'ın ihtiyaç duyduğu bilgi domain nesnelerinden alınarak DTO'ya aktarılır, **DTO UI'a bind** edilir
- UI bileşenleri DTO nesnelerine bind edildiği için **kullanıcı input'u önce DTO'da birikir**
- Bu input uygun zamanda **DTO'dan domain nesnelerine aktarılır** ve iş katmanında işlemler gerçekleştirilir

DTO, Bir Anti-Pattern Değil Mi?

- DTO, ilk dönem J2EE uygulamalarında **katmanlar arası veri taşımak** için kullanılmıştır
- Öncesi **Value Object** örüntüsüne dayanır
- EJB metot çağrılarının remote olması ve bu **remote çağrılar performans problemi** yaratması söz konusu idi
- Remote metot çağrılarının, giden gelen parametrelerin sayısını azaltmak için **DTO örüntüsünden yararlanılmıştır**

DTO, Bir Anti-Pattern Değil Mi?

- DTO örüntüsünün en çok eleştirildiği nokta **DRY prensibinin ihlalidir**
- DRY (dont repeat yourself) prensibine göre **bir iş sadece bir defa ve tek bir yerde yapılmalıdır**
- Çoğu zaman **domain sınıflarındaki property ve metotların büyük bir kısmı** DTO sınıflarında da tekrar etmektedir
- Bunlara ilaveten **bazı property ve metotlar DTO'ya özel** olarak eklenmektedir



DTO, Bir Anti-Pattern Değil Mi?

- Bir takım UI ve persistence framework'lerin domain sınıflarını doğrudan UI'a bindetmeyi cesaretlendirmeleri ile de DTO ağırlıklı olarak bir **anti-pattern** olarak nitelendirilmiştir



Günümüzde Mevcut Durum

- Günümüzde **JPA/Hibernate** gibi **bir persistence framework** ile domain nesneleri DB'den elde edilmektedir
- Ardından da **JSF** gibi bir **UI framework** ile geliştirilen ekranlara doğrudan bind edilmektedir
- Böylece UI üzerinden girilen verinin de doğrudan **domain nesneleri üzerinden DB'ye yansıtılması** genel geçer bir pratik halini almıştır



İsimlendirmede Revizyon: **View Model**

- Malesef DTO veya value object şeklinde bir isimlendirme **UI katmanı ile domain katmanı arasındaki ayrımın** gerekliliğini gölgelemiştir
- Dolayısı ile daha farklı bir isimlendirmeye gidilmesi **bu katmanın işlevini de tam ortaya koyması** açısından faydalı olacaktır
- UI katmanı ile doğrudan ilişkili olduğunu daha net ortaya koyması açısından bizim tercihimiz **View Model**'dir

DRY Problemi Aynen Devam Ediyor!

- Ancak **isimlendirmede revizyon** temel problemi ortadan kaldırmamıştır
- DRY prensibini ihlal etmeden **View Model katmanı nasıl** oluşturulabilir?



Çözüm : Dinamik Proxy Sınıf Üretmek !

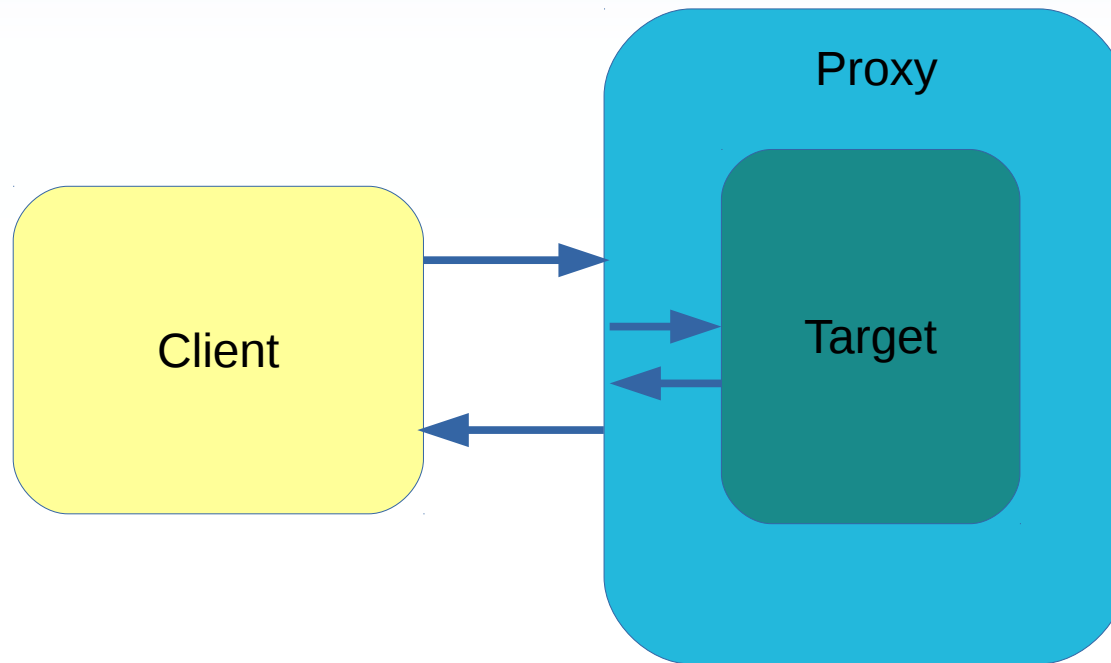
- **Proxy örüntüsü** ile domain sınıflarından dinamik olarak View Model sınıfları üretilebilir



Proxy Örüntüsü

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

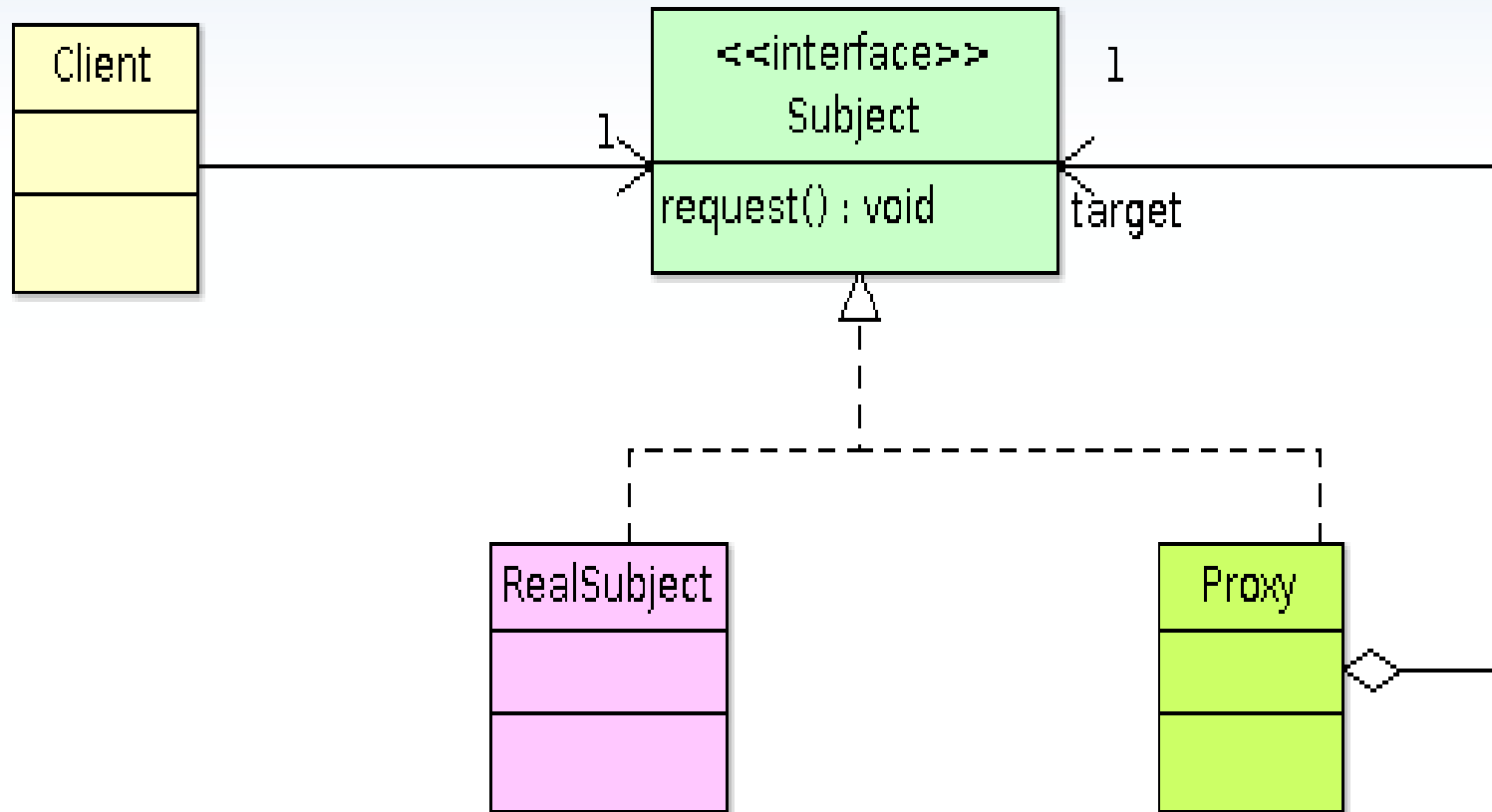
Client proxy nesne ile konuştuğunun farkında değildir



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

Proxy, metot çağrısından önce veya sonra bir takım işlemler gerçekleştirebilir

Proxy Sınıf Diagramı



Proxy Oluşturma Yöntemleri

- **Interface Proxy**

- Asıl nesnenin sahip olduğu arayüzler kullanılır
- JDK proxy olarak da bilinir

- **Class Proxy**

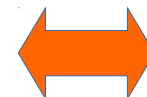
- Asıl nesnenin ait olduğu sınıf extend edilerek gerçekleştirilir
- CGLIB/Javassist proxy olarak da bilinir

View Model API

- View Model nesnelerinin UI ile persistent domain nesneleri arasında **köprü vazifesi görmelerini sağlayacak bir API**'ye de ihtiyaç vardır
- Üretilen proxy sınıflar domain sınıflarından türemelerinin yanı sıra bu API'ye de sahip olmalıdırlar



UI



Domain Model

View Model API

- **getModel**
 - Wrap edilen domain modele erişim sağlar
- **flush**
 - View Model nesnesi üzerinde biriken kullanıcı işlemlerini domain modele aktarır
- **refresh**
 - View Model state'ini domain model'in ilk haline döndürür
- **savepoint(id)/rollback(id)**
 - View Model'in current state'ini kaydedip, daha sonra istenirse bu state'e geri dönmeyi sağlar

View Model API

- **isDirty**
 - View model state'nin değişip değişmediğinin kontrolünü sağlar
- **isSelected/setSelected**
 - View model'in UI bileşeni içerisinde seçilip seçilmediğini takip eder
- **isTransient**
 - View model'in wrap ettiği domain model'in DB'de daha önce kaydedilip edilmediğini anlamayı sağlar
- **replace(Object model)**
 - View model'in wrap ettiği domain model'in başka bir nesne ile replace edilmesini sağlar

View Model API

- **addedElements(propertyName)**
 - Property name ile belirtilen collection property'si içerisine eklenen elemanları döner
- **removedElements(propertyName)**
 - Property name ile belirtilen collection property'si içerisinden çıkarılan elemanları döner
- **dirtyElements(propertyName)**
 - Property name ile belirtilen collection property'si içerisinde state'i değişen elemanları döner

View Model API in Action:

Senaryo 1'in View Model ile Gerçekleştirimi

```
EntityManager em = emf.createEntityManager();  
em.getTransaction().begin();
```

```
List<Owner> owners = em.createQuery(  
    "from Owner").getResultList();
```

```
List<Owner> viewModels = new  
    ArrayList<Owner>(owners.size());  
for(Owner model:owners) {  
    Owner viewModel = viewModelCreator  
        .create(Owner.class, model);  
    viewModels.add(viewModel);  
}
```

Owner List View

<input type="checkbox"/>	First Name	Last Name	E-Mail
<input type="checkbox"/>	Ali	Güç	ali@example.com
<input checked="" type="checkbox"/>	Veli	Doğru	veli@test.com
<input type="checkbox"/>	Cengiz	Çetin	cengiz@gmail.com
<input type="checkbox"/>	Ayşe	Us	ayse@yahoo.com

Add OwnerRemove OwnersEdit Owner

```
Owner selectedOwner = null;  
for(Owner viewModel:viewModels) {  
    if(((ViewModel<Owner>)viewModel)._isSelected()) {  
        selectedOwner = viewModel;  
        break;  
    }  
}
```

View Model API in Action:

Senaryo 1'in View Model ile Gerçekleştirimi

Owner Detail Tab View

Owner Detail
Owner Pets

First Name
Veli

Last Name
Doğru

E-Mail
veli@test.com.tr

Save Changes
Cancel

```
selectedOwner.setEmail("veli@test.com.tr");
```

...

```
((ViewModel<Owner>) selectedOwner)
    ._savepoint_("pets_tab_view");
```

Owner Pets Tab View

Owner Detail
Owner Pets

<input type="checkbox"/>	Name	Birth Date
<input type="checkbox"/>	Karabaş	01.01.2010
<input checked="" type="checkbox"/>	Cangöz	10.12.2015

Add Pet
Remove Pets
Edit Pet

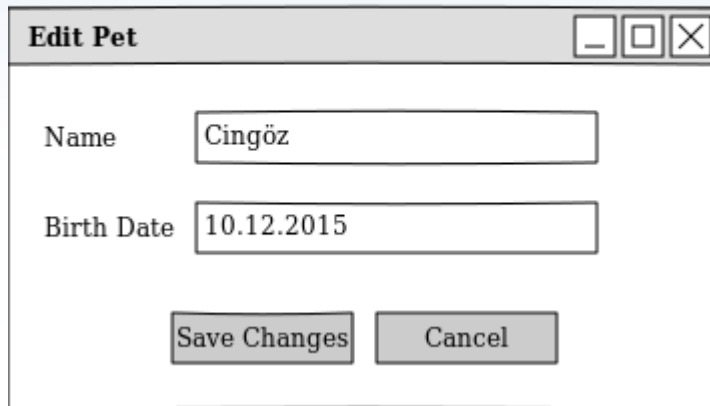
```
Pet selectedPet = null;
```

```
for(Pet pet:selectedOwner.getPets()) {
    if(((ViewModel<Pet>)pet)._isSelected()) {
        selectedPet = pet;
        break;
    }
}
```

View Model API in Action:

Senaryo 1'in View Model ile Gerçekleştirimi

Edit Pet Dialog



A screenshot of a Windows-style dialog box titled "Edit Pet". It contains two text input fields: "Name" with the value "Cingöz" and "Birth Date" with the value "10.12.2015". At the bottom, there are two buttons: "Save Changes" and "Cancel".

```
selectedPet.setName("Cingöz");
```

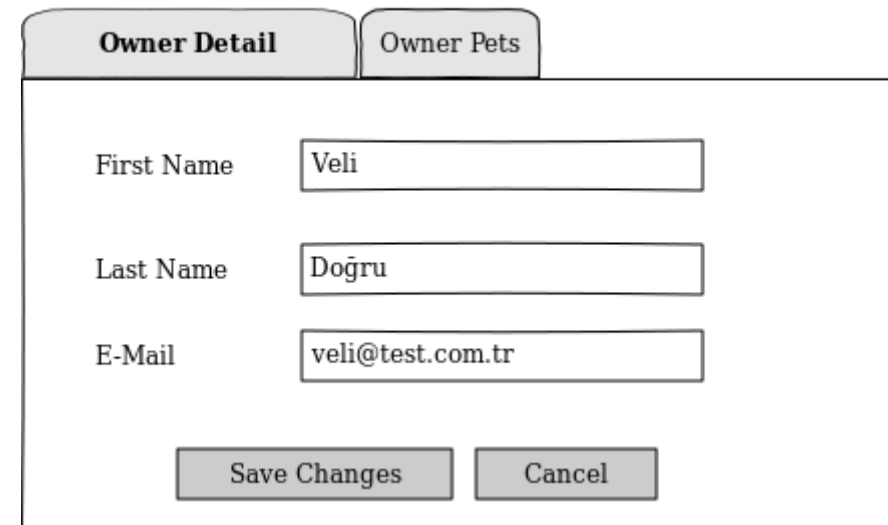
...

```
((ViewModel<Owner>)selectedOwner)  
    ._rollback_("pets_tab_view");
```

```
((ViewModel<Owner>) selectedOwner)._flush();
```

```
em.getTransaction().commit();  
em.close();
```

Owner Detail Tab View



A screenshot of a web application showing a tabbed interface. The "Owner Detail" tab is active, displaying three text input fields: "First Name" with the value "Veli", "Last Name" with the value "Doğru", and "E-Mail" with the value "veli@test.com.tr". At the bottom, there are two buttons: "Save Changes" and "Cancel". The "Owner Pets" tab is also visible but inactive.

View Model API in Action:

UI'a Özel Alanların Eklenmesi

```
public interface OwnerViewModel {
    public String getFullName();
}
```

```
public class OwnerViewModelImpl
    extends ViewModelImpl<Owner>
    implements OwnerViewModel {
    public OwnerViewModelImpl(Owner model,
        ViewModelDefinition definition) {
        super(model, definition);
    }
}
```

```
@Override
public String getFullName() {
    String firstName = _getModel().getFirstName();
    String lastName = _getModel().getLastName();
    String fullName = "";
    if (StringUtils.isEmpty(firstName)) {
        fullName += firstName;
    }
    if (StringUtils.isEmpty(lastName)) {
        if (StringUtils.isEmpty(fullName)) {
            fullName += " ";
        }
        fullName += lastName;
    }
    return fullName;
}
```

Owner List View

<input type="checkbox"/>	Full Name	E-Mail
<input checked="" type="checkbox"/>	Ali Güç	ali@example.com
<input type="checkbox"/>	Veli Doğru	veli@test.com
<input checked="" type="checkbox"/>	Cengiz Çetin	cengiz@gmail.com
<input type="checkbox"/>	Ayşe Us	ayse@yahoo.com

Add Owner

Remove Owners

Edit Owner

View Model API in Action:

UI'a Özel Alanların Eklenmesi

```
public class PetClinicViewModelDefinitionProvider
    implements ViewModelDefinitionProvider {

    @Override
    public Collection<ViewModelDefinition> getViewModelDefinitions() {
        ViewModelDefinition petDef = new ViewModelDefinition(Pet.class);
        ViewModelDefinition ownerDef =
            new ViewModelDefinition(Owner.class, OwnerViewModelImpl.class);
        ownerDef.addDefinition("pets", petDef);
        return Arrays.asList(ownerDef, petDef);
    }
}
```

Sonuç

- Persistent domain nesnelerinin doğrudan UI katmanında kullanılması **bir takım problemlere** yol açmaktadır
- UI ile domain model arasında **ara bir katmana** ihtiyaç vardır
- UI ile domain model arasında köprü vazifesi gören bu katman üzerinde çalışmak için **bir de API gereklidir**
- **View Model** olarak adlandırılan bu katman **dinamik proxy sınıf üretme yöntemi** ile oluşturulabilir



Soru & Cevap



İletişim

- **Harezmi** Bilişim Çözümleri A.Ş.
- <http://www.harezmi.com.tr>
- info@harezmi.com.tr