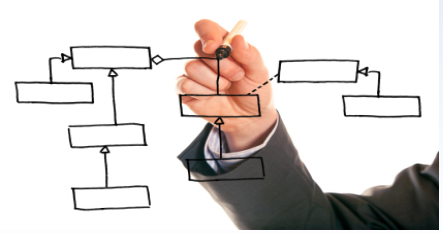
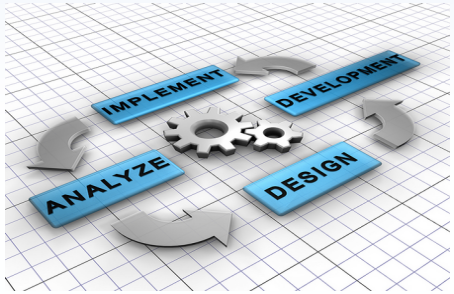
Dynamic Proxy Based View Model API

harezmi
it solutions

Who is Kenan Sevindik?

- Over **15 years** of enterprise application development **experience**
- Involved in creation and **development of architectures** of various projects
- Has **extensive knowledge and experience** about enterprise Java technologies, such as Spring, Spring Security, Hibernate, Vaadin and so on...





spring

Java

Spring SECURITY

Hibernate JPA

vaadin

Who is Kenan Sevindik?

- Co-author of **Beginning Spring** Book
- Founded **Harezmi IT Solutions** in 2011
- What Harezmi does?
  - Involves in **development** of enterprise Java applications
  - Offers **consultancy** and mentoring services
  - Organizes enterprise Java technologies related **trainings**

harezmi bilişim çözümleri

Speedy Framework

JAVA Eğitimleri

Problem

Several problems arise in case persistent **domain objects** fetched via JPA/Hibernate **are directly bound to UI** layer!

Scenario 1

- Let's assume we have a typical **master-detail** UI in which **Owner** and its **Pets** are displayed and edited

Owner List View

	First Name	Last Name	E-Mail
☐	Ali	Güç	ali@example.com
☑	Veli	Doğru	veli@test.com
☐	Cengiz	Çetin	cengiz@gmail.con
☐	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

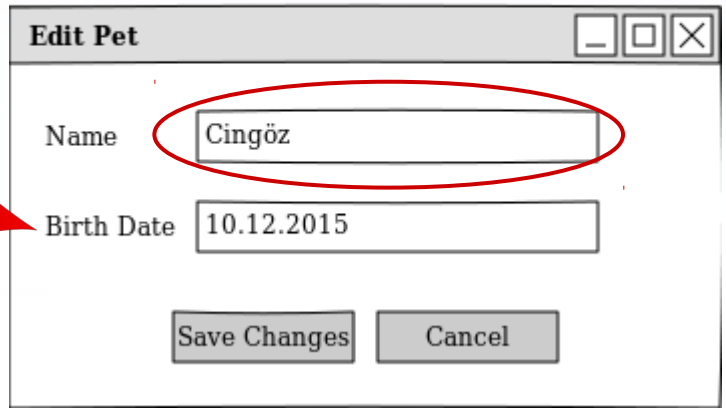
User lists a group of Owners, and selects an Owner from the list, then goes into the detail view

He performs changes over some of the properties of selected Owner, but doesn't click “Save Changes” button yet

Scenario 1

Owner Pets Tab View

Edit Pet Dialog



Later, without saving those performed changes, it switches into Owner Pets tab, and starts editing a Pet instance from there

Both changes in Owner record, and Pet record are reflected altogether, when he clicks “Save Changes” button after finishing his changes over Pet record

Scenario 1: Overview

- Both **changes in Owner and Pet records** are reflected into DB althogether
- User was not able to **cancel out changes made over Pet**, and reflect only the changes made over Owner
- User was not able to cancel out changes **made over Owner**, and reflect only the changes made over Pet, either

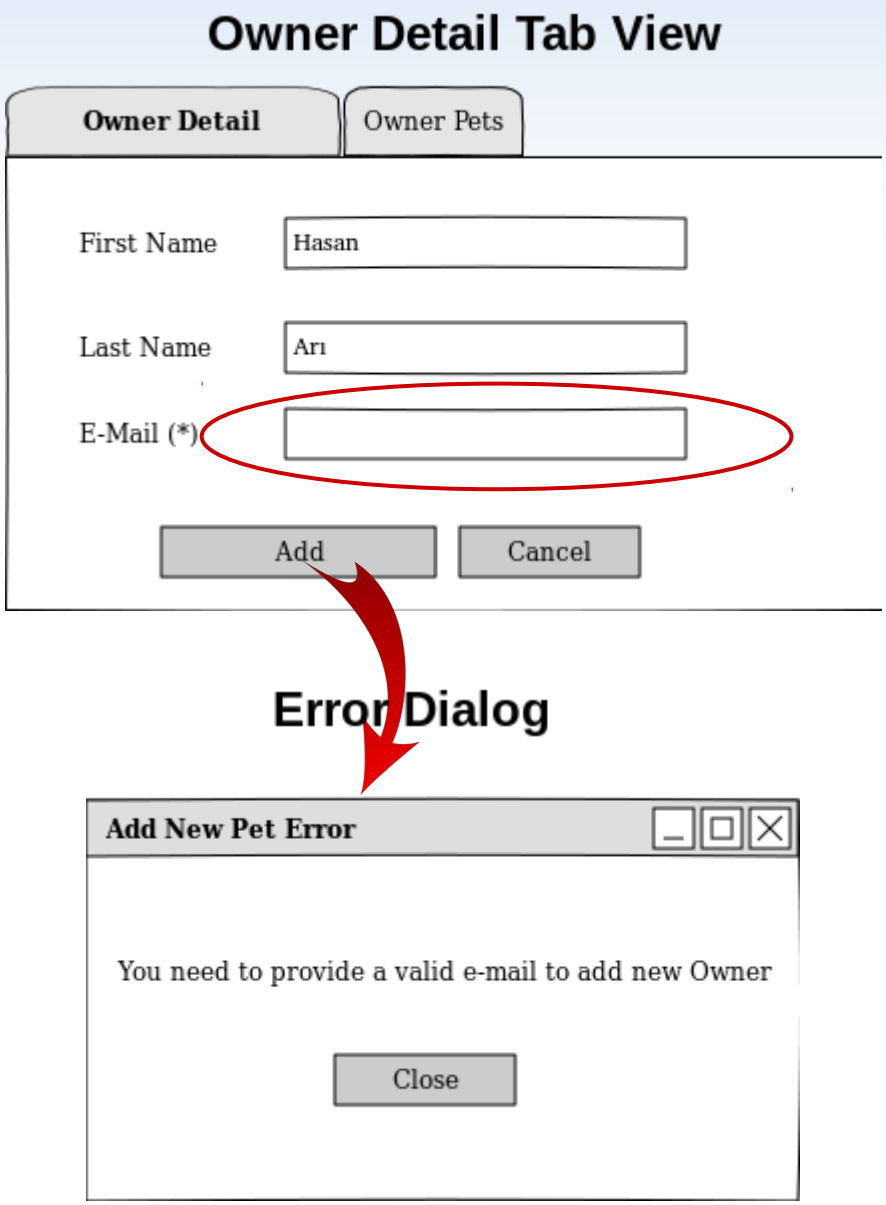
Scenario 1: Overview

- It is necessary that **old property values** should be **stored somewhere else**, so that user **changes could be reverted** back using those stored values whenever user cancels out his operation

Scenario 1 Derivatives

- Similar results with this scenario may occur like **addition of a new Pet** record
- Or **deletion of an existing Pet** record unintentionally
- It is necessary that **changes in pets collection** should be reverted back whenever **user decides to cancel out** his current operation

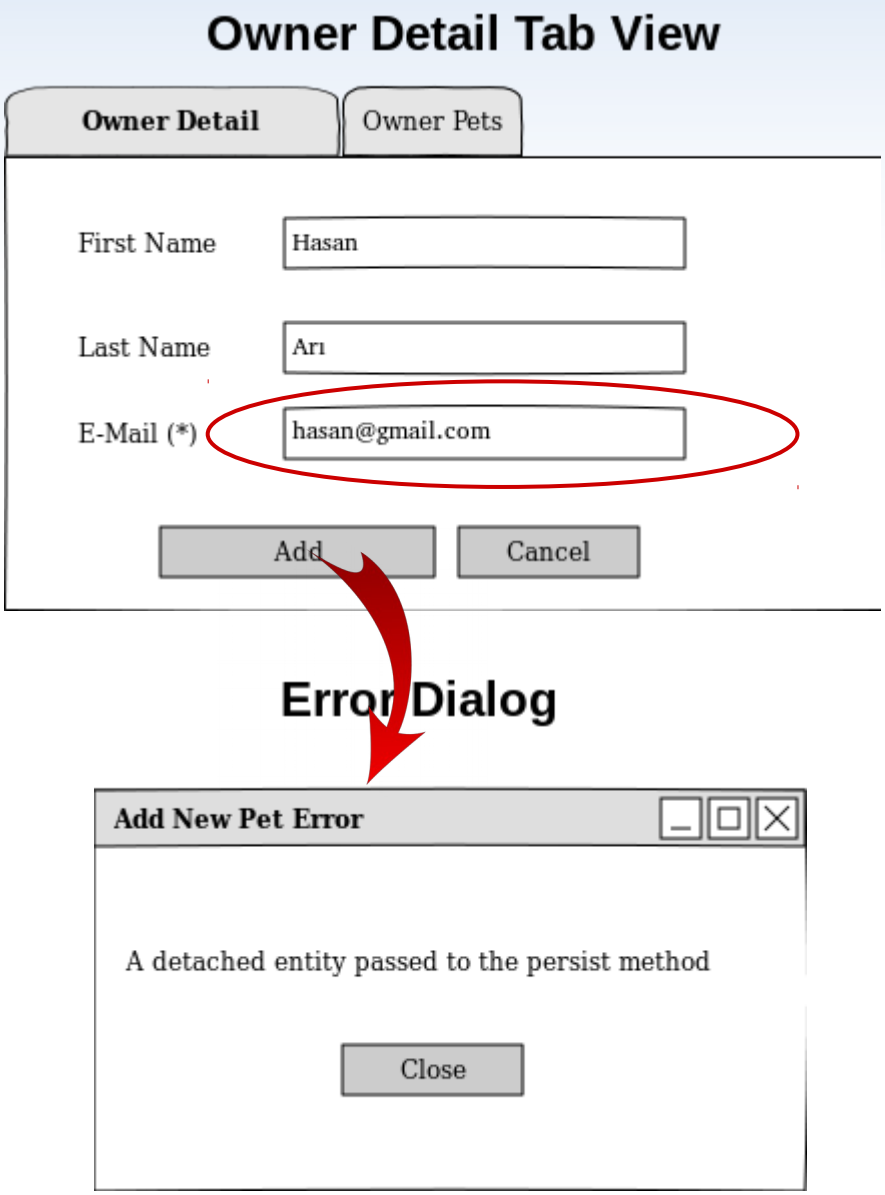
Scenario 2



User enters details in order to create a new Owner record in DB and clicks “Add” button

However, an error is rised from within business layer because of errornous data entered by the user

Scenario 2



User corrects his fault, and clicks “Add” button again

Unfortunately, this time persistence layer fails because of the state change occured within domain object during his first attempt

Scenario 2: Overview

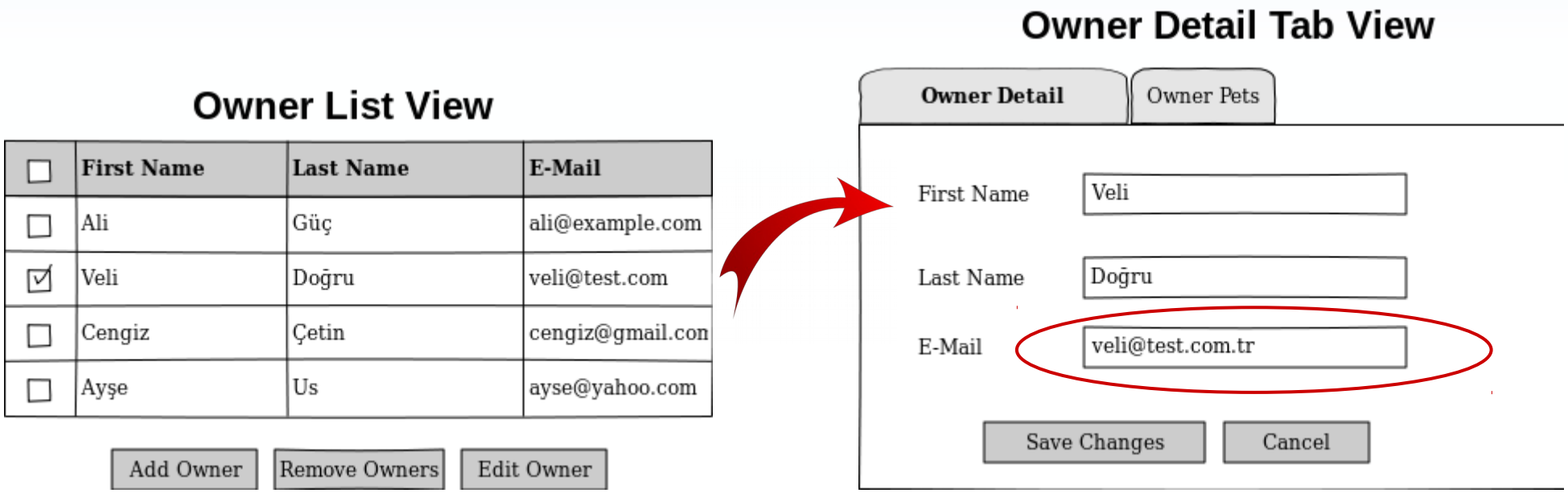
- Persistence layer **assigns PK value** to identifier property of Owner object which is in transient state during persist operation
- However, **transaction is rollbacked** and Owner is not saved into DB due to error occured within business layer
- During the second trial, persistence layer thinks that Owner is in **detached state because of the assigned PK value** and fails persistence operation because of this

Scenario 2: Overview

- State of the domain object should have ben **reverted back to its initials** when transaction rolled back during the first attempt

Scenario 3

- Assume, Owner – Pets association is **fetched on demand** (lazy)



User lists a group of Owner records, selects an Owner within the list, and navigates to the detail view

User works on several properties of selected Owner record, changes them and so on

Scenario 3

Owner Pets Tab View

When he switches to the “Owner Pets” tab in order to list Pet records, detached owner is re-attached to the persistence context in order to fetch contents of the pets collection from DB. During this re-attachment, however, those changes performed previously will be flushed into DB as a side effect of this re-attachment

Scenario 3: Overview

- **Lazy association** should have been handled on its own
- Any **changes performed on the detached object** should not have been reflected into DB as a side effect of this handling of lazy association

Scenario 4

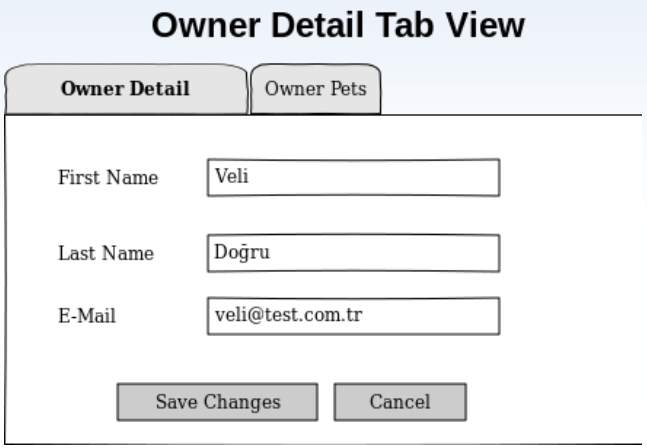
It is required to store selection information of Owner records somewhere in the application. The most practical place for this looks like Owner domain objects themselves. A property for such purpose is added into domain classes and used only to store selection of domain Object. It is not related with business logic at all.

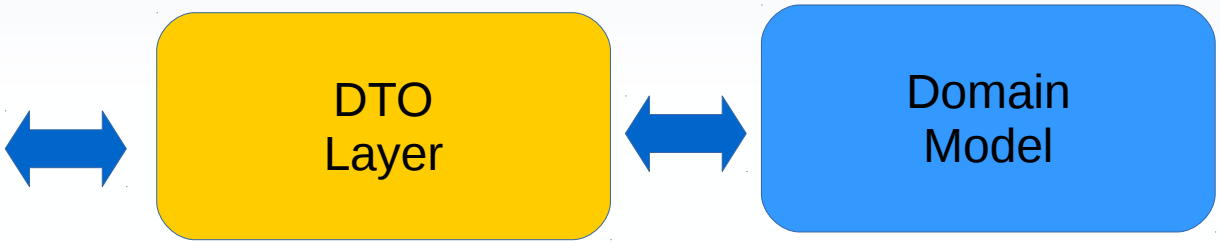
Users may ask for Owners' names to be listed together as “fullName”, instead of firstName and lastName separated. Again, the easiest way to achieve this looks like adding a method as getFullName() to return firstName and lastName concatenated. This method has nothing to do with business logic, either.

Scenario 4: Overview

- **Properties and methods** which have **nothing to do with business logic** have been added into domain classes
- Those **domain classes** would be aimed to be used **as reusable units** in different applications
- In such a case, adding such properties and methods would **pollute domain model** at hand

Solution !: DTO Layer





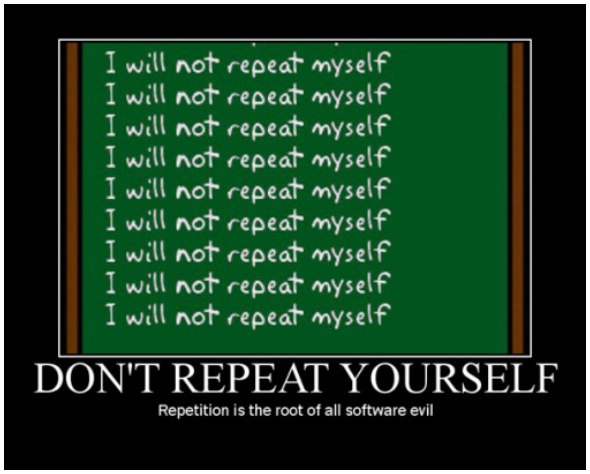
- **Data** needed by the UI is **obtained from domain objects** and **transferred into DTOs** and DTOs are **bound to the UI**
- **Use input**, therefore, is first **accumulated into DTOs** as UI components are bound to DTOs
- The input accumulated within DTOs are **transferred into domain** instances **at appropriate times** and business is performed using it

DTO, Wasn't It An Anti-Pattern?

- In the early ages of Java EE application development, DTOs were used to **transfer data between layers**
- It predates back to **Value Object pattern**
- **EJB method calls** were remote only, and those remote procedure calls were **causing performance problems**
- **DTOs** were then employed in order **to reduce communication overhead** of those RPCs caused by translation of excessive number of method parameters

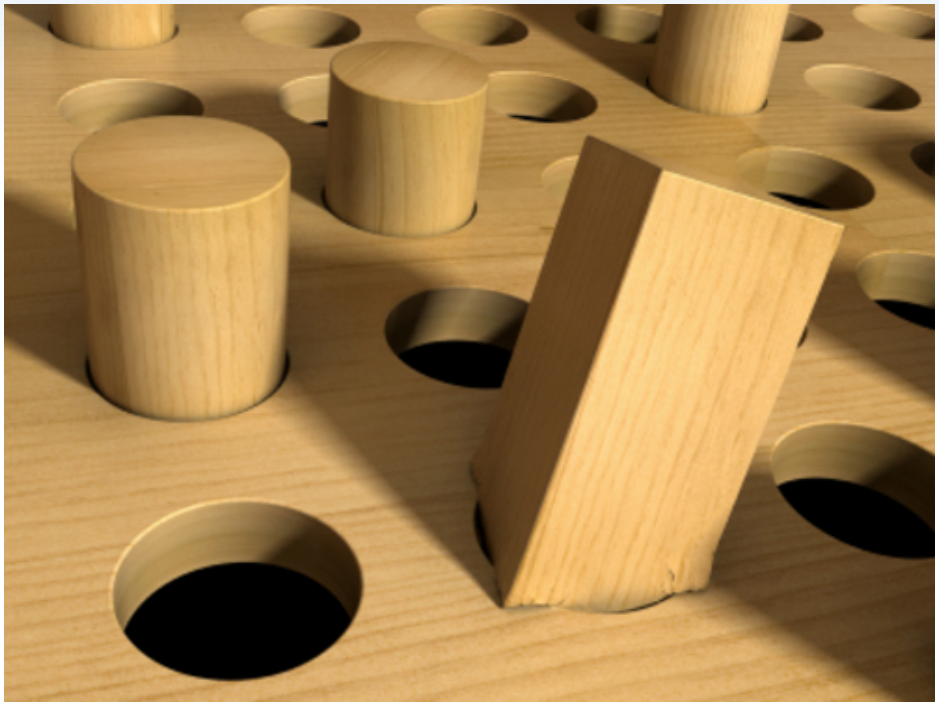
DTO, Wasn't It An Anti-Pattern?

- The most criticized aspect of DTO pattern is its **violation of DRY principle**
- According to DRY (dont repeat yourself) principle, **a task should be implemented only once and at only one single place** in the system
- Most of the time, many of the **properties and methods in domain classes are simply repeated in DTO** classes as well
- Apart from such repetition, several other **properties and methods specific to DTOs** are added, too



DTO, Wasn't It An Anti-Pattern?

- DTOs, today mostly are qualified **as anti-pattern** because of such repetition, and ecouragement of several UI and persistence frameworks to bind UI to domain classes directly



Today's Situation

- Nowadays, domain instances are usually **fetched from DB**, using a persistence framework, such as JPA/Hibernate
- Afterwards, they are **directly bound to UI** components which are developed using a UI framework, life JSF
- Hence, transferring **user input from over domain objects directly into the DB** has become mainstream

Revision in Naming: **View Model**

- Unfortunately, such a naming like DTO or Value Object may cause underestimation of the need of **separating UI and domain layers** from each other
- Therefore, entitling the solution with **a different name** might be useful in terms of revealing functionality of such a new layer
- Our preference is to use **View Model** as it reveals its direct relationship with UI layer more

Problem with DRY Still Exists!

- However, revision in the naming doesn't help us to get away from **core of the problem**
- How such a View Model layer can be generated **without violating DRY principle**?

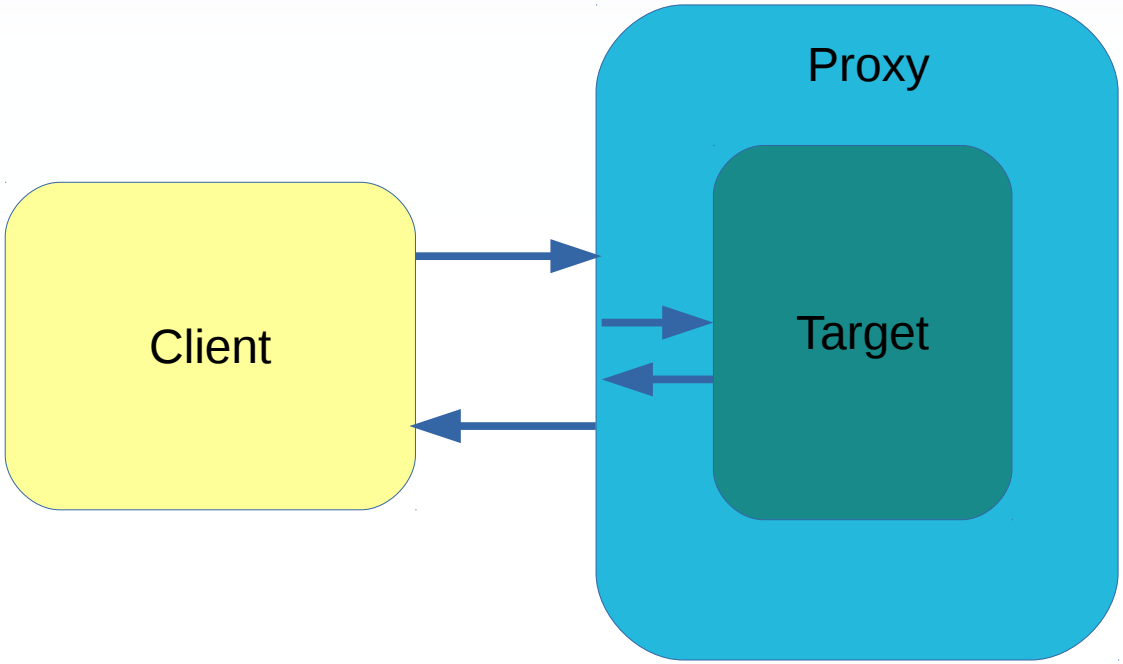
Solution : Dynamic Proxy Class Generation !

- View Model classes are generated out of domain classes dynamically using **Proxy pattern**

Proxy Pattern

Proxy is of same type with its target, and it intercepts method calls occuring between client and the target

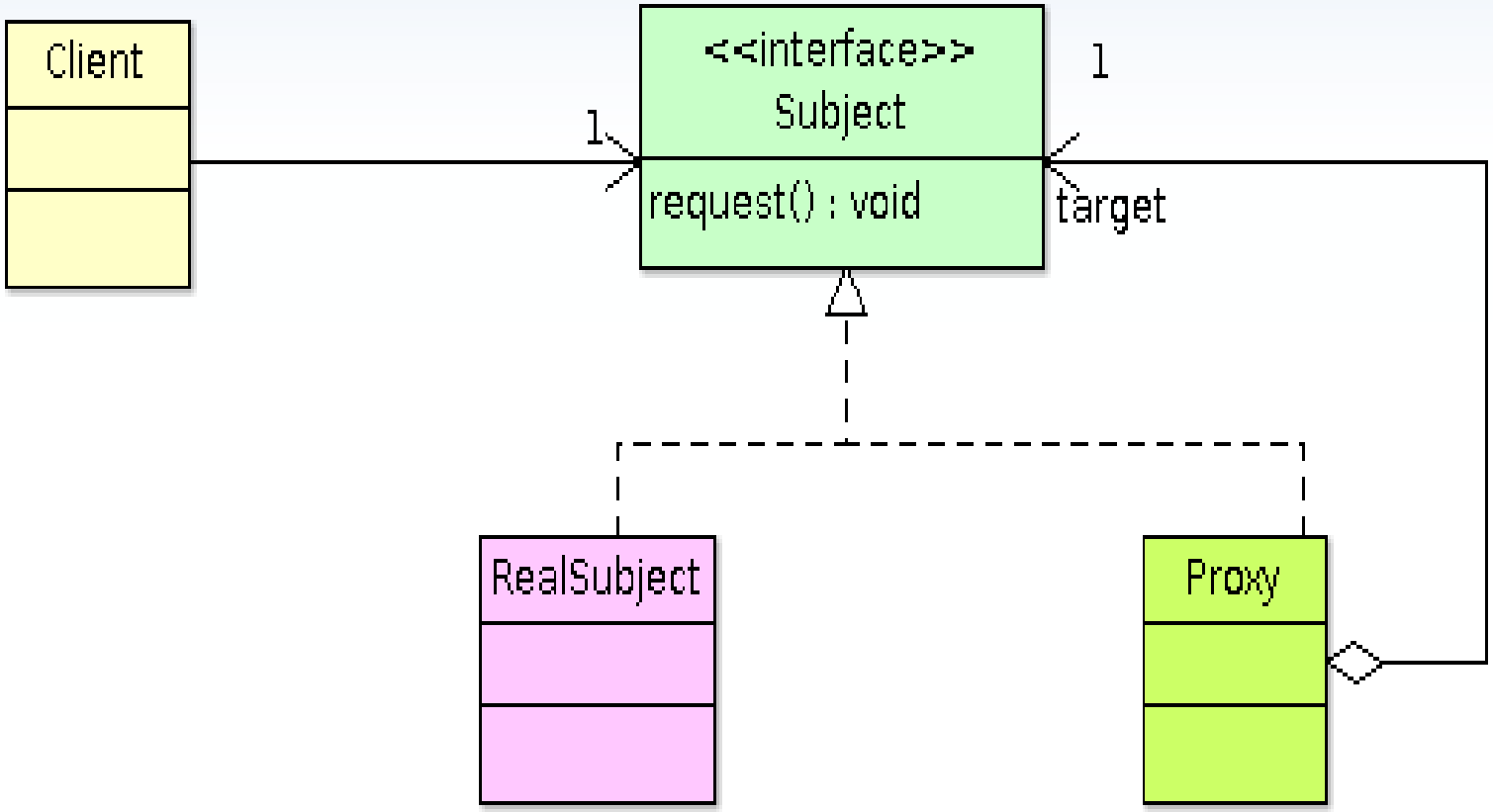
Client, on the other hand, is not aware of it interacts with proxy instance instead



Method calls from client first arrive at proxy instance before reaching their target destination

Proxy, before and after thos method calls can perform tasks related with the call

Proxy Class Diagram



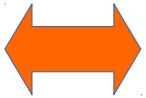
Proxy Generation Strategies

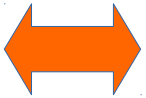
- **Interface Proxy**
  - Interfaces implemented by the actual model class are used to generate proxy class
  - Known as JDK proxy
- **Class Proxy**
  - Domain model class is extended to generate proxy class
  - Known as CGLIB or Javassist proxy

View Model API

- An API, in the role of a **bridge between UI and persistent domain objects** is necessary to operate
- **Proxy classes** generated from those domain classes should also **own this API** as well

UI





Domain Model

View Model API

- **getModel**
  - Allows acces to the wrapped domain model instance
- **flush**
  - Reflects changes accumulated in the view model instance into the wrapped domain model
- **refresh**
  - Reverts state of the view model into its initial version
- **savepoint(id)/rollback(id)**
  - Allows to save current state of view model associating it with the given identifier, then to roll back the changes in the view model state back to the state identified by the given identifier

View Model API

- **isDirty**
  - Detects if view model state has been changed or not
- **isSelected/setSelected**
  - Helps to identify if view model instance is selected within the bounded UI component, and to mark it as selected
- **isTransient**
  - Helps to check if domain model wrapped by the view model is persisted into DB before or not
- **replace(Object model)**
  - Replaces given model object with the already wrapped model instance within the view model

View Model API

- **addedElements(propertyName)**
  - Returns elements which are added into the collection property identified by the given propertyName
- **removedElements(propertyName)**
  - Returns elements which are removed from the collection property identified by the given propertyName
- **dirtyElements(propertyName)**
  - Returns elements whose state has been changed in the collection property identified by the given propertyName

View Model API in Action:

Implementing Scenario 1 Using View Model API

```
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

	First Name	Last Name	E-Mail
☐	Ali	Güç	ali@example.com
☑	Veli	Doğru	veli@test.com
☐	Cengiz	Çetin	cengiz@gmail.con
☐	Ayşe	Us	ayse@yahoo.com

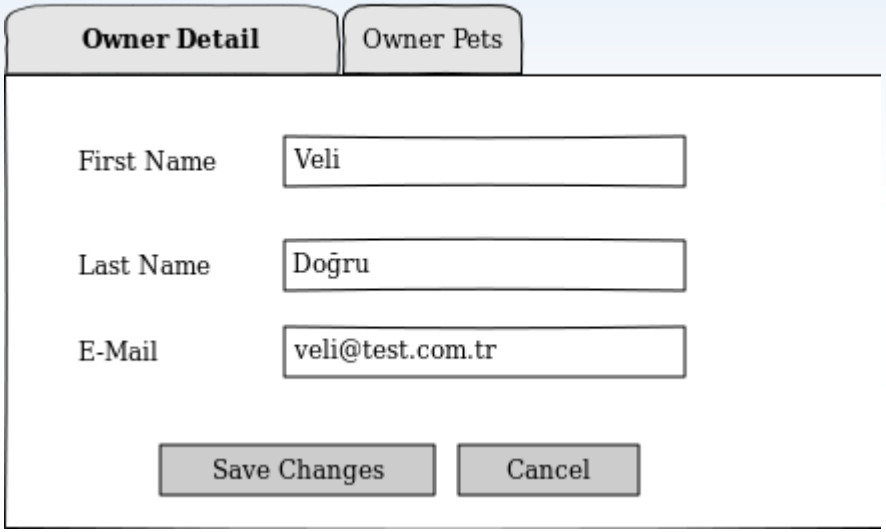
Add Owner | Remove Owners | Edit Owner

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

View Model API in Action:

Implementing Scenario 1 Using View Model API

Owner Detail Tab View




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

```
...

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

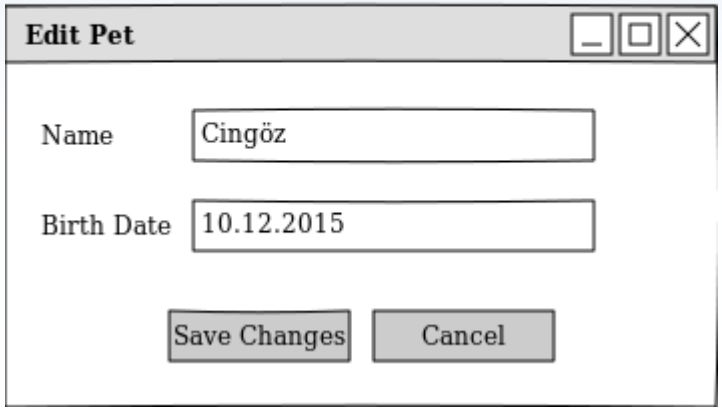
Owner Pets Tab View


```
Pet selectedPet = null;

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

View Model API in Action:

Implementing Scenario 1 Using View Model API

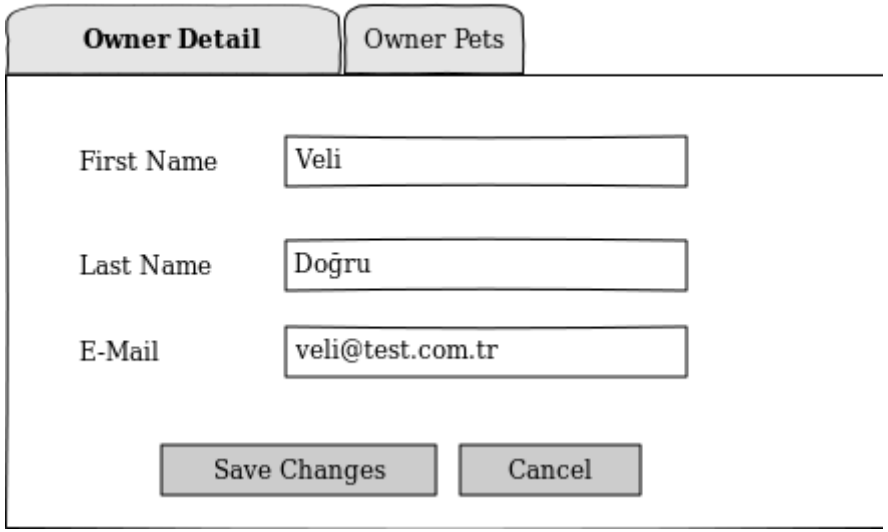
Edit Pet Dialog




```
selectedPet.setName("Cingöz");

...

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

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

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

Owner Detail Tab View



View Model API in Action:

Adding UI Specific Fields & Methods

Owner List View

☐	Full Name	E-Mail
☑	Ali Güç	ali@example.com
☐	Veli Doğru	veli@test.com
☑	Cengiz Çetin	cengiz@gmail.com
☐	Ayşe Us	ayse@yahoo.com

Add Owner | Remove Owners | Edit Owner

```
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.isNotEmpty(firstName)) {
            fullName += firstName;
        }
        if (StringUtils.isNotEmpty(lastName)) {
            if (StringUtils.isNotEmpty(fullName)) {
                fullName += " ";
            }
            fullName += lastName;
        }
        return fullName;
    }
}
```

View Model API in Action:

Adding UI Specific Fields & Methods

```
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);
    }
}
```

Conclusion

- Reusing persistent domain objects within the UI layer causes several **persistence related problems** in the system
- An **intermediate layer between UI and domain model** is required in this case
- **An API to execute operations** through this intermediate layer which becomes a bridge between UI and domain model
- Such a layer, which is called as **"View Model"** can be created by employing **dynamic proxy class generation** method

Questions & Answers

Contact

- Harezmi IT Solutions
- http://www.harezmi.com.tr
- info@harezmi.com.tr