



# **ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ Я MVC, MVP, MVVM**

# Обзор

- Схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента.
- Концепция разработана в 1979 году для языка Smalltalk

# Пассивная модель

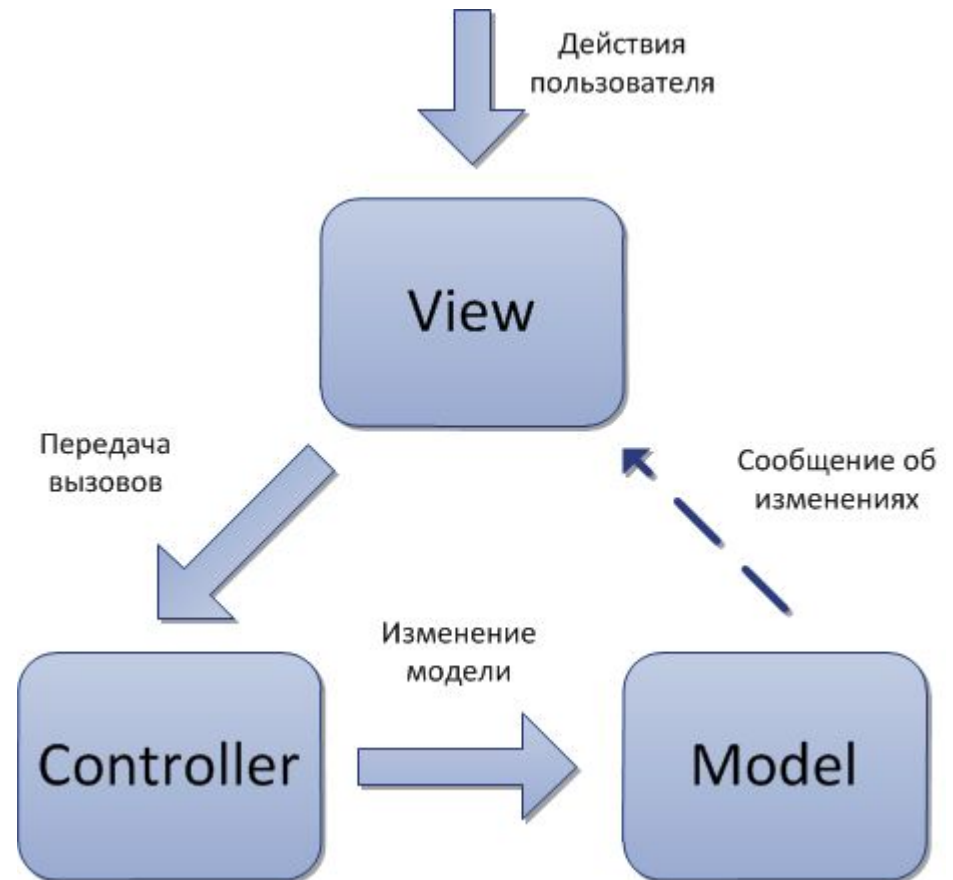
- Passive Model (пассивная модель) - Модель не имеет вообще никаких способов воздействовать на Представление или Контроллер и только используется ими в качестве источника данных для отображения. Все изменения модели отслеживаются Контроллером и он же отвечает за перерисовку Представления, если это необходимо.

# Активная модель MVC

Active Model (активная модель) - Модель имеет возможность оповестить Представление о том, что в ней произошли некие изменения, и Представление может эти изменения отобразить. Модель просто бросает сообщение, а Представления, которые заинтересованы в оповещении, подписываются на эти сообщения, что позволяет сохранить независимость Модели как от Контроллера так и от Представления, не нарушая тем самым основного свойства паттерна.

# MVC

- View представление, пользовательский интерфейс,
- Model - модель, бизнес логика
- Controller - контроллер, содержит логику на изменение модели при определенных действиях



# Реализация MVC в качестве GUI framework

- Cocoa framework
- Java Swing
- Qt4
- GTK+

# Реализация MVC как web framework

- Java – JSF, Oracle App Framework, Play Framework
- C# - ASP.NET MVC Framework, PureMVC
- Ruby – Ruby on Rails
- ...

# Brand.java

```
• package springmvc.model;  
•  
• public class Brand {  
•     private Long id;  
•     private String name;  
•     private String country;  
•  
•     public String getCountry() {  
•         return country;  
•     }  
•     public void setCountry(String country) {  
•         this.country = country;  
•     }  
• }
```



# Brand.java

```
.    public void setCountry(String country) {  
.        this.country = country;  
.    }  
.    public Long getId() {  
.        return id;  
.    }  
.    public void setId(Long id) {  
.        this.id = id;  
.    }  
.    public String getName() {  
.        return name;  
.    }  
.    public void setName(String name) {  
.        this.name = name;  
.    }  
.}
```

# Car.java

```
• package springmvc.model;  
•  
• import java.math.BigDecimal;  
•  
• public class Car {  
•     private Long id;  
•     private Brand brand;  
•     private String model;  
•     private BigDecimal price;  
•  
•     public Long getId() {  
•         return id;  
•     }  
• }
```

# Car.java

- `public void setId(Long id) {`
- `this.id = id;`
- `}`
- `public Brand getBrand() {`
- `return brand;`
- `}`
- `public void setBrand(Brand brand) {`
- `this.brand = brand;`
- `}`
- `public String getModel() {`
- `return model;`
- `}`
- `}`

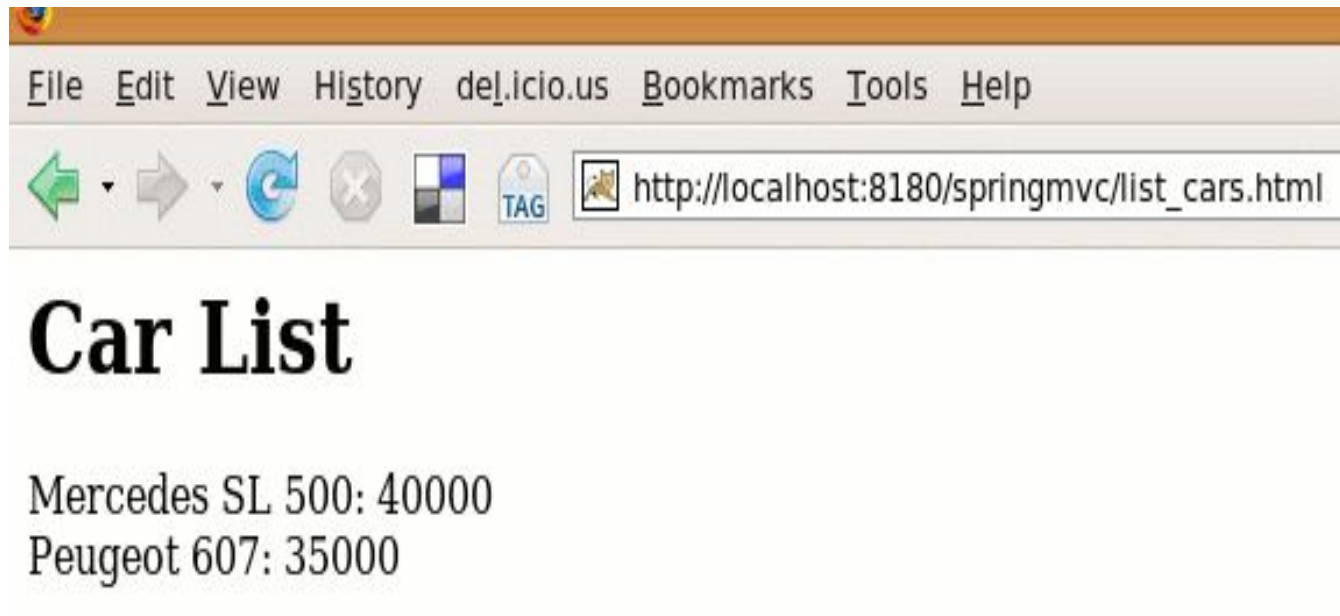
# CarListController.java

```
. package springmvc.web;
.
. import springmvc.service.CarManager;
.
. public class CarListController implements Controller {
.
.
.     public ModelAndView handleRequest(HttpServletRequest arg0, HttpServletResponse arg1) throws Exception {
.
.
.         CarManager carManager = new CarManager();
.
.
.         ModelAndView modelAndView = new ModelAndView("carList");
.         modelAndView.addObject("carList", carManager.getCarList());
.
.         return modelAndView;
.     }
. }
```

# CarList.jsp

- `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
- `<html>`
- `<body>`
- `<h1>Car List</h1>`
- `<c:forEach items="${carList}" var="car">`
- `${car.brand.name} ${car.model}: ${car.price}`
- `<br />`
- `</c:forEach>`
- `</body>`
- `</html>`

# Отображение страницы



# Model-view-presenter

- Разработан в начале 90-х годов в проекте Taligent.
- Паттерн был перенесен в JAVA Mike Potel.
- В 2006 году Microsoft начала включать MVP в свои примеры и документацию для программирования UI.

# Model-view-presenter

- Контроллер (Presenter) дает знать представлению об изменениях.
- Данный подход позволяет создавать абстракцию представления





# Реализации MVP

- Google Web Toolkit
- ASP.NET Web Forms Model-View-Presenter (MVP)
- Java Swing/AWT
- Silverlight
- ...

# Пример IView

```
. public interface IView
. {
.     /// <summary>
.     /// Вывод градусов Фаренгейта
.     /// </summary>
.     void SetFahrenheit(double value);
.
.     /// Ввод нового значения градусов
.     /// </summary>
.     double InputDegree { get; }
.
.     /// <summary>
.     /// Событие ввода значения по Фаренгейту
.     /// </summary>
.     event EventHandler<EventArgs> SetFahrenheit;
.
. }
```

# Пример Presenter

```
. public class Presenter
. {
.     private Model _model = new Model();
.     private IView _view;

.     /// <summary>
.     /// В конструктор передается конкретный экземпляр представления
.     /// и происходит подписка на все нужные события.
.     /// <summary>
.     public Presenter(IView view)
.     {
.         _view = view;
.         _view.SetCelsius += new EventHandler<EventArgs>(OnSetCelsius);
.         _view.SetFahrenheit += new EventHandler<EventArgs>(OnSetFahrenheit);
.         RefreshView();
.     }
```

# Пример Presenter

```
.    /// <summary>
.
.    /// Обработка события, установка нового значения градусов по Фаренгейту
.
.    /// </summary>
.
.    private void OnSetFahrenheit(object sender, EventArgs e)
.
.    {
.
.        _model.valueFahrenheit = _view.InputDegree;
.
.        RefreshView();
.
.    }
.
.    /// <summary>
.
.    /// Обновление Представления новыми значениями модели.
.
.    /// По сути Binding (привязка) значений модели к Представлению.
.
.    /// </summary>
.
.    private void RefreshView()
.
.    {
.
.        _view.SetCelsius(_model.valueCelsius);
.
.        _view.SetFahrenheit(_model.valueFahrenheit);
.
.    }
.
. }
```

# Пример View

```
.    /// <summary>
.
.    /// Обновление Представления новыми значениями модели.
.
.    /// По сути Binding (привязка) значений модели к Представлению.
.
.    /// </summary>
.
.    private void RefreshView()
.
.    {
.
.        _view.SetCelsius(_model.valueCelsius);
.
.        _view.SetFahrenheit(_model.valueFahrenheit);
.
.    }
.
. }
.
.    /// <summary>
.
.    /// Ввод нового значения градусов
.
.    /// </summary>
.
.    public double InputDegree
.
.    {
.
.        get { return Convert.ToDouble(_inputBox.Text); }
.
.    }
```

# Отличия от MVC

- Более слабая связь с моделью. Presenter отвечает за связывание модели и представления
- Более легкая организация тестирования, т.к. взаимодействие с представлением идет через интерфейс

# Литература

- MVC: XEROX PARC 1978-79 (1979) by Trygve Reenskaug, <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>
- Spring MVC Fast Tutorial <http://maestric.com/doc/java/spring/mvc>
- Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования.
- И. Бодягин. Model-View-Presenter и сопутствующие паттерны. <http://www.rsdn.ru/article/patterns/ModelViewPresenter.xml>

# Вопросы

- Паттерн проектирования MVVM
- Строение паттерна MVP
- Особенности MVC