Luni

MVC para aplicações Web

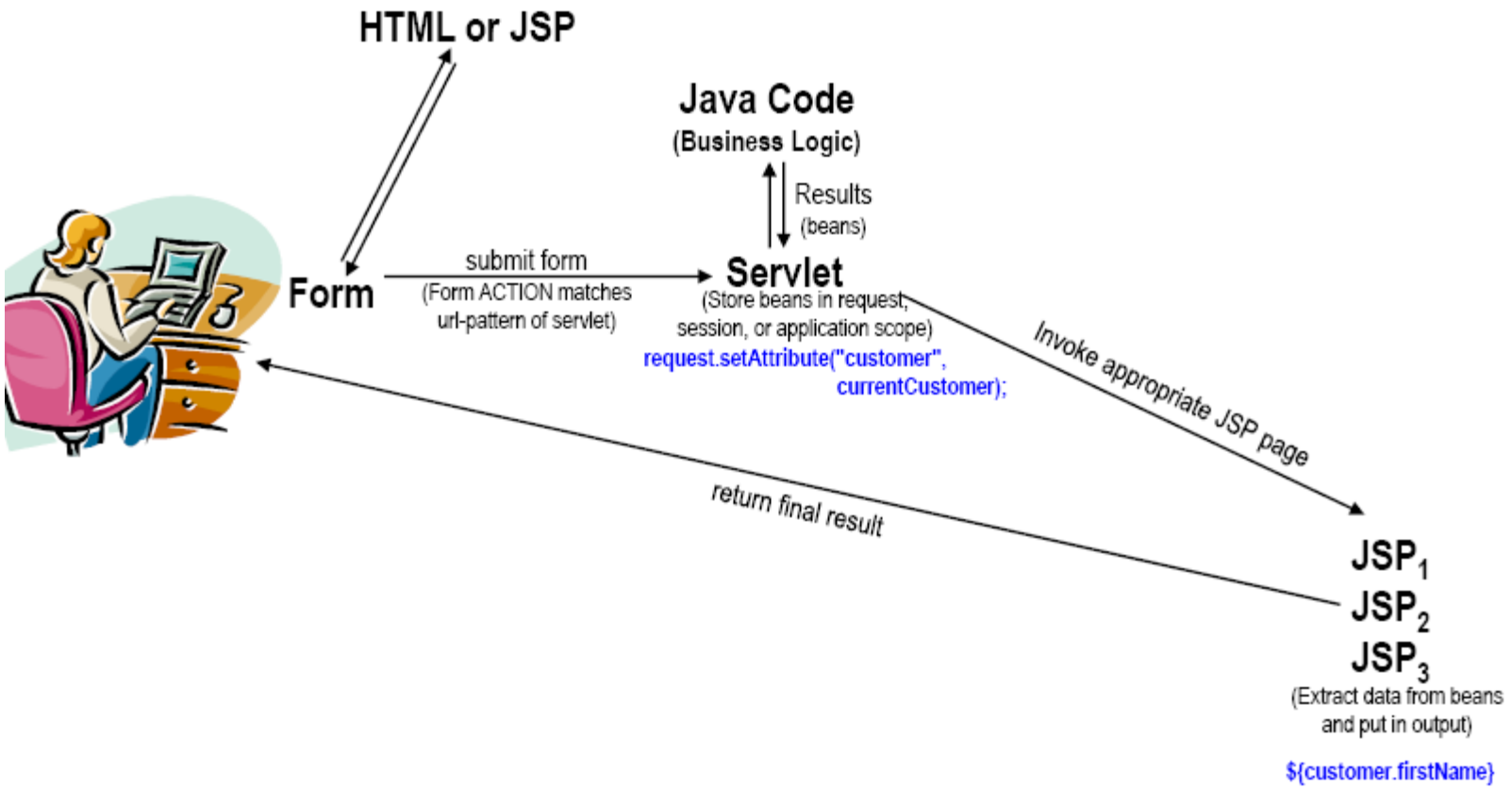
Laércio Baldochi

COM222
Aula 07

Conteúdo

- Modelo MVC para aplicações Web
  - Uso de JSP, Servlets e Java Beans em conjunto
    - Servlet como um dispatcher de requisições
    - JSP como View
    - Beans como model
    - Classes Java como control

MVC – Fluxo de controle



Beans

Revisão: o que são Java beans?

- Classes Java que seguem certas convenções
  - Devem ter um construtor vazio (omitir construtor)
  - Não devem possuir atributos públicos (apenas privados)
  - Valores persistentes (atributos privados) devem ser acessados por métodos getXxx e setXxx
    - Se a classe tem um método getTitulo que retorna uma String, diz-se que a classe tem uma **propriedade** String chamada Titulo
      - Daí vem o **getProperty** e **setProperty**
    - Para atributos booleanos pode-se usar isXxx ao invés de getXxx

Bean properties
Exemplos

Method Names	Property Name	Example JSP Usage
getFirstName setFirstName	firstName	<jsp:getProperty … property="firstName"/> <jsp:setProperty … property="firstName"/> ${customer.firstName}
isExecutive setExecutive (boolean property)	executive	<jsp:getProperty … property="executive"/> <jsp:setProperty … property="executive"/> ${customer.executive}
getExecutive setExecutive (boolean property)	executive	<jsp:getProperty … property="executive"/> <jsp:setProperty … property="executive"/> ${customer.executive}
getZIP setZIP	ZIP	<jsp:getProperty … property="ZIP"/> <jsp:setProperty … property="ZIP"/> ${address.ZIP}

MVC com RequestDispatcher

Passos

1. Definir Java Beans para conter os dados (model)
2. Utilizar um servlet para processar requisições
3. Popular os Java Beans

   O servlet invoca o **business logic** da aplicação e coloca os **resultados nos beans** definidos no passo 1
4. Armazenar o bean em um contexto: request, session ou application

   O servlet chama **setAttribute** em um objeto de contexto para armazenar uma referência aos beans que representam os resultados da requisição

MVC com RequestDispatcher
Passos

5. Enviar (forward) a requisição para uma página JSP
   - O servlet determina qual página JSP é apropriada para a situação e usa o método **forward** do RequestDispatcher para transferir o controle para a página
6. Extrair os dados do(s) bean(s)
   - A página JSP acessa os beans com **jsp:useBean** e um escopo apropriado (de acordo com item 4).
   - Usa **jsp:getProperty** para obter as propriedades do bean

Usando RequestDispatcher

Código

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
  throws ServletException, IOException {
  ... // Do business logic and get data
  String operation = request.getParameter("operation");
  if (operation == null) {
    operation = "unknown";
  }
  String address;
  if (operation.equals("order")) {
    address = "/WEB-INF/Order.jsp";
  } else if (operation.equals("cancel")) {
    address = "/WEB-INF/Cancel.jsp";
  } else {
    address = "/WEB-INF/UnknownOperation.jsp";
  }
  RequestDispatcher dispatcher =
    request.getRequestDispatcher(address);
  dispatcher.forward(request, response);
```

MVC (Servlet + JSP)

- Model
  - Java Beans
- View
  - Páginas JSP
- Control
  - Servlet
    - Classes Java (business logic)

View

Páginas JSP

- Páginas JSP não devem criar objetos
  - O servlet deve ser responsável pela criação dos objetos. Para garantir que páginas JSP não criem objetos deve-se usar
    - <jsp:useBean … type="package.class"/>

    ao invés de
    - <jsp:useBean … class="package.class"/>

- Páginas JSP não devem modificar objetos
  - Permitido apenas jsp:getProperty e nunca jsp:setProperty

jsp:useBean
Alternativas de escopo

- **request**
  - <jsp:useBean id="..." type="..." scope="request" />
- **session**
  - <jsp:useBean id="..." type="..." scope="session" />
- **application**
  - <jsp:useBean id="..." type="..." scope="application" />
- **page**
  - <jsp:useBean id="..." type="..." scope="page" />
    or just
    <jsp:useBean id="..." type="..." />

Compartilhamento de dados no escopo de requisição: exemplo

- **Servlet**

Assume that the Customer constructor handles missing/malformed data.

```
Customer myCustomer =
  new Customer(request.getParameter("customerID"));
request.setAttribute("customer", myCustomer);
RequestDispatcher dispatcher =
  request.getRequestDispatcher
                              ("/WEB-INF/SomePage.jsp");
dispatcher.forward(request, response);
```

- **JSP 1.2**

```
<jsp:useBean id="customer" type="somePackage.Customer"
             scope="request" />
<jsp:getProperty name="customer" property="firstName"/>
```

- **JSP 2.0**

```
${customer.firstName}
```

Note: the Customer class must have a method called "getFirstName".

Compartilhamento de dados no escopo de aplicação (servletContext)

- **Servlet**

```
synchronized(this) {
  ValueObject value = new ValueObject(...);
  getServletContext().setAttribute("key", value);
  RequestDispatcher dispatcher =
    request.getRequestDispatcher
                        ("/WEB-INF/SomePage.jsp");
  dispatcher.forward(request, response);
}
```

- **JSP 1.2**

```
<jsp:useBean id="key" type="somePackage.ValueObject"
             scope="application" />
<jsp:getProperty name="key" property="someProperty" />
```

- **JSP 2.0**

```
${key.someProperty}
```

Estudo de caso
Saldo bancário

- Faremos uma aplicação que informa o saldo bancário de clientes com páginas personalizadas para diferentes tipos de clientes
- Aplicação implementada segundo os preceitos do MVC:
  - **Java beans** contendo o modelo de domínio
  - **Classes Java** para executar lógica de negócio
  - **Um servlet** para fazer o controle
  - **Páginas JSP** para a parte de visão

Estudo de caso
Saldo bancário

- Bean
  - BankCustomer
- Business Logic
  - BankCustomerLookup
- Servlet
  - Carrega o bean e faz o forward para uma página JSP apropriada
- 4 páginas JSP para mostrar resultados
  - Saldo negativo: página com mensagem de atenção
  - Saldo positivo: página padrão
  - Saldo elevado: página com propaganda
  - Cliente desconhecido: página de erro

BankCustomer.java

```
public class BankCustomer {
  private String id, firstName, lastName;
  private double balance;

  public BankCustomer(String id, String firstName, String lastName, double balance) {
    this.id = id; this.firstName = firstName; this.lastName = lastName; this.balance = balance;
  }

  public String getId() {
    return id;
  }

  public String getFirstName() {
    return(firstName);
  }

  public String getLastName() {
    return(lastName);
  }

  public double getBalance() {
    return(balance);
  }

  public double getBalanceNoSign() {
    return(Math.abs(balance));
  }
}
```

BankCustomerLookup.java

```
import java.util.*;

public class BankCustomerLookup {
  // Monta uma pequena tabela de clientes
  private static Map<String,BankCustomer> customers;

  static {
    customers = new HashMap<String,BankCustomer>();
    customers.put("id001", new BankCustomer("id001", "John", "Hacker", -3456.78));
    customers.put("id002", new BankCustomer("id002", "Jane", "Hacker", 1234.56));
    customers.put("id003", new BankCustomer("id003", "Juan", "Hacker", 987654.32));
  }

  public static Map<String,BankCustomer> getSampleCustomers() {
    return(customers);
  }

  /** Finds the customer with the given ID.
   *  Returns null if there is no match.
   */

  public static BankCustomer getCustomer(String id) {
    return(customers.get(id));
  }

  private BankCustomerLookup() {} // Classe não instanciável
}
```

ShowBalance.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ShowBalance extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    BankCustomer customer = BankCustomerLookup.getCustomer(request.getParameter("id"));
    String address;
    if (customer == null) {
        address = "/WEB-INF/bank-account/UnknownCustomer.jsp";
    } else if (customer.getBalance() < 0) {
        address = "/WEB-INF/bank-account/NegativeBalance.jsp";
        request.setAttribute("badCustomer", customer);
    } else if (customer.getBalance() < 10000) {
        address = "/WEB-INF/bank-account/NormalBalance.jsp";
        request.setAttribute("regularCustomer", customer);
    } else {
        address = "/WEB-INF/bank-account/HighBalance.jsp";
        request.setAttribute("eliteCustomer", customer);
    }
    RequestDispatcher dispatcher = request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
  }
}
```

Páginas JSP

- Analisando o código do servlet percebemos que os arquivos JSP devem ser colocados no diretório bank-account, o qual deve ser criado em WEB-INF

```
if (customer == null) {
    address = "/WEB-INF/bank-account/UnknownCustomer.jsp";
} else if (customer.getBalance() < 0) {
    address = "/WEB-INF/bank-account/NegativeBalance.jsp";
    request.setAttribute("badCustomer", customer);
} else ....
```

HighBalance.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Seu saldo</TITLE>
<LINK REL=STYLESHEET
     HREF="./bank-support/JSP-Styles.css"
     TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
     Seu saldo</TABLE>
<P>
<CENTER><IMG SRC="./bank-support/Sailing.gif"></CENTER>
<BR CLEAR="ALL">
É uma honra servi-lo, Sr
${eliteCustomer.firstName} ${eliteCustomer.lastName}!
<P>
Considerando que o Sr é um dos nossos melhores clientes, gostaríamos
de oferecer-lhe a oportunidade de investir uma pequena fração dos seus
$${eliteCustomer.balance}
em um iate digno do seu status. Por favor, visite nossa loja de barcos
para maiores informações.
</BODY>
</HTML>
```

NegativeBalance.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Você nos deve dinheiro!</TITLE>
<LINK REL=STYLESHEET
    HREF="./bank-support/JSP-Styles.css"
    TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
    Nós sabemos onde você mora!</TABLE>
<P>
<IMG SRC="./bank-support/Club.gif" ALIGN="LEFT">
Fique atento, ${badCustomer.firstName},
nós sabemos onde você mora!
<P>
Melhor você pagar os $${badCustomer.balanceNoSign}
que nos deve antes que seja tarde!
</BODY>
</HTML>
```

NormalBalance.jsp

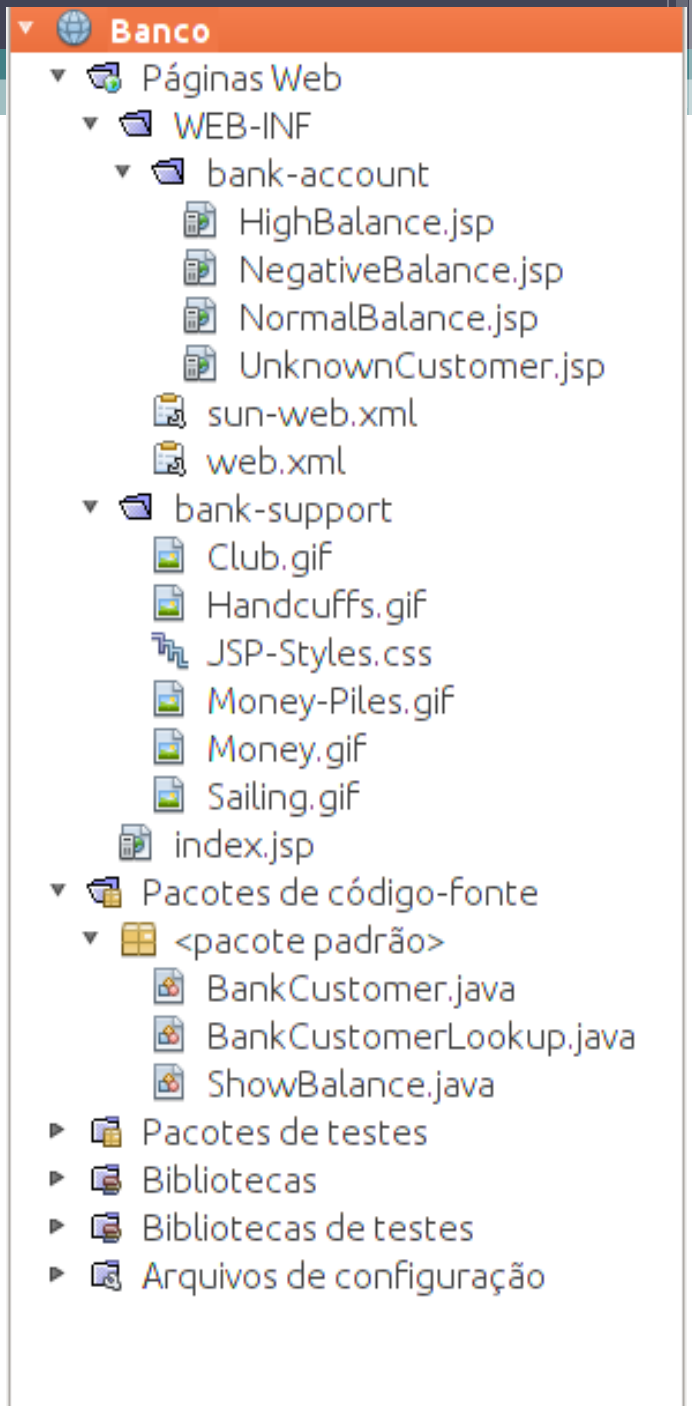
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Seu saldo</TITLE>
<LINK REL=STYLESHEET
    HREF="./bank-support/JSP-Styles.css"
    TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
    Seu saldo</TABLE>
<P>
<IMG SRC="./bank-support/Money.gif" ALIGN="RIGHT">
<UL>
  <LI>Nome: ${regularCustomer.firstName}
  <LI>Sobrenome: ${regularCustomer.lastName}
  <LI>ID: ${regularCustomer.id}
  <LI>Saldo: $${regularCustomer.balance}
</UL>
</BODY>
</HTML>
```

UnknownCustomer.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Unknown Customer</TITLE>
<LINK REL=STYLESHEET
    HREF="./bank-support/JSP-Styles.css"
    TYPE="text/css">
</HEAD>
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
    Cliente desconhecido</TABLE>
<P>
Identificador de cliente não cadastrado.
</BODY>
</HTML>
```

Figuras

- Notem que os JSPs usam figuras, as quais devem devem ser colocadas no diretório bank-support
- Download das figuras
  - www.baldochi.unifei.edu.br/COM222/bank-support.rar



index.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
   Transitional//EN">
<HTML>
<HEAD>
<TITLE>Saldo bancário</TITLE>
<LINK REL=STYLESHEET
     HREF="./bank-support/JSP-Styles.css"
     TYPE="text/css">
</HEAD>
<BODY>
<fieldset>
 <legend>Saldo bancário</legend>
 <form action="./ShowBalance">
  ID do Cliente: <input type="text" name="id"><br>
  <input type="submit" value="Mostra saldo">
 </form>
</fieldset>
<br>
</BODY></HTML>
```

Exercício 1

Entrega: hoje

- Crie a tabela Cliente no banco de dados Banco
  - Nro conta: int
  - Nome: varchar(30)
  - Saldo: real

- CREATE TABLE `cliente` (`nro_conta` int(11) NOT NULL, `nome` varchar(40) NOT NULL, `saldo` double NOT NULL) ENGINE=InnoDB;
- ALTER TABLE `cliente` ADD PRIMARY KEY (`nro_conta`);

Exercício 1

Entrega: hoje

- Popule a tabela Cliente com alguns registros
  - Saldo negativo
  - Saldo positivo, porém menor que 10.000
  - Saldo superior a 10.000
- Modifique o servlet de modo a verificar se o número de conta informado existe no banco
  - Em caso afirmativo, criar um bean com os dados do banco que correspondem ao número de conta informado

Exercício 1

Entrega: hoje

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ShowBalance extends HttpServlet {
  public void doGet(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
    BankCustomer customer =
    BankCustomerLookup.getCustomer(request.getParameter("id"));
    String address;
    if (customer == null) {
    ......
```

Exercício 2

Entrega: 30/03

- Elabore uma aplicação de home banking utilizando o modelo MVC com as seguintes operações
  - Criação de contas com saldo inicial
  - Saques
  - Depósitos
  - Transferências
  - Saldo

- As informações dos clientes devem ser mantidas no banco de dados, conforme exercício 1

Exercício 2

Entrega: 30/03

- A página inicial (index.jsp) deve conter um conjunto de radio buttons para permitir que o usuário escolha qual operação deseja executar
- A escolha deve ativar um servlet (operationServlet) que faz o "dispatch" para a página JSP que colherá os dados para realizar a operação desejada
  - 5 páginas JSP são necessárias
  - Para cada página JSP deve haver um servlet que recebe os dados, executa a lógica de negócio e faz o dispatch para outras páginas JSP com a resposta da transação solicitada
  - As páginas de resposta devem conter um link para a página inicial

Exercício 2

Entrega: 30/03

- Bean Cliente
  - Deverá conter o modelo de dados da aplicação (model)
- Classes de lógica de negócio
  - Criar classes que executem a lógica de negócio necessária à aplicação (control)
- Importante
  - Toda a parte de visão deve utilizar páginas JSP