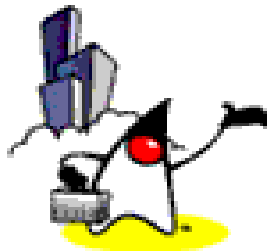




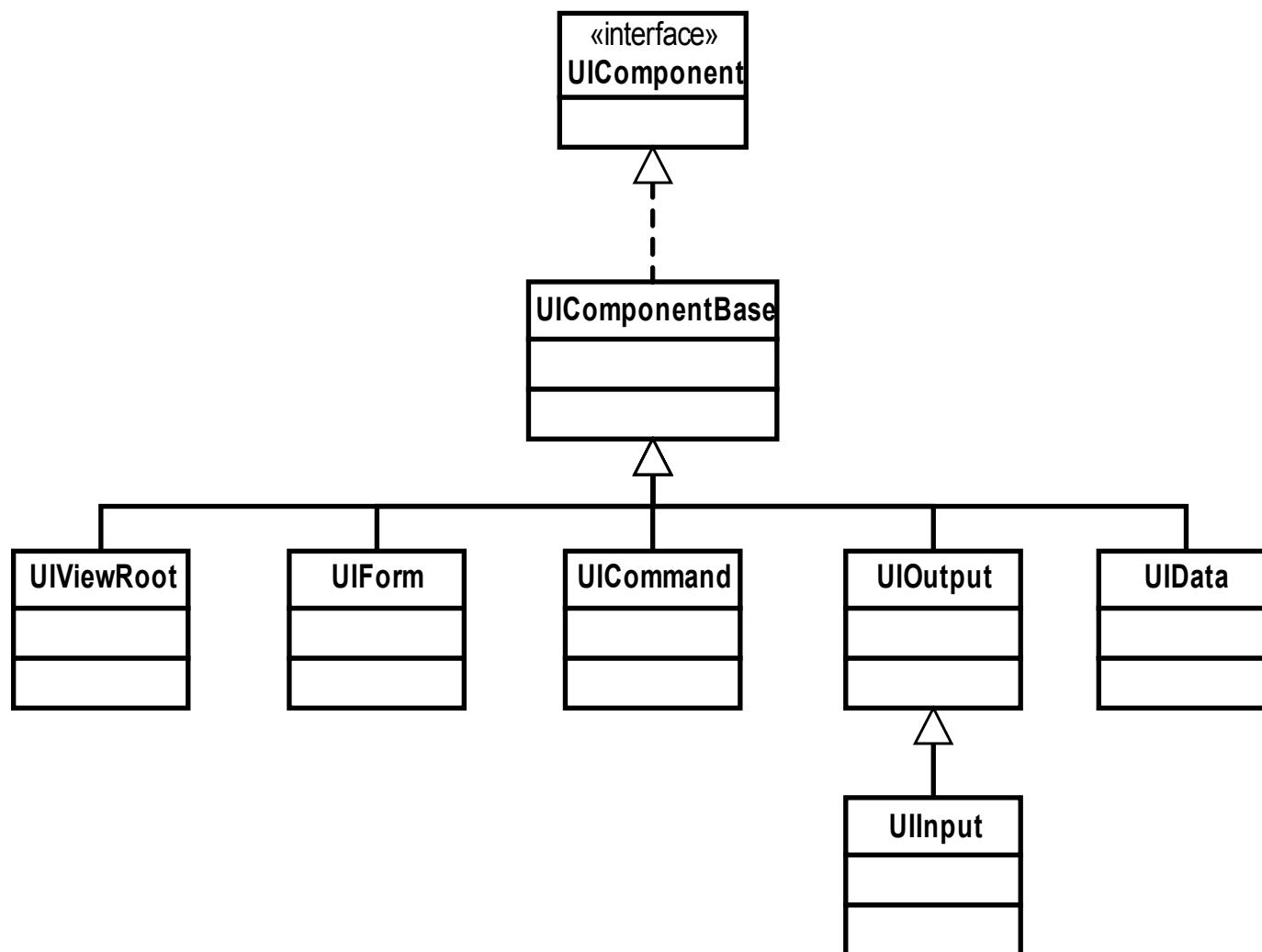
# Представление в JSF



# Основные концепции представления в JSF (1)

- **UIComponent**
  - Характеристики компонента, независящие от конкретного способа отображения
  - Базовый класс со стандартным поведением
- **Стандартные подклассы UIComponent:**
  - UICommand, UIForm, UIGraphic, UIInput, UIOutput, UIPanel, UISelectBoolean, UISelectMany, UISelectOne
- **FacesEvent**
  - Базовый класс событий

# Представление - Стандартные компоненты

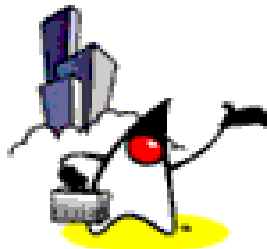


# Основные концепции представления в JSF (2)

- **Renderer**
  - Преобразует компоненты в элементы конкретного языка разметки (и наоборот)
  - Поддерживает атрибуты, специфичные для отображения
  - Может поддерживать более одного типа компонентов
- **RenderKit**
  - Библиотека Renderer'ов
  - Может расширяться в процессе выполнения
  - Basic HTML RenderKit входит в спецификацию



# **Компонентная модель интерфейса пользователя**

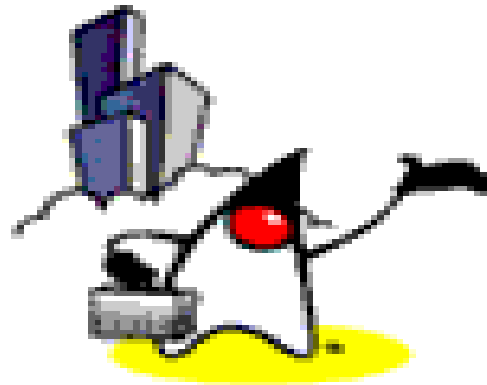


# Что такое компонент?

- Хорошо известная идиома проектирования интерфейса пользователя
- Конфигурируемый и повторно используемый элемент, обеспечивающий интерфейс пользователя в JSF-приложениях
- Может быть простым (кнопка) или сложным (таблица, которая сама содержит множество компонентов)
- Доступен на JSP-страницах с помощью особых действий JSF

# Компонентная модель интерфейса пользователя в JSF

- Набор **классов**, инкапсулирующих состояние и поведение компонентов UI
- **Модель отображения**, определяющая отображение компонентов для разных средств вывода
- **Модель обработки событий**
- **Модель конвертации**, определяющая подключение конвертеров данных к компоненту
- **Модель валидации**, определяющая способ регистрации валидаторов для компонента



# UI Component Model:

## **Классы компонентов UI**



# Классы компонентов UI

- Классы компонентов UI выполняют все функции компонента пользовательского интерфейса
  - Получение значений из формы ввода (декодирование)
  - Хранение состояния компонента
  - Поддержание ссылок на объекты модели
  - Вызов обработчиков событий
  - Отображение – создание элементов разметки (кодирование)

# Классы компонентов UI

- Реализация JSF предоставляет набор классов компонентов UI
  - Разработчики могут расширять эти классы для создания специфических компонентов UI
- Все классы компонентов UI в JSF расширяют вспомогательный класс **UIComponentBase**
  - UIComponentBase определяет состояние и поведение компонента по умолчанию

# Как авторы страниц используют классы компонентов UI?

- Большинство авторов страниц и разработчиков приложений не используют эти классы напрямую
  - Компоненты помещаются на страницу с помощью **соответствующих действий JSP**
- Большинство компонентов можно отобразить различными способами
  - Например, компонент `UICommand` можно отобразить как кнопку или как гиперссылку, для чего используются различные действия JSP

# Встроенные классы компонентов UI (1)

- **UIForm:**
  - Содержит группу управляющих элементов, которые отправляют данные в приложение. Этот компонент аналогичен тегу `<form>` в HTML.
- **UICommand:**
  - Представляет управляющий элемент, который при активации начинает какое-либо действие.
- **UIGraphic:**
  - Выводит изображение.

# Встроенные классы компонентов UI (2)

- **UIOutput:**
  - Выводит данные на страницу
- **UIInput:**
  - Обеспечивает ввод данных от пользователя
  - Является подклассом UIOutput
- **UIPanel:**
  - Выводит таблицу
- **UIParameter:**
  - Представляет подстановочные параметры

# Встроенные классы компонентов UI (3)

- `UISelectItems` и `UISelectItem`:
  - Набор объектов и конкретный объект в наборе.
- `UISelectBoolean`:
  - Позволяет выбрать логическое значение. Подкласс `UIInput`.
- `UISelectOne`:
  - Позволяет выбрать один объект из набора объектов. Подкласс `UIInput`.
- `UISelectMany`:
  - Позволяет выбрать несколько объектов из набора объектов. Подкласс `UIInput`.

# Интерфейс ValueHolder

- Интерфейс для тех компонентов, которые хранят некоторое локальное значение, а также имеют доступ к модели через EL-выражение
- Поддерживает преобразование между классом String и типом данных, используемым в модели
- Реализуется классом UIOutput

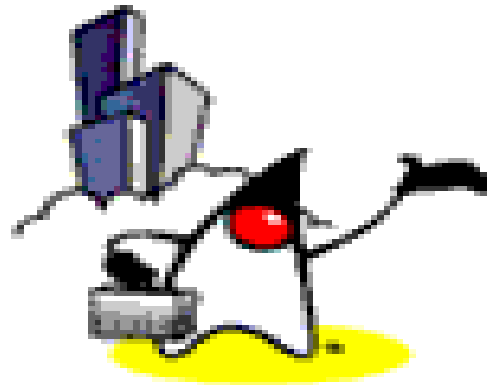
# Интерфейс `EditableValueHolder`

- Расширяет *ValueHolder*
- Для компонентов, которые поддерживают редактирование своего значения
  - Поддержка обработчиков событий изменения значения `ValueChangeEvent`
  - Поддержка валидации (`Validator`)
- Реализуется классом `UIInput`



# UIViewRoot

- *UIViewRoot* – это *UIComponent*, который представляет корень дерева компонентов
- Дерево компонентов создается для каждого отдельного представления (страницы)
- К нему прикрепляются обработчики событий цикла обработки запроса *PhaseListeners*



# UI Component Model: Модель отображения компонентов

# Отображение компонентов

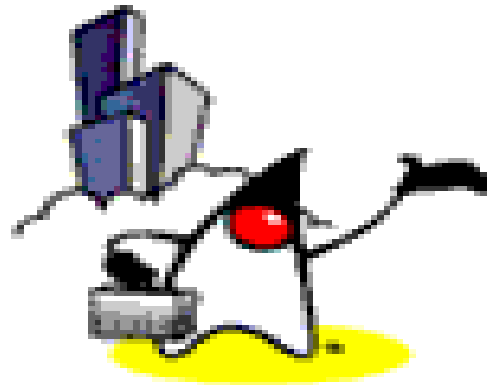
- Отображение выполняется **набором рендереров**, а не классами компонентов
- Авторы страниц и разработчики могут изменить внешний вид компонента на странице, выбирая **действие JSP**, представляющее комбинацию **компонента/рендерер**
  - `<h:commandButton>` (компонент `UICommand`, отображаемый в виде кнопки)
  - `<h:commandLink>` (компонент `UICommand`, отображаемый в виде ссылки)

# RenderKit – набор рендереров

- Определяет соответствие между классами компонентов и действиями JSP, подходящими для конкретного вида клиентов
- Реализация JSF включает встроенный RenderKit для вывода HTML-страниц
- Для каждого поддерживаемого компонента UI в RenderKit'e определен набор рендереров (объектов **Renderer**)

# Рендерер (объект типа `Renderer`)

- Определяет конкретный способ отображения компонента в тот формат, для которого определен `RenderKit`
- Пример
  - У компонента `UISelectOne` есть три различных рендерера
    - Набор переключателей (радио-кнопок)
    - Выпадающий список
    - Список

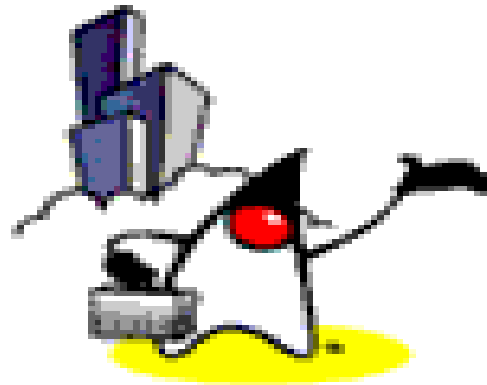


# UI Component Model:

## Модель обработки событий

# Модель обработки событий

- Аналогична модели событий, принятой в JavaBeans
  - Интерфейсы обработчиков и классы событий
  - Объект события определяет компонент, вызвавший событие, и содержит информацию о событии
  - Чтобы получать уведомления о событиях некоторого компонента, приложение должно реализовать обработчик соответствующего типа и зарегистрировать его в компоненте
  - Событие запускается, когда пользователь активирует компонент (например, нажимает кнопку)



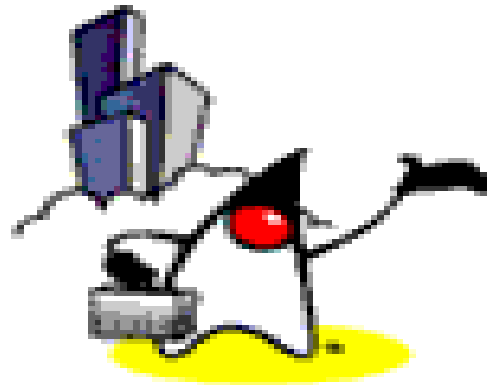
# UI Component Model:

## Модель конвертации



# Модель конвертации

- Компонент может быть связан с объектом модели
- Данные компонента выглядят по-разному:
  - С точки зрения модели
  - С точки зрения представления
- Данные компонента можно конвертировать между тем и другим видом
  - Обычно эта конвертация выполняется автоматически **рендерером компонента**
  - Поддерживается конвертация по особым правилам, которая реализуется **Converter'ом**



# UI Component Model:

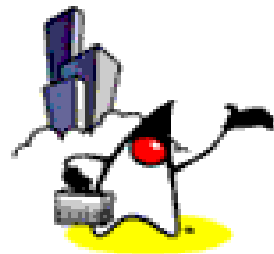
## Модель валидации

# Модель валидации

- Аналогично модели конвертации, модель валидации определяет набор стандартных классов для выполнения типичных проверок данных
- Программист может реализовать собственный валидатор – класс, реализующий интерфейс **Validator**



# **Использование библиотек действий JSF**



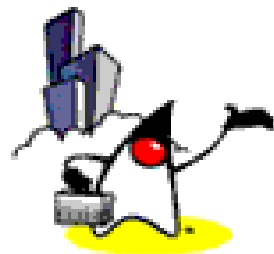
# Две библиотеки действий

- **jsf\_core (основные действия)**
  - Не зависит от конкретной технологии отображения
- **html\_basic (базовые действия для HTML)**
  - Определяет действия для общих компонентов веб-интерфейса (HTML-страниц)
- В JSP-странице требуется объявить эти библиотеки перед использованием

```
<%@ taglib uri="http://java.sun.com/jsf/html/" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core/" prefix="f" %>
```



# **Использование основных действий**



# Виды основных действий

- Обработка событий
- Добавление атрибутов
- Конвертация данных
- Фасет
- Локализация
- Подстановка параметров
- Представление объектов в списке выбора
- Валидация данных
- Буквальный вывод
- Контейнеры компонентов

# Действия и атрибуты для обработки событий

- `<f:actionListener>` и атрибут `actionListener`
  - Регистрирует обработчик действия компонента (например, нажатия кнопки)
- `<f:valueChangeListener>` и атрибут `valueChangeListener`
  - Регистрирует обработчик изменения значения компонента
- `<f:phaseListener>`
  - Регистрирует обработчик событий цикла обработки запроса



# Примеры обработки событий

- Обработка нажатия на кнопку:

```
<h:commandButton id="NAmerica" action="storeFront"
    value="#{bundle.english}"
    actionListener="#{carstore.chooseLocaleFromLink}">
</h:commandButton>
```

- Обработка ввода текста:

```
<h:inputText id="firstName" value="#{customer.firstName}"
    required="true">
    <f:valueChangeListener type="carstore.FirstNameChanged" />
</h:inputText>
```

# Действия конвертации данных

- `<f:converter>`
  - Регистрирует произвольный конвертер для родительского компонента
- `<f:convertDateTime>`
  - Регистрирует конвертер даты/времени для родительского компонента
- `<f:convertNumber>`
  - Регистрирует конвертер чисел для родительского компонента

# Фасет

- `<f:facet>`
  - Обозначает вложенный компонент, у которого есть особая связь с родительским компонентом
    - Например, у столбца таблицы есть фасет “заголовок”

# Действие подстановки параметра

- `<f:parameter>`
  - Подставляет параметры в сообщение (объект `MessageFormat`) или в URL запроса

# Представление объектов в списке выбора

- `<f:selectItem>`
  - Представляет один объект в списке для компонентов `UISelectOne` и `UISelectMany`
- `<f:selectItems>`
  - Представляет набор объектов для компонентов `UISelectOne` и `UISelectMany`

# Действия валидации данных

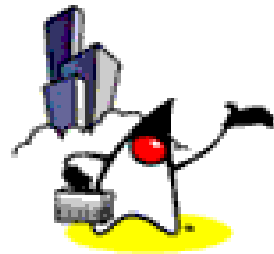
- `<f:validateDoubleRange>`
  - Регистрирует `DoubleRangeValidator` (диапазон вещественных чисел) для компонента
- `<f:validateLongRange>`
  - Регистрирует `LongRangeValidator` (диапазон целых чисел)
- `<f:validateLength>`
  - Регистрирует `LengthValidator` (длина значения)
- `<f:validator>`
  - Регистрирует произвольный валидатор (объект `Validator`) для компонента

# Контейнеры компонентов

- Действие `<f:view>` представляет корень дерева компонентов – объект `UIViewRoot`
- Все компоненты на странице должны размещаться **внутри** действия `<f:view>`  
    `<f:view>`  
        ... другие действия JSF и прочее содержимое ...  
    `</f:view>`
- Действие `<f:subview>` должно содержать все действия JSF на странице, которая включается в другую JSF-страницу



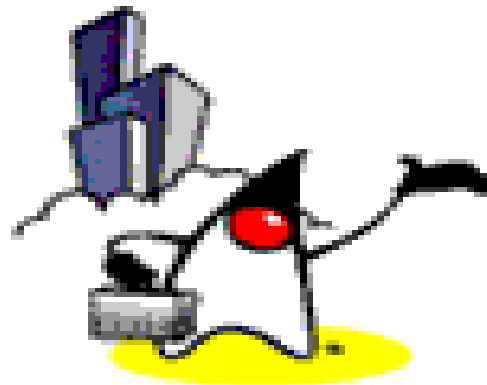
# **Использование базовых действий для HTML**





# Базовые действия для HTML

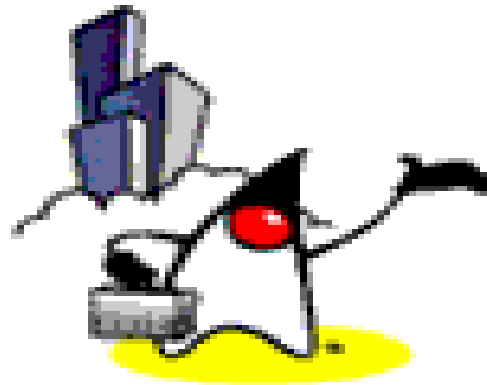
- Управляют отображением данных и вводом данных от пользователя
- Название каждого действия состоит из
  - Функции компонента, определенной в классе, реализующем `UIComponent`
  - Способа отображения компонента, определяемого рендерером
- Общие атрибуты
  - **id**: идентификатор компонента
  - **value**: источник данных (отложенное EL-выражение), сопоставляемый со значением компонента
  - **binding**: свойство `backing bean`, представляющее экземпляр компонента на странице



**UIForm и <h:form>**

# UIForm и действие <h:form>

- Компонент UIForm
  - Форма ввода, дочерние компоненты которой представляют данные, отображаемые или вводимые пользователем
- Заключает в себя все управляющие элементы
- Включает HTML-разметку для размещения управляющих элементов на странице
  - Действие <h:form> не выполняет размещение компонентов



**UICommand и  
<h:commandButton>**

# UICommand и действие <h:commandButton>

- Компонент UICommand выполняет действие (отправляет запрос) при активации пользователем
- Отображается как кнопка (действие <h:commandButton>) или ссылка (действие <h:commandLink>)

# UICommand и действие <h:commandButton>

- Дополнительные атрибуты
  - **action**:
    - строка результата или отложенное EL-выражение, указывающее на метод, возвращающий строку результата
    - Строка результата используется контроллером для определения следующей отображаемой страницы
  - **actionListener**:
    - отложенное EL-выражение, указывающее на метод, обрабатывающий событие типа *ActionEvent*, отправляемое компонентом *UICommand*

# Пример 1: <h:commandButton> на странице carDetail.jsp

```
<h:commandButton action="#{carstore.buyCurrentCar}"  
                 value="Buy" />
```

- Атрибут **action**
  - указывает на метод в классе managed bean **CarStore**, который выполняет обработку действия и возвращает результат
  - результат обрабатывается контроллером согласно правилам навигации, определенным в конфигурационном файле

# Пример 1: метод buyCurrentCar() класса CarStore

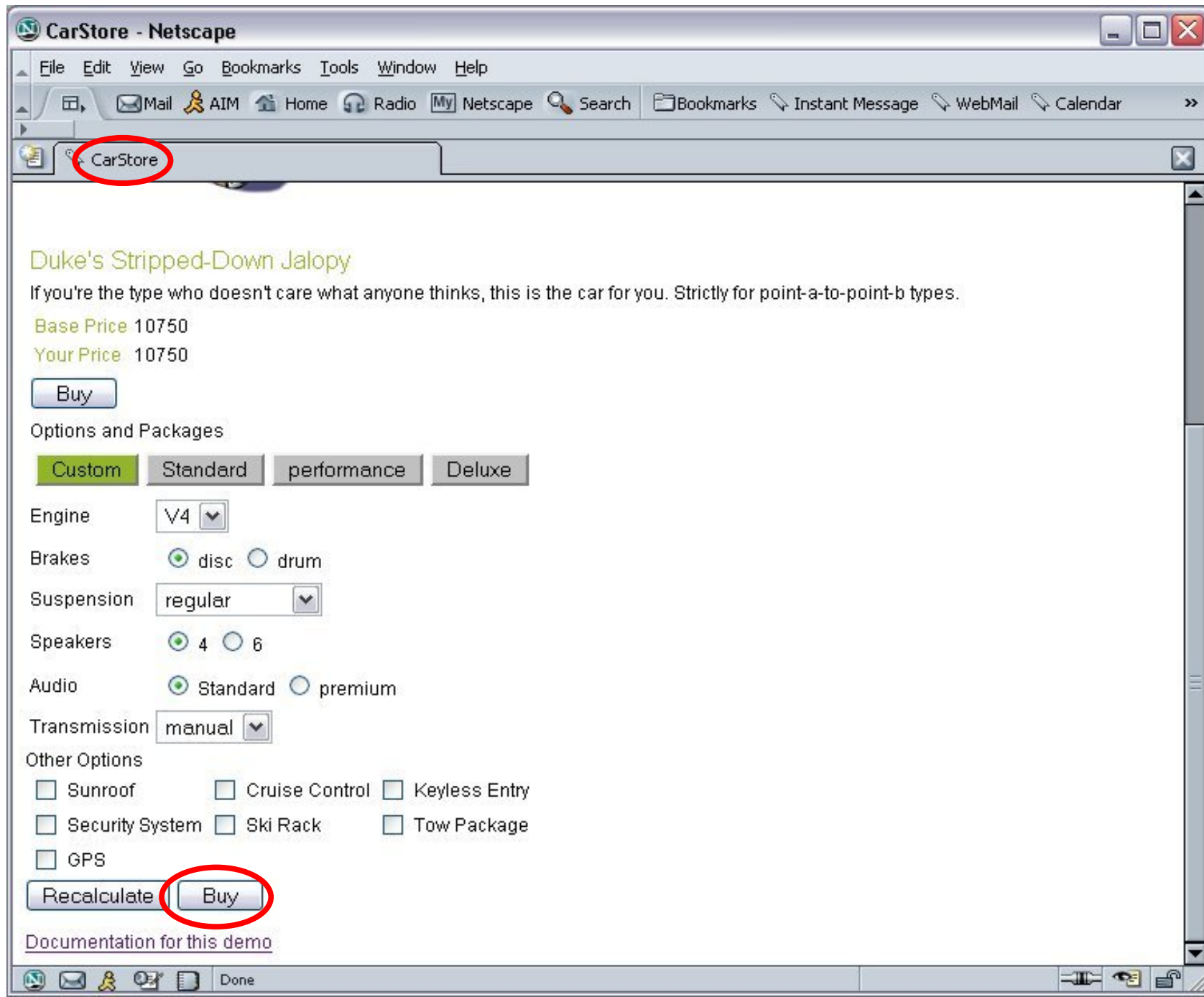
```
public class CarStore {  
    ...  
    public String buyCurrentCar() {  
        getCurrentModel().getCurrentPrice();  
        return "confirmChoices";  
    }  
    ...  
}
```



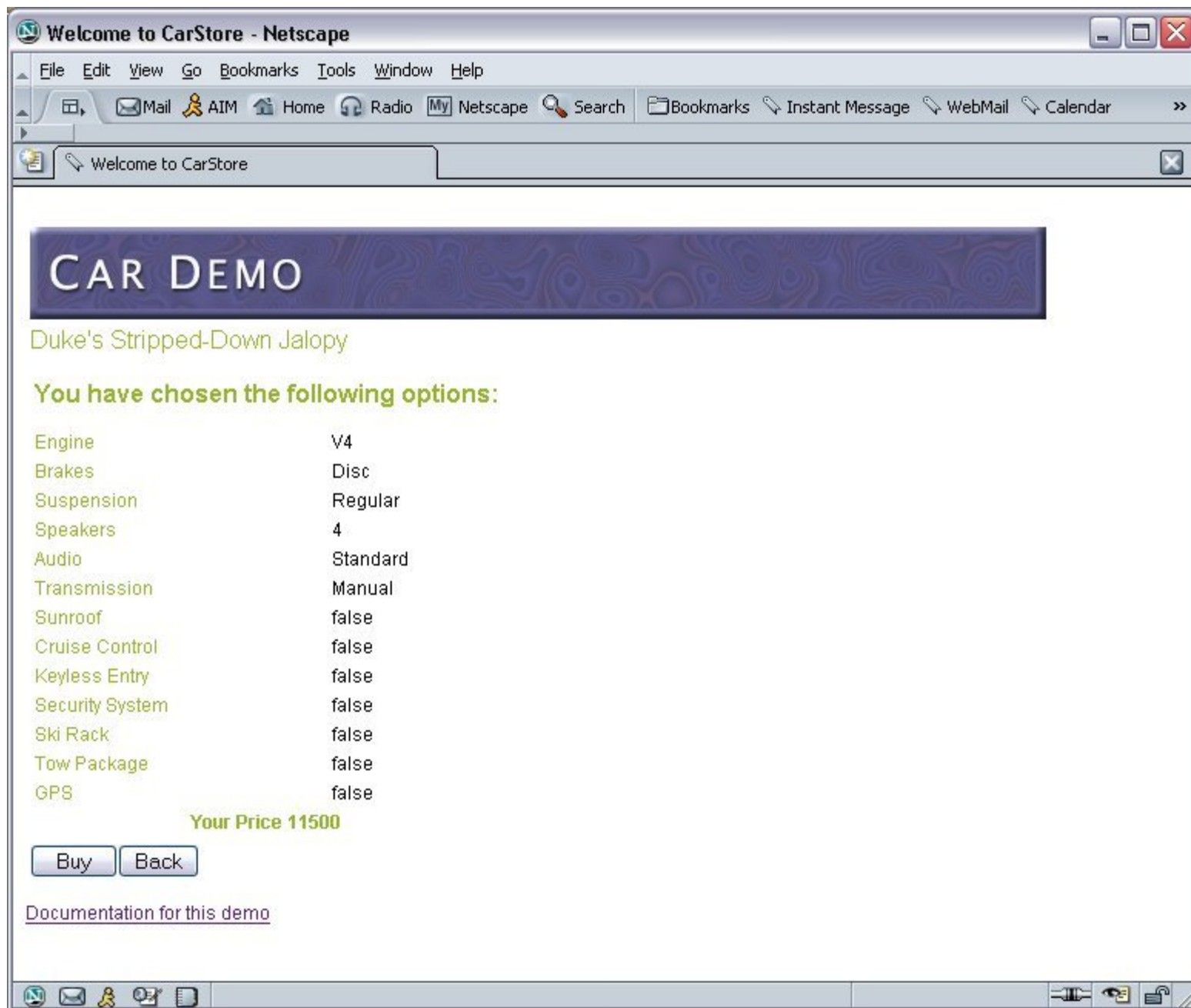
# Пример 1: Правило навигации для “confirmChoices” в faces-config.xml

```
<navigation-rule>  
  <from-view-id>/carDetail.jsp</from-view-id>  
  <navigation-case>  
    <description>  
      Любое действие, которое возвращает "confirmChoices"  
      на странице carDetail.jsp, должно вызывать переход  
      на страницу confirmChoices.jsp  
    </description>  
    <from-outcome>confirmChoices</from-outcome>  
    <to-view-id>/confirmChoices.jsp</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

# carDetail.jsp



# confirmChoices.jsp



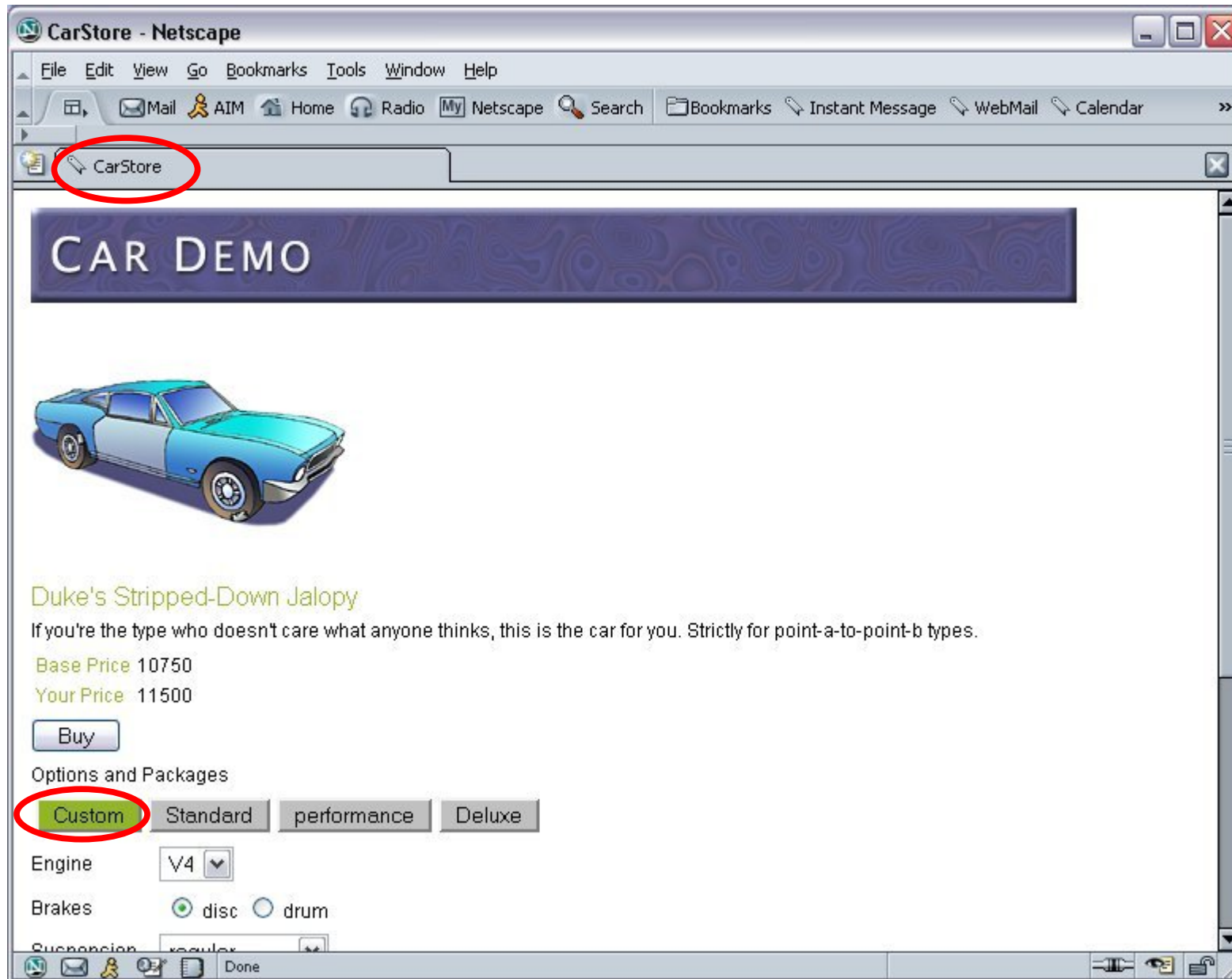
## Пример 2: <h:commandButton> на странице optionsPanel.jsp

```
<h:commandButton  
  id="Custom"  
  value="Custom"  
  styleClass="#{carstore.customizers.Custom.buttonStyle}"  
  actionListener="#{carstore.choosePackage}" />
```

## Пример 2: метод `choosePackage()` класса `CarStore`

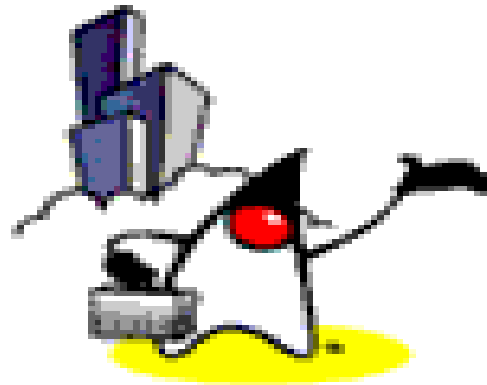
```
public class CarStore {  
    ...  
    public void choosePackage(ActionEvent event) {  
        String packageName = event.getComponent().getId();  
        choosePackage(packageName);  
    }  
  
    public void choosePackage(String packageName) {  
        // Выполнение бизнес-логики  
    }  
    ...  
}
```

# optionsPanel.jsp



# Сравнение атрибутов action и ActionListener

- action
  - Предназначен для бизнес-логики
  - Участвует в навигации
- ActionListener
  - Принимает объект ActionEvent в качестве параметра
  - Обрабатывает логику представления (логику интерфейса пользователя)
  - Не участвует в навигации



# UIInput и UIOutput



# Компоненты *UInput* и *UOutput*

- Компонент *UInput* отображает значение и позволяет пользователю изменить его
  - Например, **текстовое поле**
- Компонент *UOutput* отображает данные, которые не могут быть изменены
  - Например, **метка**
- При этом может выполняться конвертация данных
- И *UInput*, и *UOutput* можно отобразить несколькими разными способами

# Действия для UInput

- `<h:inputText>`
  - Однострочное текстовое поле
- `<h:inputTextarea>`
  - Многострочное текстовое поле
- `<h:inputHidden>`
  - Скрытое поле ввода
- `<h:inputSecret>`
  - Однострочное текстовое поле для ввода секретной информации (любые символы отображаются как \*)

# Действия для UIOutput

- `<h:outputText>`
  - Отображает текстовую строку
- `<h:outputMessage>`
  - Отображает локализованное сообщение
- `<h:outputLabel>`
  - Отображает метку для указанного поля ввода
- `<h:outputLink>`
  - Отображает тег `<a href >`, который позволяет перейти на другую страницу без генерации события `ActionEvent`

# Атрибуты действий <h:inputText> и <h:outputText>

- id
- value
- converter
- validator
  - отложенное EL-выражение, указывающее на метод класса backing bean, выполняющий проверку данных компонента
- valueChangeListener
  - отложенное EL-выражение, указывающее на метод класса backing bean, обрабатывающий событие ввода данных в этот компонент

# Пример: `<h:inputText>` на странице `customerInfo.jsp`

- `<h:inputText value="#{customer.lastName}" />`

Customer Details - Netscape

File Edit View Go Bookmarks Tools Window Help

Mail AIM Home Radio My Netscape Search Bookmarks Instant Message WebMail Calendar

Customer Details

## CAR DEMO

Please fill in your name and address.

Title

**First Name**

Middle Initial

Last Name

Mailing Address


City

State

Zip Code

Credit Card Number

Expiry Date

 Thanks for stopping by!

Done

# customerInfo.jsp

Customer Details - Netscape

File Edit View Go Bookmarks Tools Window Help

Mail AIM Home Radio My Netscape Search Bookmarks Instant Message WebMail Calendar


Customer Details

## CAR DEMO

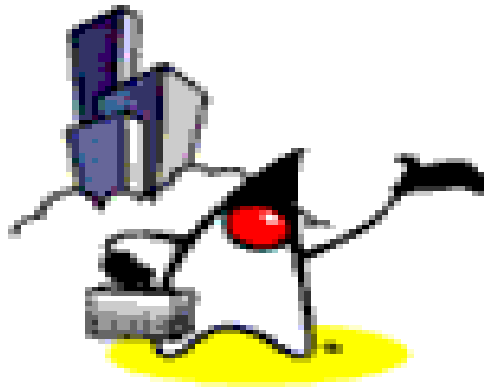
Please fill in your name and address.

|                    |                  |
|--------------------|------------------|
| Title              | Mr. ▾            |
| First Name         | Sang             |
| Middle Initial     | C                |
| Last Name          | Shi n            |
| Mailing Address    | 123 Wonderland   |
| City               | Wondercity       |
| State              | AL ▾             |
| Zip Code           | 12345            |
| Credit Card Number | 1234567812345678 |
| Expiry Date        | 01 ▾ 2005 ▾      |

Finish

 Thanks for stopping by!

Done



**UIPanel**  
**<h:panelGrid> и**  
**<h:panelGroup>**

# Компонент UIPanel

- Используется как менеджер компоновки для дочерних компонентов
- Должен иметь заранее определенное количество строк



# Действия для UIPanel

- `<h:panelGrid>`
  - Отображает HTML-таблицу
  - Выводит таблицу целиком
  - Атрибуты, влияющие на отображение:
    - `columnClasses`, `columns`, `footerClass`, `headerClass`, `panelClass`, `rowClasses`
- `<h:panelGroup>`
  - Группирует компоненты
  - Представляет строку таблицы

# Пример: `<h:panelGrid>` на странице `confirmChoices.jsp`

```
<h:panelGrid columns="2" footerClass="subtitle"
  headerClass="subtitlebig" styleClass="medium"
  columnClasses="subtitle,medium">

  <f:facet name="header">
    <h:outputText value="You have chosen the following options:" />
  </f:facet>
  <h:outputText value="Engine" />
  <h:outputText value="#{carstore.currentModel.attributes.engine}" />
  <h:outputText value="Brakes" />
  ...
  <f:facet name="footer">
    <h:panelGroup>
      <h:outputText value="Your Price" />
      &nbsp;
      <h:outputText value="#{carstore.currentModel.currentPrice}" />
    </h:panelGroup>
  </f:facet>
  ...
</h:panelGrid>
```

# confirmChoices.jsp

