JAVA Eğitimleri

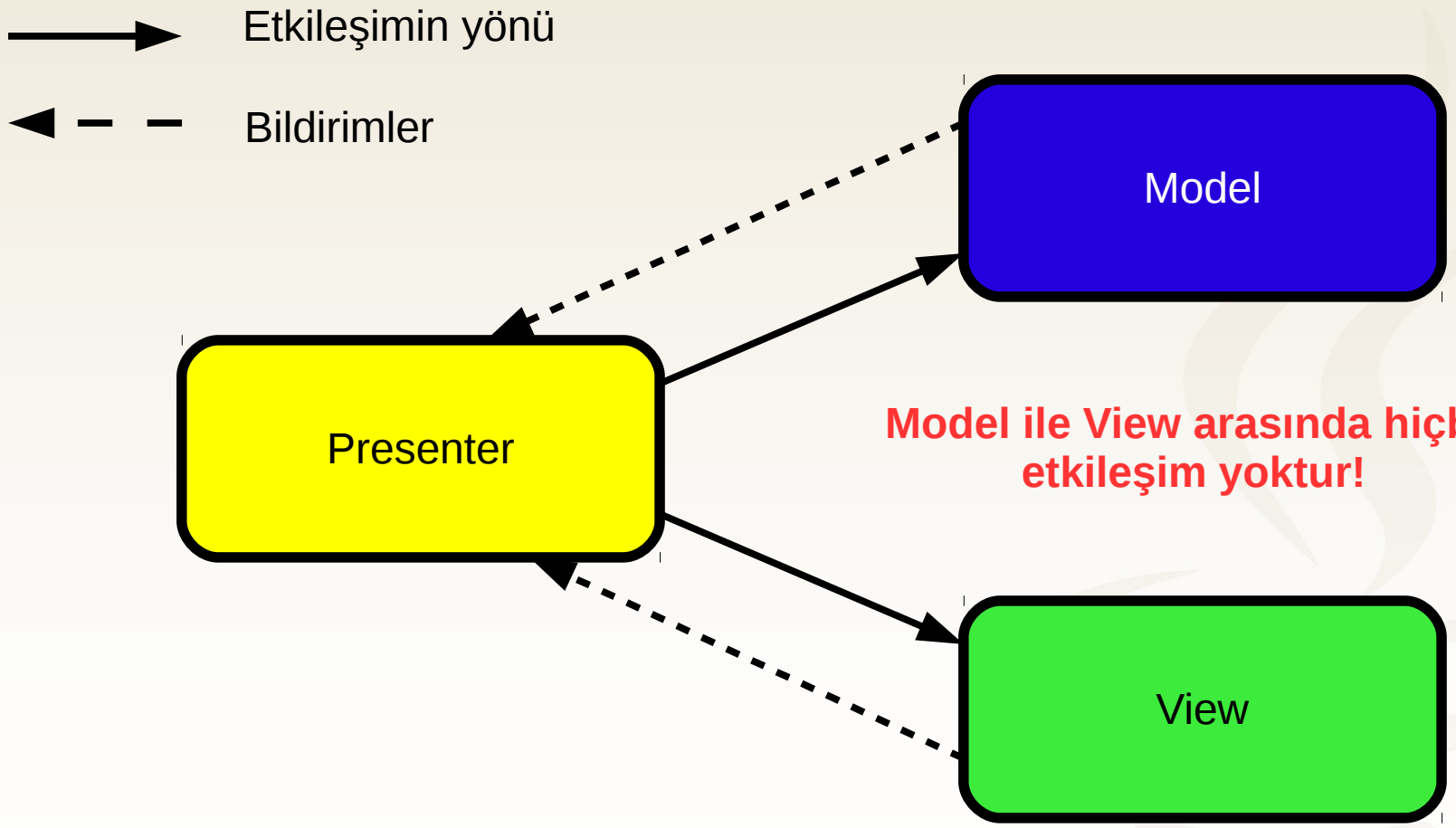
MVP ve Mediator ile Modüler UI Geliştirme

harezmi bilişim çözümleri

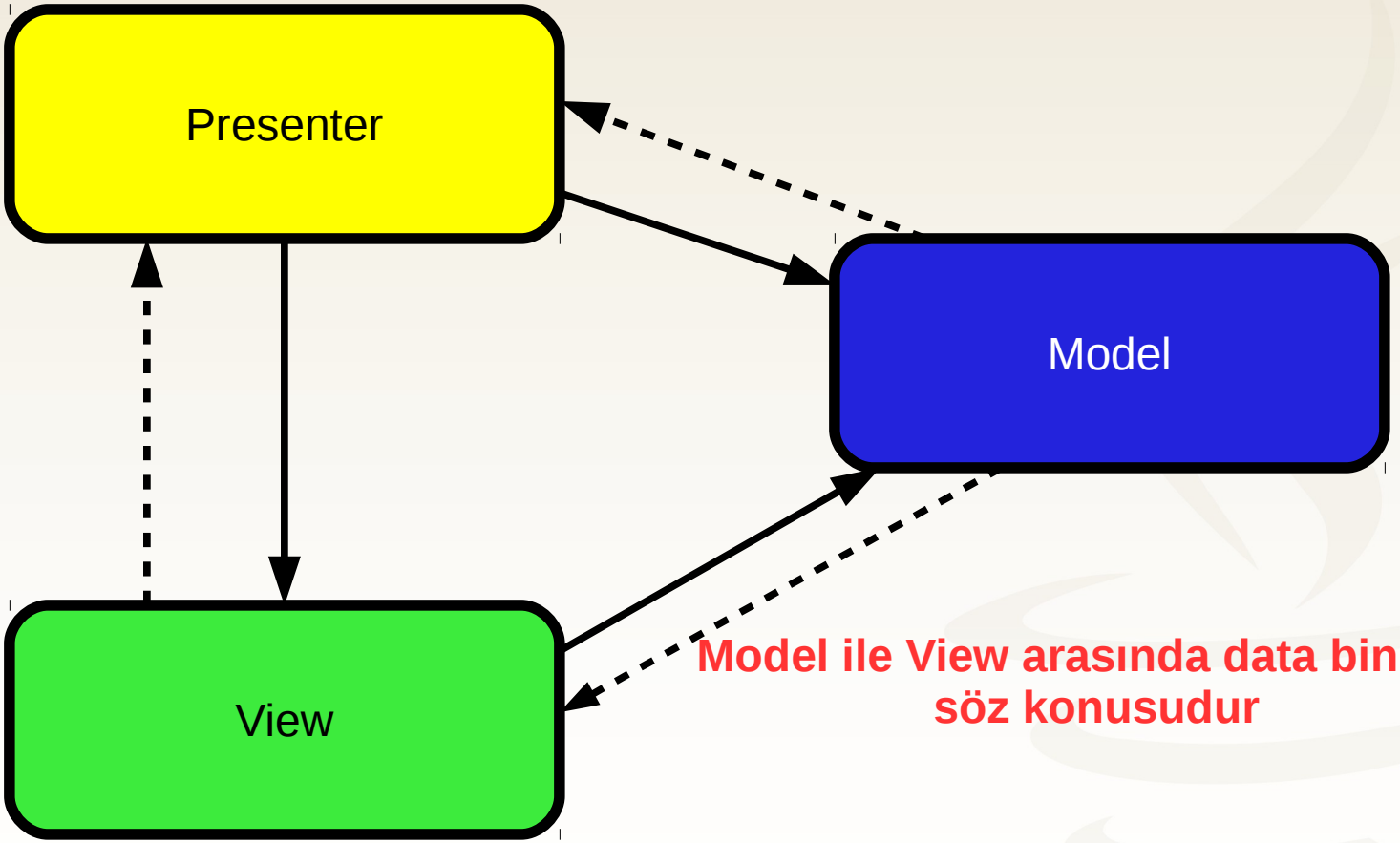
MVP Nedir?

- **MVP** örüntüsü **MVC'nin bir türevi**dir
- Amacı View tarafındaki **UI mantığının** UI bileşenlerinden tamamen çıkartılması ve
- UI bileşenlerinin görevinin sadece **UI render** ile sınırlandırılmasıdır
- **İki farklı türü** vardır
  - Passive View
  - Supervising Controller

Passive View



Supervising Controller



Model ile View arasında data binding söz konusudur

MVP ve TDD

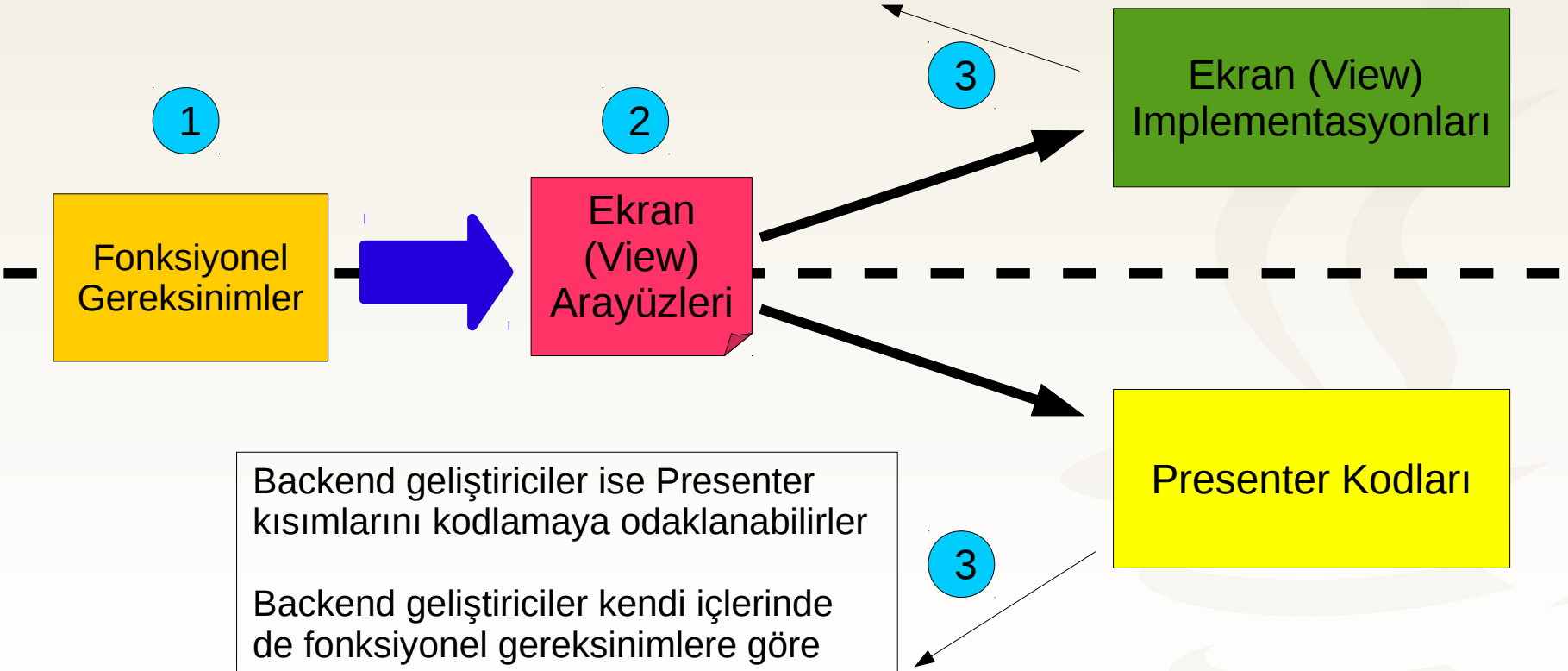
- MVP örüntüsü, **TDD** ile yazılım geliştirmeyi de kolaylaştırmaktadır
- Presenter sınıflarının **UI bileşenlerinden bağımsız** biçimde geliştirilmesi mümkündür
- Kullanıcı senaryoları ve gereksinimleri bire bir **Presenter içindeki fonksiyonlar**a karşılık gelmektedir

MVP ve TDD

- Presenter kodları geliştirilirken ilgili **model** ve **view** nesnelerine veya **servis bileşenleri**ne ihtiyaç duyulur
- Bu nesnelerin gerçek implemantasyonları yerine sahteleri **(mock)** Presenter'a sunulabilir
- Böylece Presenter kodları **diğer katmanlardan bağımsız biçimde** kendi başına geliştirilebilir

MVP ve TDD

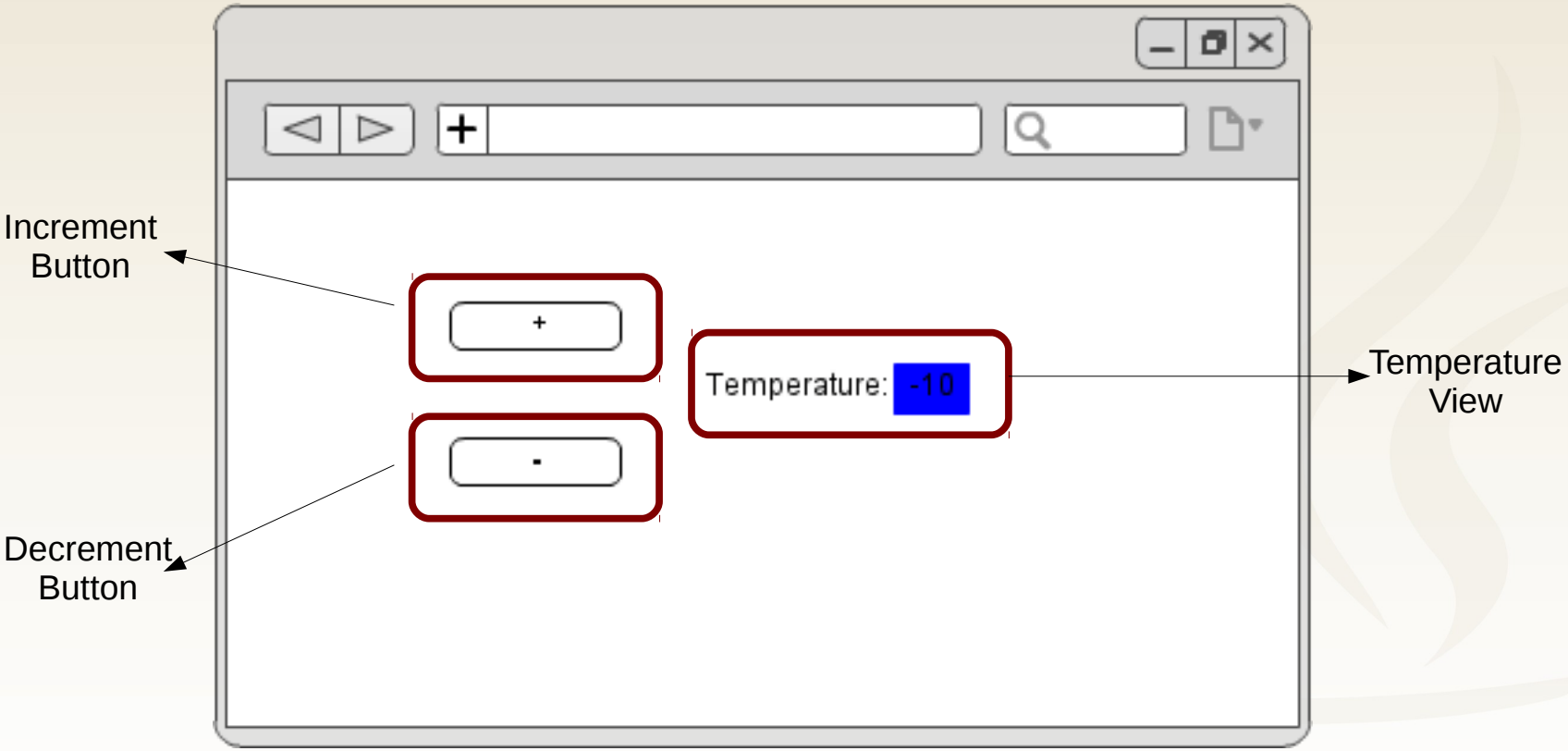
UI geliştiriciler tamamen GUI geliştirmeye odaklanabilirler. View içerisinde sadece UI widget'ların oluşturulması ve sayfalara yerleştirilmesi söz konusudur



Backend geliştiriciler ise Presenter kısımlarını kodlamaya odaklanabilirler

Backend geliştiriciler kendi içlerinde de fonksiyonel gereksinimlere göre gruplara ayrılarak paralel biçimde de çalışabilirler

LAB ÇALIŞMASI



Kullanıcı + ve – butonlarına tıklayarak sıcaklık değerini birer birer artırıp, düşürebilecektir. Sıcaklık değeri 0 ve 0'ın altında iken arka plan mavi, 0'ın üstünde yeşil, 20'nin üstünde sarı, 30'ın üstünde de kırmızı olmalıdır. Sıcaklık -40'ın altına düşürülememeli, +40'ın üstüne de çıkarılamamalıdır.

Yalnız Başına MVP

- Karmaşık bir ekranda UI bileşenleri ve Presenter'lar kolayca **birbirlerine bağımlı** hale gelebilmektedirler
- Bu durum kodu **karmaşık** hale getirmektedir
- Ayrıca UI bileşenlerinin ve Presenter'ların **farklı senaryo'larda yeniden kullanılmaları**nı da engellemektedir

UI – Presenter Etkileşimi ve BusinessEvent'ler

View

UI event'leri uygulamaya özel business event'lere dönüştürülür

Presenter

UI state değişiklikleri UI'a Presenter tarafından yansıtılır

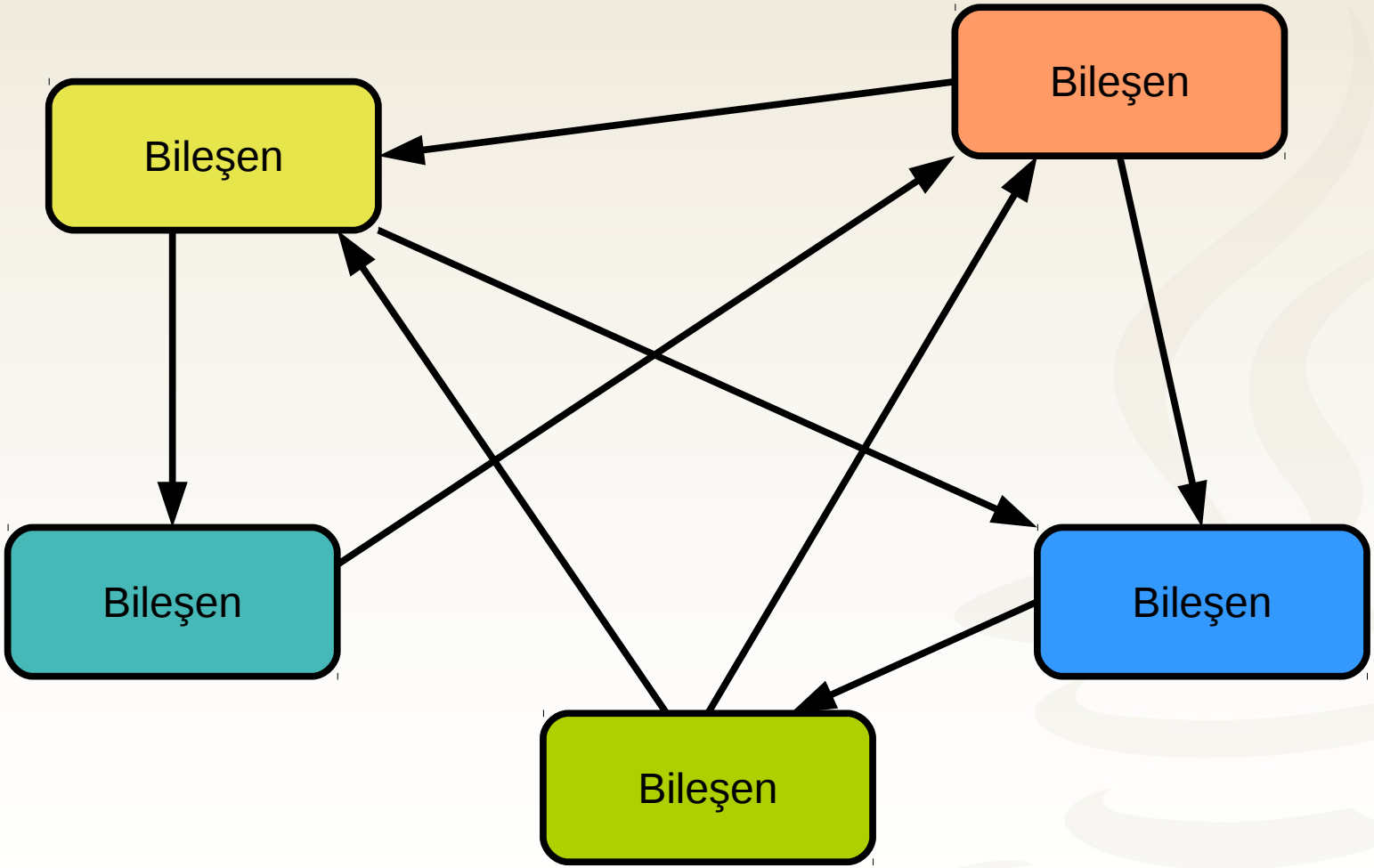
Presenter üzerinden iş mantığı koşturulur

Model üzerindeki değişiklikler Presenter tarafından takip edilir

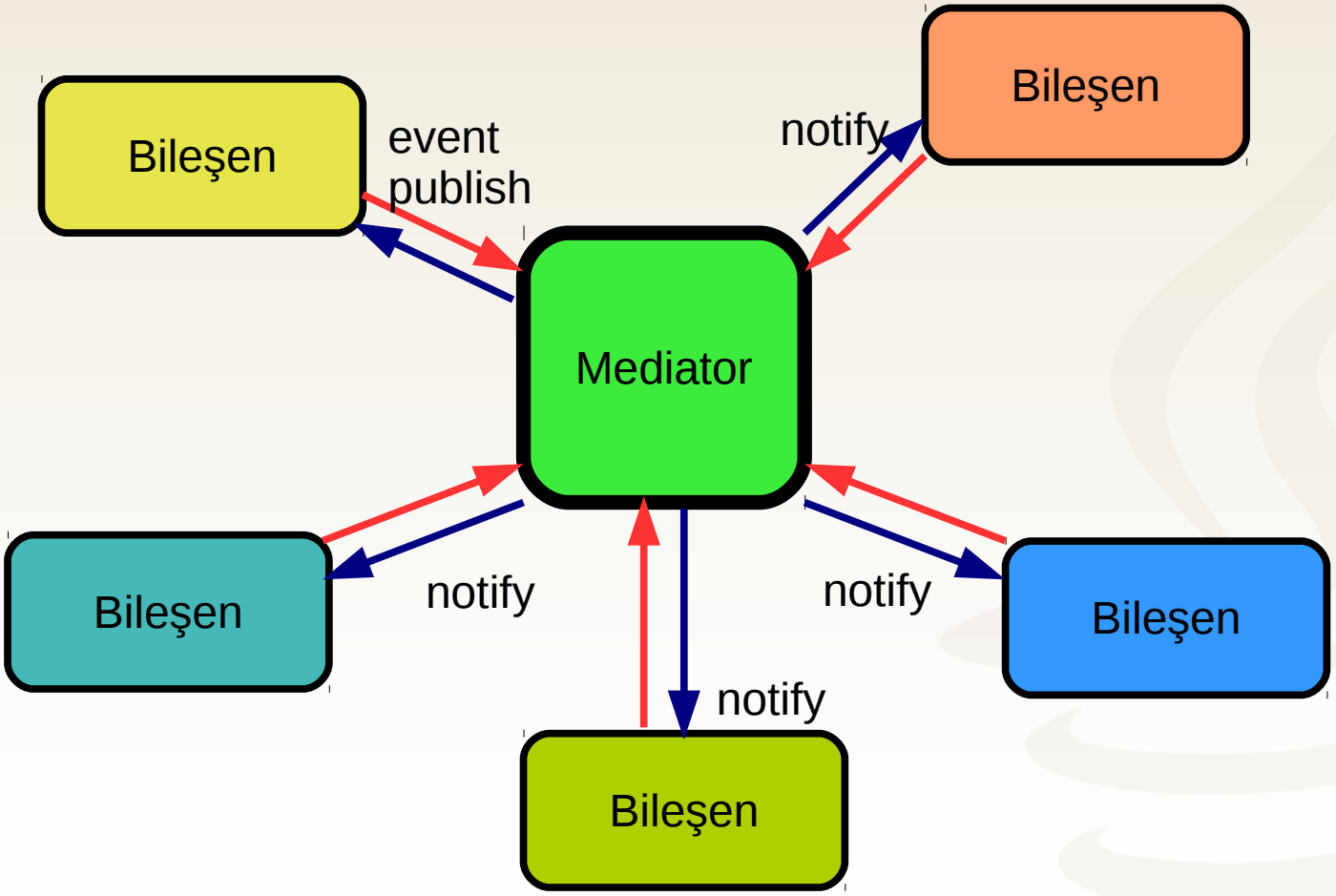
View ile Model arasında data binding'de mümkündür

Model

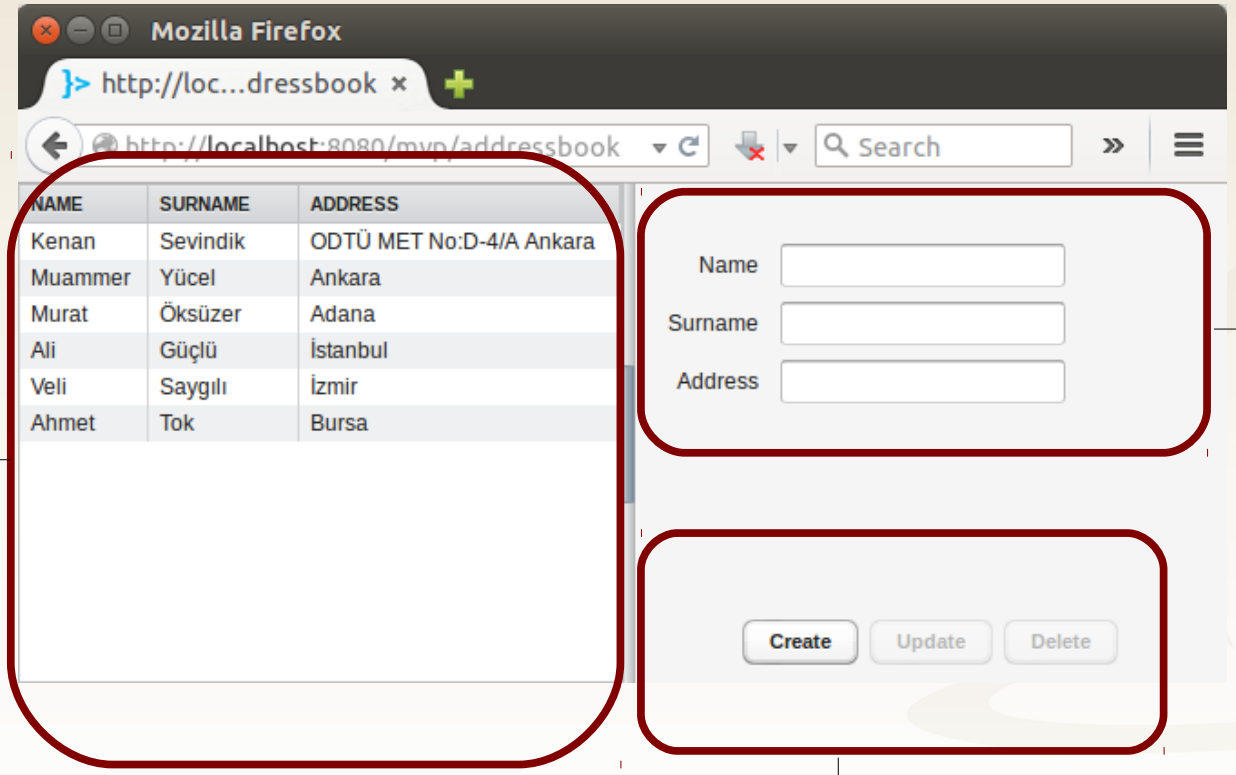
Mediator Öncesi Bileşenler Arası Etkileşim



Mediator Sonrası Bileşenler Arası Etkileşim



MVP ve Mediator



Address List View

Address Detail View

Address ToolBar View

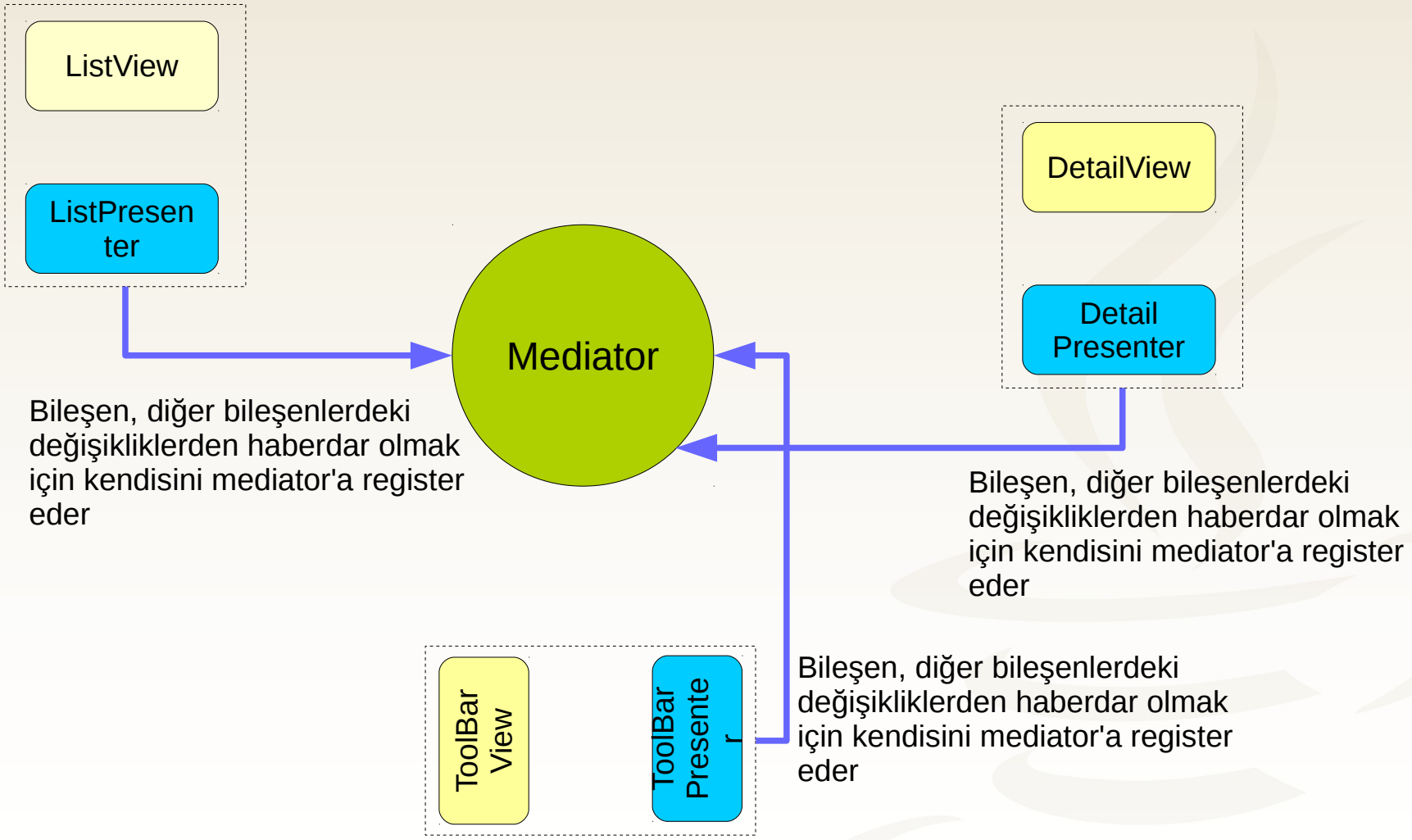
Mediator.java

```
public class Mediator {
   private Collection<Presenter> listeners = new
            ArrayList<Presenter>();
   public void addListener(Presenter listener) {
      listeners.add(listener);
   }

   public void removeListener(Presenter listener) {
      listeners.remove(listener);
   }

   public void publish(BusinessEvent event) {
      for(Presenter listener:listeners) {
         listener.handle(event);
      }
   }
}
```

Presenter.java

```
public interface Presenter {

    public void handle(BusinessEvent event);

}
```

Adım 1: Mediator Registration



Address List Presenter

```
public class AddressListPresenter implements Presenter
{

   private AddressListView view;

   public AddressListPresenter(AddressListView view,
      Mediator mediator) {

      this.view = view;
      mediator.addListener(this);
   }

   @Override
   public void handle(BusinessEvent event) {
      ...
   }
}
```

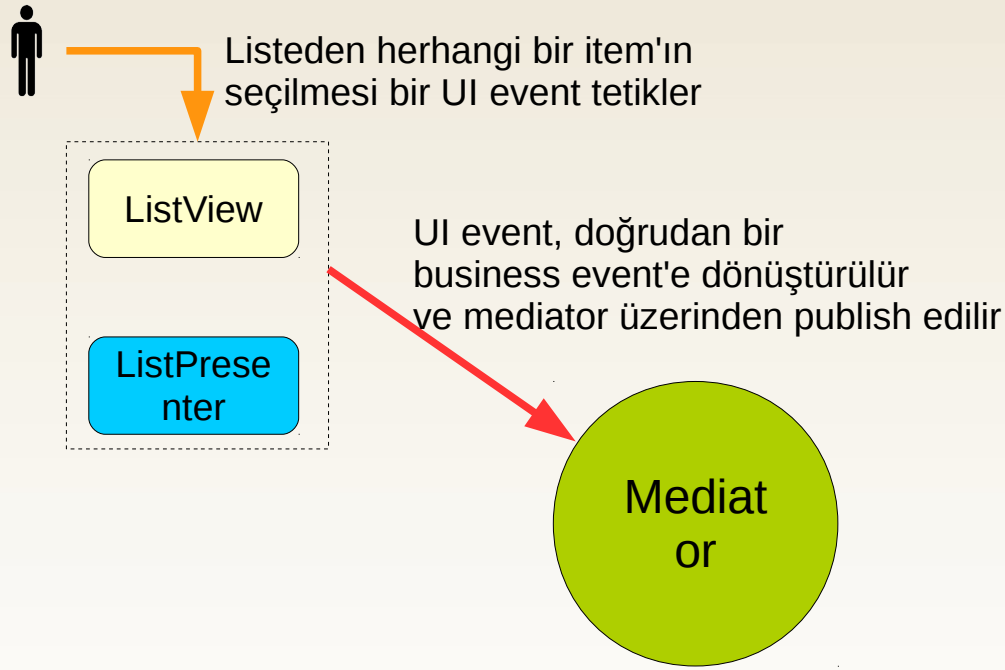
Address Detail Presenter

```
public class AddressDetailPresenter implements
Presenter {

   private AddressDetailView view;

   public AddressDetailPresenter(AddressDetailView
view,
         Mediator mediator) {

      this.view = view;
      mediator.addListener(this);
   }

   @Override
   public void handle(BusinessEvent event) {
      ...
   }
}
```

Address ToolBar Presenter

```
public class AddressToolBarPresenter implements
Presenter {

   private AddressToolBarView view;

   public AddressToolBarPresenter(AddressToolBarView
view,
         Mediator mediator) {

      this.view = view;
      mediator.addListener(this);
   }

   @Override
   public void handle(BusinessEvent event) {
      ...
   }
}
```

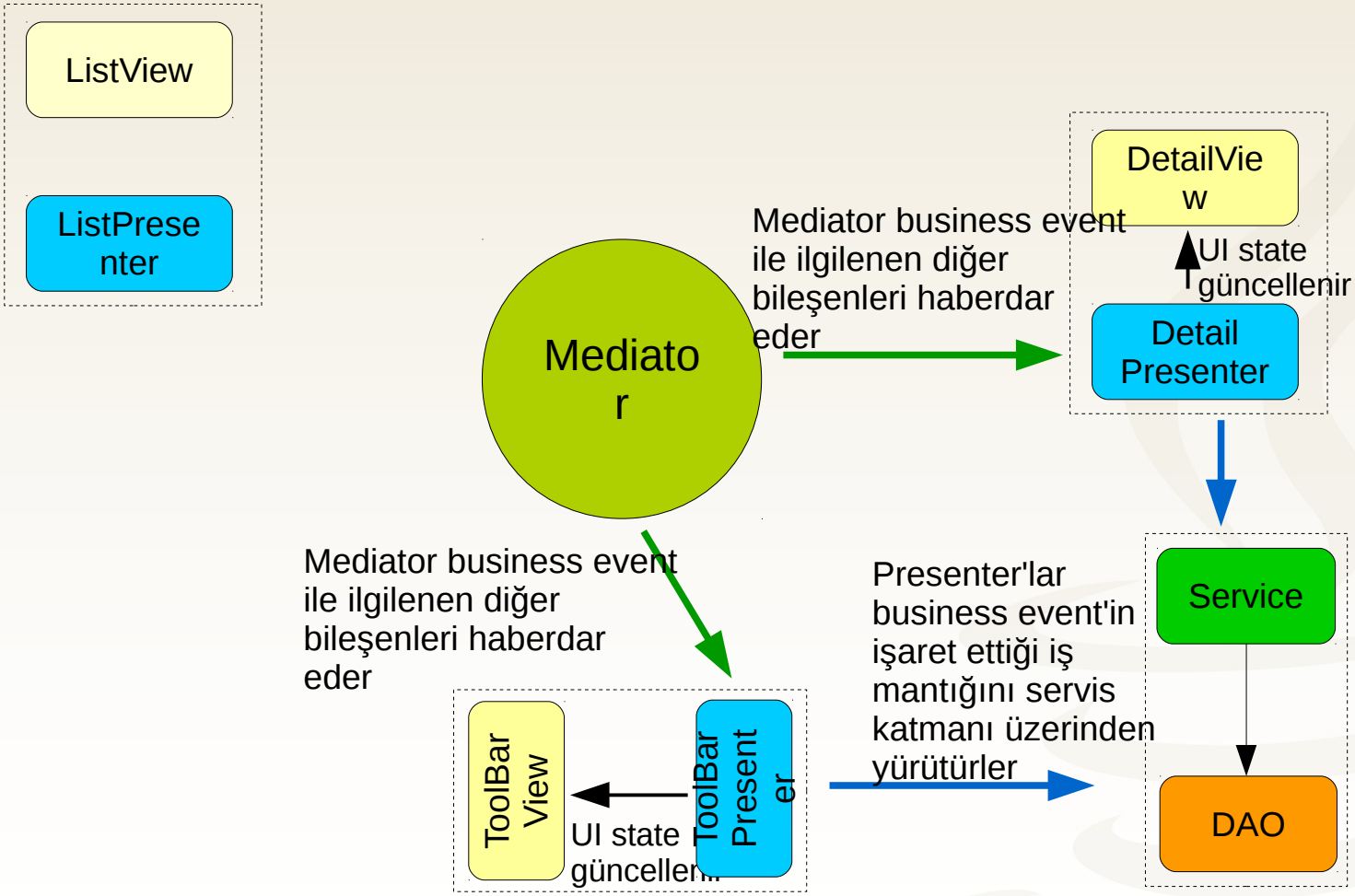
Adım 2: UI Interaction (Item Select)



Address List View

```
public class AddressListViewImpl implements
AddressListView, ValueChangeListener {

    public AddressListView(Mediator mediator) {
        this.mediator = mediator;
    }

    @Override
    public void valueChange(ValueChangeEvent event) {
        Address address = (Address) table.getValue();

        AddressSelectedEvent selectedEvent = new
                        AddressSelectedEvent(address);
        mediator.publish(selectedEvent);
    }

    ...
}
```

Adım 3: Event Notification (Address Selected)



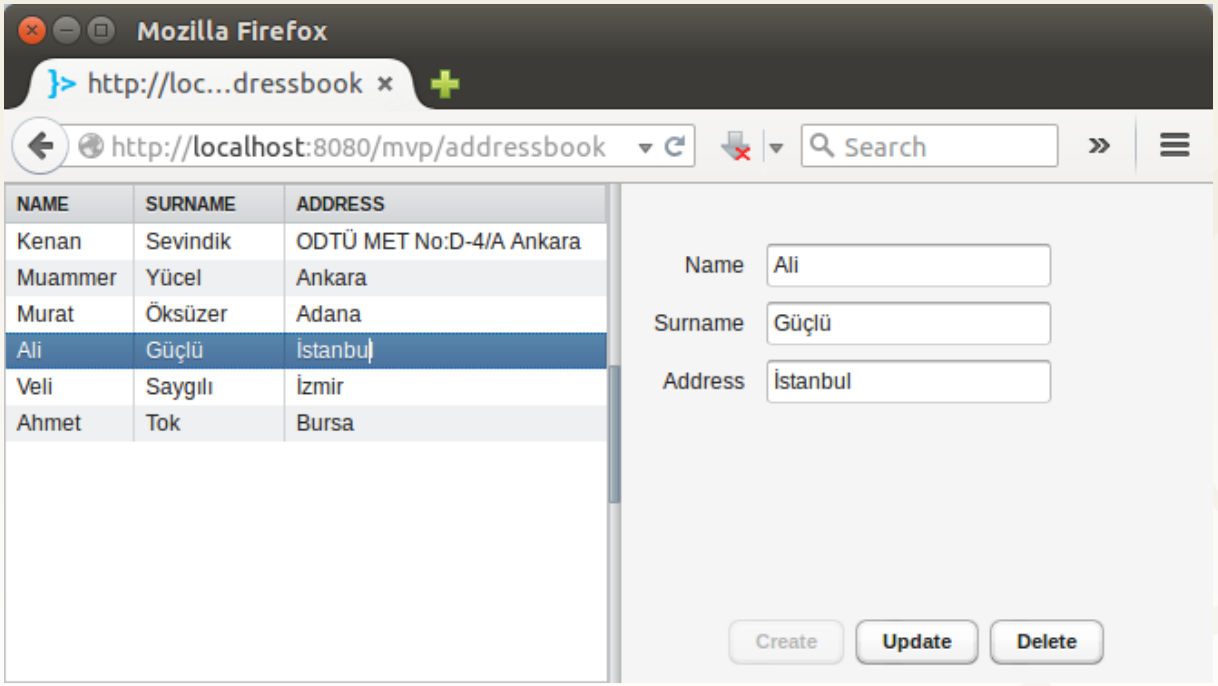
Address Detail Presenter

```
public class AddressDetailPresenter implements Presenter
{
   @Override
   public void handle(BusinessEvent event) {
      if(event instanceof AddressSelectedEvent) {

         AddressSelectedEvent selectedEvent =
                        (AddressSelectedEvent)event;

         Address address =
                        selectedEvent.getSelectedAddress();

         view.displayAddress(address);
      }
   }
   ...
}
```

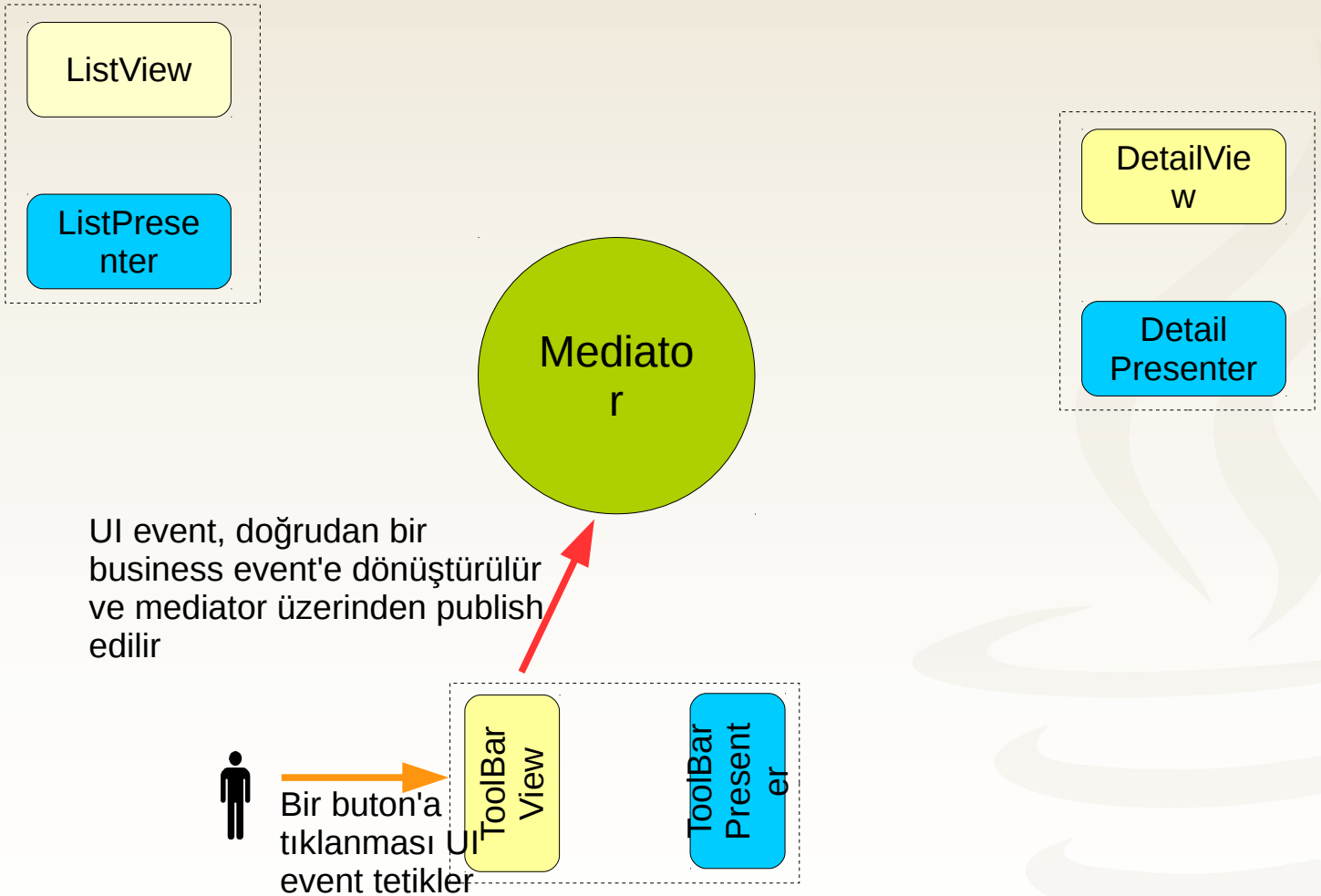
Address ToolBar Presenter

```
public class AddressToolBarPresenter implements Presenter {
    @Override
    public void handle(BusinessEvent event) {
       if(event instanceof AddressSelectedEvent)
{

            AddressSelectedEvent selectedEvent =
                        (AddressSelectedEvent)event;

            Address address =
                        selectedEvent.getSelectedAddress();

            view.switchToUpdateMode();
            view.setAddress(address);

        }
    }
    ...
}
```

Address Selected



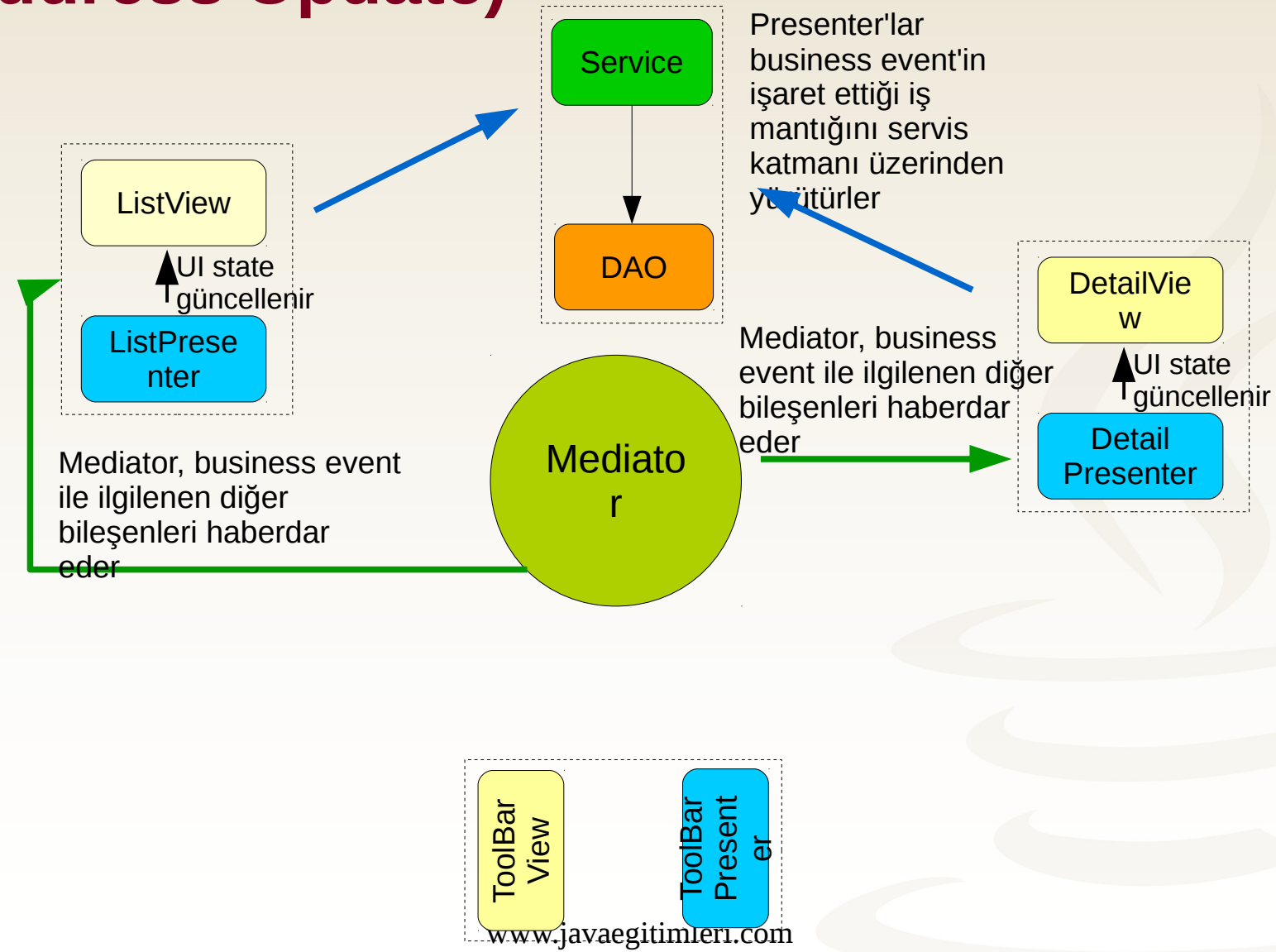
Adım 2: UI Interaction (Button Click)



Address ToolBar View

```
public class AddressToolBarViewImpl implements
AddressToolBarView, ClickListener {
   public AddressToolBarView(Mediator mediator) {
      this.mediator = mediator;
   }

   @Override
   public void buttonClick(ClickEvent event) {

      if(event.getButton() == updateButton) {
         AddressUpdateEvent updateEvent =
               new AddressUpdateEvent(address);
         mediator.publish(updateEvent);

      }
   }
   ...
}
```

Adım 3:Event Notification (Address Update)



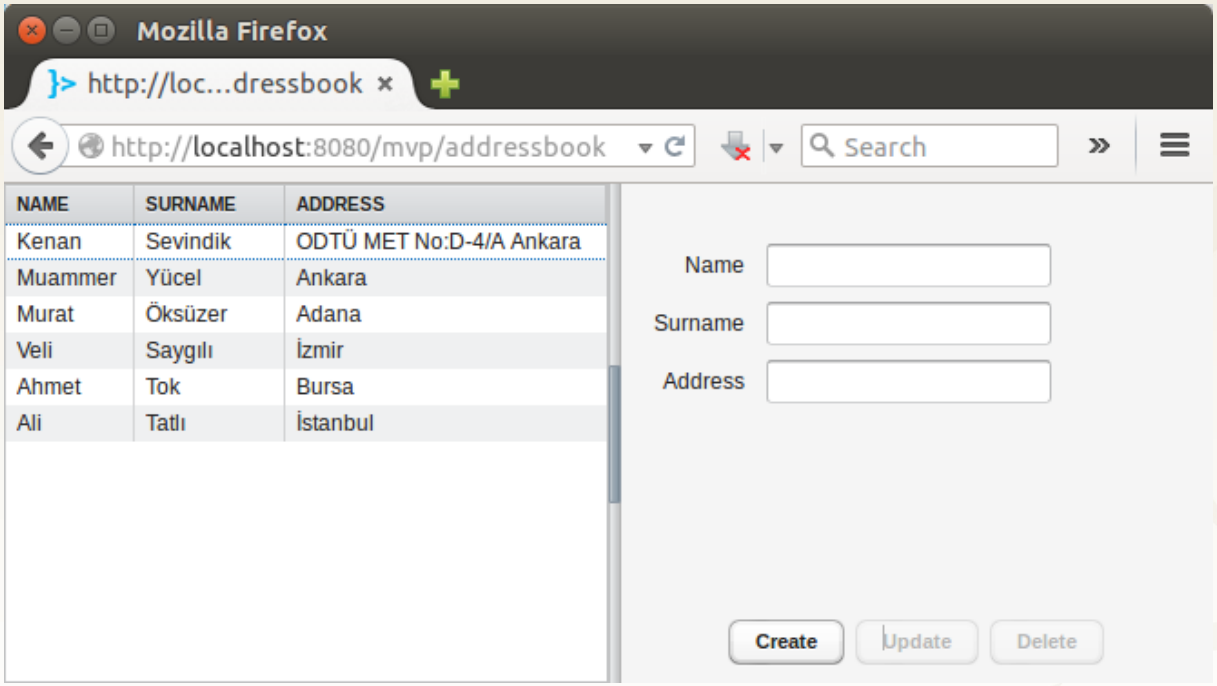
Address List Presenter

```
public class AddressListPresenter implements Presenter
{
   @Override
   public void handle(BusinessEvent event) {
      if(event instanceof AddressUpdateEvent) {

         AddressUpdateEvent updateEvent =
            (AddressUpdateEvent)event;

         Address address = updateEvent.getAddress();

         view.reloadAddress(address);

      }
   }
   ...
}
```

Address ToolBar Presenter

```
public class AddressToolBarPresenter implements Presenter {
    @Override
    public void handle(BusinessEvent event) {
        if(event instanceof AddressSelectedEvent) {
            AddressSelectedEvent selectedEvent =
                        (AddressSelectedEvent)event;
            Address address =
                        selectedEvent.getSelectedAddress();
            view.switchToUpdateMode();
            view.setAddress(address);

        }else if(event instanceof
AddressUpdateEvent) {
            view.switchToSelectionMode();
        }
    }
    ...
}
```

Address Updated



İletişim

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