

# INE5646 – Programação para Web

Unidade III – Seção IV

## Aplicações Baseadas na *Java Virtual Machine (JVM)*

Prof. Frank Siqueira – Turma A

Prof. Leandro Komosinski – Turma B

# Conteúdo

- A Plataforma Java EE
- Servidores Java EE
- JSP
- Servlets
- JSF
- EJB
- JPA
- Segurança

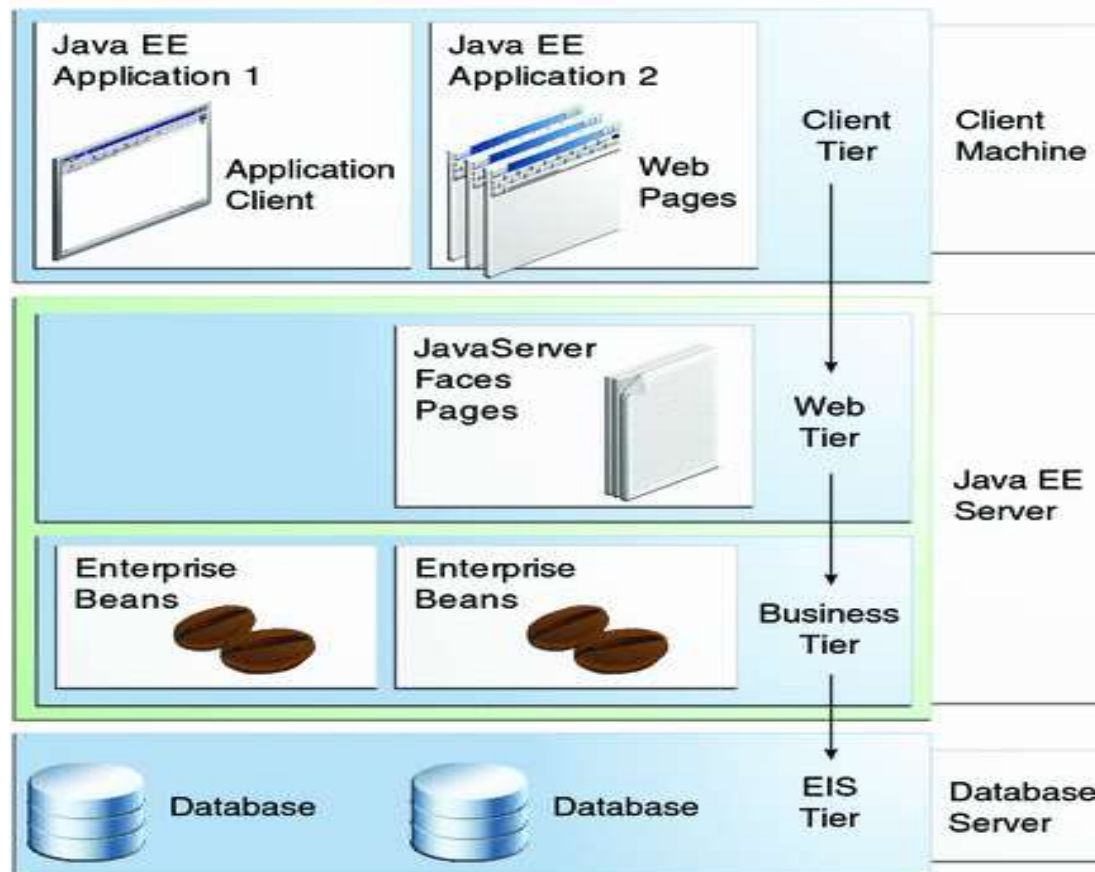
# A Plataforma Java EE



- Plataforma Java Enterprise Edition
  - Adiciona ao Java suporte para:
    - Desenvolvimento de Aplicações Web: JSP, *Servlets* e JSF
    - Componentes de Negócio: EJB
    - Interconexão com Sistemas Legados: Java *Connectors*
  - Fornece diversas APIs para:
    - Comunicação: JMS, JavaMail
    - Gerenciamento de Transações: JTA
    - Persistência de Dados: JPA
    - ... entre outros.

# A Plataforma Java EE

- Arquitetura do Java EE



# A Plataforma Java EE

- Camadas do Java EE
  - Camada Cliente
    - Clientes Web (navegadores, etc.)
    - Aplicações Clientes
  - Camada Web
    - Páginas JSP, JSF, *Servlets* e *JavaBeans*
  - Camada de Negócios
    - Componentes EJB
  - Camada de Sist. de Informações Empresariais
    - Integração com BDs e outros sist. legados

# A Plataforma Java EE

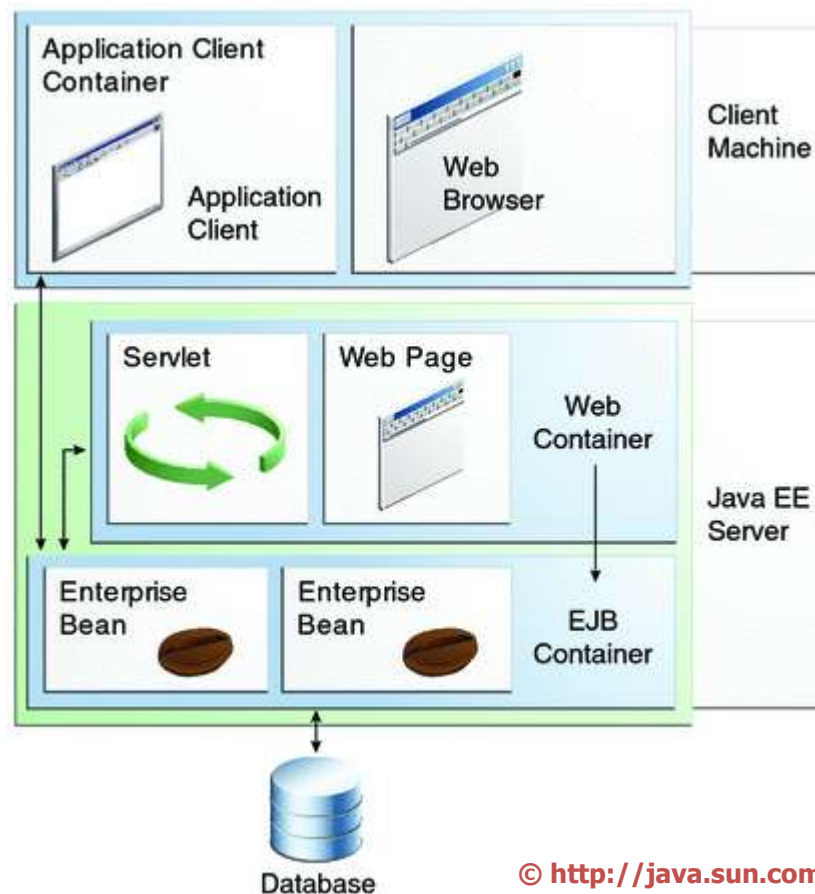
- Camada Cliente
  - Clientes Web
    - Acessam a camada Web, que gera as páginas visualizadas no navegador
    - Utilizam protocolos HTTP e HTTPS
  - Aplicações Clientes
    - Aplicações Java ou CORBA (multi-linguagem)
    - Acessam a camada de negócios diretamente
    - Interagem através do protocolo IIOP

# A Plataforma Java EE

- Servidor Java EE
  - Possui duas camadas:
    - Camada Web
      - Composta por páginas JSP e JSF, *Servlets* e *JavaBeans*
      - Acessada pelos clientes Web
    - Camada de Negócios
      - Formada por componentes EJB
      - Usada pela camada Web e por aplicações clientes

# A Plataforma Java EE

- Contêineres do Servidor Java EE





# A Plataforma Java EE

- Servidor Java EE
  - Existem dois tipos de contêiner
    - Contêiner Web: hospeda páginas JSP e JSF, *Servlets* e *JavaBeans*
    - Contêiner EJB: gerencia a execução dos *Enterprise JavaBeans*
  - Nem todos os servidores fornecem os dois tipos de contêiner
    - Ex.: Tomcat possui apenas contêiner Web

# A Plataforma Java EE

- *Java Server Pages (JSP)*
  - Permite a geração dinâmica de conteúdo Web
  - Código Java é inserido em páginas HTML ou XML para gerar conteúdo dinâmico
  - Geração do conteúdo baseada em parâmetros passados na URL, na identificação do usuário, dados de um BD ou de JavaBeans, etc.
  - Requer um servidor Web compatível com Java EE para executar o código JSP
  - Cliente Web não tem acesso ao código JSP
  - JSP, ao ser compilado, gera um *servlet*

# A Plataforma Java EE

- *Servlets*
  - São classes Java que implementam a interface `javax.servlet.Servlet`
  - Instanciados e mantidos em execução no servidor
  - Processam requisições enviadas para uma URI
  - Servlet HTTP
    - Recebe requisições via HTTP[S]
    - Possui métodos que tratam cada tipo de mensagem do protocolo (GET, POST, etc.)
    - Geram saída exibida no *browser* (ex: HTML)

# A Plataforma Java EE

- *Java Server Faces* (JSF)
  - Suporte para criação de aplicações Web utilizando componentes
  - Facilita o desenvolvimento de aplicações Web
  - Fornece componentes para criação de páginas
  - Efetua tratamento de eventos gerados pela interação do usuário com o navegador Web

# A Plataforma Java EE

- *Enterprise JavaBeans* (EJB)
  - Componentes que rodam no servidor
  - Acessam os sistemas legados da empresa para implementar regras de negócio
  - Ciclo de vida gerenciado pelo contêiner
  - Persistência de dados efetuada pela JPA (*Java Persistence API*)

# A Plataforma Java EE

- Camada de Sistemas de Informações Empresariais (EIS)
  - Usada pelos componentes EJB da camada de negócio p/ acesso a *software* de infraestrutura
    - Banco de Dados
    - Monitores de Transações
    - *Enterprise Resource Planning* (ERP)
    - *Customer Relationship Management* (CRM)
    - ... e outros sistemas legados
  - Estes sistemas geralmente rodam em *mainframes* ou servidores de médio porte
  - Conectores permitem o acesso a sist. legados

# A Plataforma Java EE

- Conectores
  - Integram diversos sistemas à plataforma Java EE
  - Fornecido pelo fabricante do sistema legado ou por terceiros
  - Para desenvolver um conector geralmente é necessário escrever código nativo para a plataforma do sistema legado e integrar ao Java usando JNI (*Java Native Interface*), CORBA ou Sockets

# A Plataforma Java EE

- *Java Messaging Service (JMS)*
  - Serviço para comunicação através de mensagens assíncronas (eventos)
- *JavaMail*
  - API para envio e recepção de e-mails
- *Java Transaction API (JTA)*
  - API para gerenciamento de transações
- *Java Persistence API (JPA)*
  - API que mapeia os dados das aplicações corporativas de/para banco de dados



# A Plataforma Java EE

- Distribuição de aplicações corporativas
  - Arquivos que compõem uma aplicação Web são empacotados num arquivo WAR
  - Arquivos necessários para implantar EJBs devem ser empacotados em arquivos JAR
  - Uma aplicação corporativa completa é empacotada em um arquivo EAR
    - Contém uma ou mais aplicações Web em arquivos WAR
    - Contém um ou mais arquivos JAR com os componentes EJB da aplicação, aplicações cliente e outras bibliotecas utilizadas

# A Plataforma Java EE

- Implantação de aplicações corporativas
  - Arquivos EAR são carregados no servidor Java EE, que abre o pacote, instale e executa a aplicação
    - O conteúdo dos arquivos é WAR implantado no contêiner Web
    - Componentes EJB contidos em arquivos JAR são implantados no contêiner EJB
  - A implantação é efetuada com base em informações obtidas de descritores em XML e de anotações feitas nas próprias classes Java

# Servidores Java EE

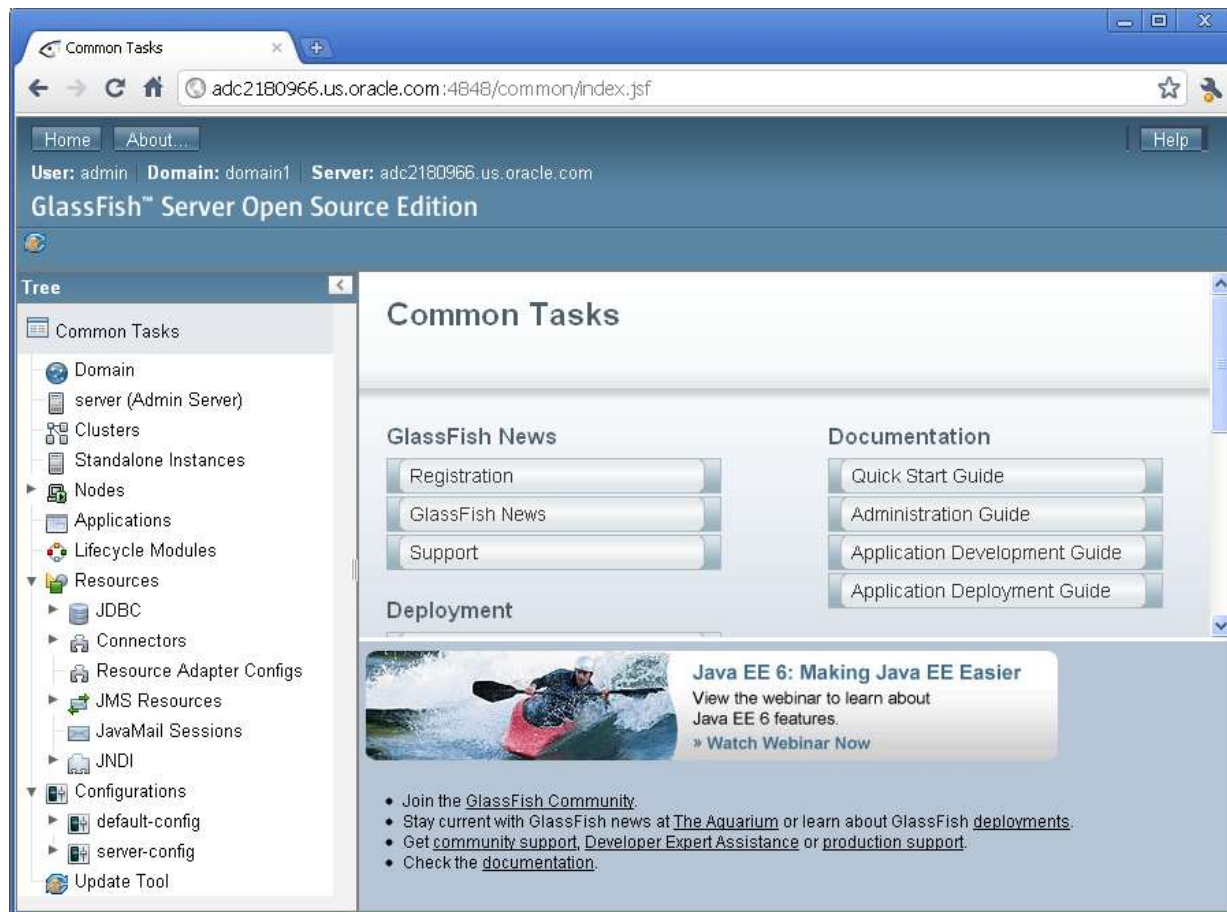
- Lista de servidores Java EE certificados
  - Tomcat (Apache) e Jetty (Eclipse)
    - Gratuitos; cód. aberto; somente camada Web do Java EE.
  - Glassfish (Oracle)
    - Implementação de referência do Java EE; gratuito e de código aberto; versão paga com suporte.
  - TomEE (Apache)
    - Java EE completo; baseado no Tomcat; gratuito e aberto.
  - JBoss (Red Hat)
    - Java EE completo; gratuito e aberto; suporte pago.
  - WebLogic (Oracle) e Websphere (IBM)
    - Java EE completo; sw. comerciais (pagos) com suporte.
  - ...

# Servidores Java EE

- Instalação do Glassfish
  - Download: acesse <http://glassfish.java.net>, baixe a última versão e descompacte o arquivo
  - Execução: `asadmin start-domain <nome-domínio>`
  - Encerramento: `asadmin stop-domain <nome-domínio>`
- Configuração e Administração
  - Acesse a interface de administração no endereço:  
<http://<hostname>:4848/>  
login: admin  
senha padrão: adminadmin

# Servidores Java EE

- GlassFish Admin Console



# JSP

- Java Server Pages
  - Linguagem para geração de conteúdo Web
  - Código Java é inserido em páginas HTML, WML ou XML para gerar conteúdo dinâmico
  - Páginas pode ser geradas com base em parâmetros passados na URL, dados mantidos em um BD, na identificação do usuário, etc.
  - Cliente Web não tem acesso ao código JSP
  - Requer um servidor Web compatível com Java EE para executar o código JSP

# JSP

- Desenvolvimento de Páginas JSP
  - Divisão de responsabilidades
    - Designers devem se preocupar com o layout das páginas Web
    - Programadores fazem o desenvolvimento do código em JSP
  - É possível separar claramente design de programação encapsulando a lógica da aplicação em componentes *JavaBeans*
  - O designer deve apenas chamar o *bean* a partir da página JSP

# JSP

- Compilação
  - A compilação de uma página JSP gera um *servlet* que irá tratar as requisições enviadas a esta página Web
- Execução
  - O *servlet* é instanciado no servidor e atende as requisições enviadas à URL correspondente



# JSP

- Importação de pacotes e classes Java:  
`<%@ page import="java.util.*, pacote.Classe" %>`
- Inclusão de páginas:
  - Inclusão na tradução da página:  
`<%@ include file="arquivo.jsp" %>`
  - Inclusão na execução da página:  
`<jsp:include page="arquivo.jsp" />`
- Encaminhamento para outra página  
`<jsp:forward page="destino.jsp" />`

# JSP

- Declarações de Atributos e Métodos

`<%! ...declarações Java... %>`

- Exemplo:

```
<%! private MinhaClasse meuAtributo;  
      public String mensagem;  
      public void meuMétodo() { ... }  
      public String getMensagem() {  
          return mensagem;  
      }  
%>
```

# JSP

- Declaração de *Scriptlets*

`<% ...código Java... %>`

- Declaração de Expressões

`<%= ...expressão Java... %>`

- Exemplo

`<% for (int i = 0; i<alunos.length(); i++) { %>`

`<p>Aluno: <%=aluno[i].getNome()%>`

`<p>Nota: <%=aluno[i].getNota()%>`

`<% } %>`

# JSP

- Usando JavaBeans em Páginas JSP

```
<jsp:useBean id="meuBean"  
    class="pacote.nomeBean" scope="session"/>
```

- Modificando Propriedades

```
<jsp:setProperty name="meuBean"  
    property="propriedade" value="Valor" />  
<% meuBean.setPropriedade(valor); %>
```

- Recuperando Propriedades

```
<jsp:getProperty name="meuBean"  
    property="propriedade" />  
<%= meuBean.getPropriedade() %>
```

# JSP

- Escopo dos *Beans*

Escopo	Ciclo de Vida
<i>Application</i>	A mesma instância é compartilhada por todos os usuários da aplicação
<i>Session</i>	Uma instância por sessão (cada usuário possui a sua instância particular)
<i>Request</i>	Instância criada/destruída a cada requisição enviada ao servidor
<i>Page</i>	A instância do <i>bean</i> é acessada somente na página atual

# JSP

- Objetos Implícitos

<i>ServletContext application</i>	Contexto da aplicação
<i>ServletConfig config</i>	Informações de inicialização
<i>JspWriter out</i>	Fluxo de saída
<i>Object page</i>	Instância do servlet que processa a página JSP
<i>PageContext pageContext</i>	Contexto da página JSP
<i>ServletRequest request</i>	Requisição da página JSP
<i>ServletResponse response</i>	Resposta à requisição atual
<i>HttpSession session</i>	Sessão atual
<i>Throwable exception</i>	Informação de erro

# JSP

- Exemplo de JSP
  - Suponha que uma operadora de cartões de crédito está distribuindo prêmios aos clientes
  - Um código de 12 caracteres é impresso nos comprovantes de compra
  - O código deve ser digitado no site para verificar se o cliente foi premiado
  - A promoção é válida até uma determinada data
  - Um *bean* contém a lista de códigos premiados
  - Seguem os códigos das páginas:
    - index.jsp: página para digitação do código
    - result.jsp: verifica se o código é premiado

# JSP

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Promoção Compra Premiada</title>
  </head>
  <jsp:useBean id="promocao" class="beans.Promocao" scope="application" />
  <body>
    <h1>Promoção Compra Premiada</h1>
    <% if (!promocao.isEncerrada()) { %>
      <p>Seu cartão de crédito está distribuindo prêmios de R$5.000,00!</p>
      <p>Entre com o código impresso no comprovante de compra:</p>
      <form name="Codigo" action="result.jsp" method="GET">
        <input type="text" name="codigo" value="" size="12" />
        <input type="submit" value="Enviar" />
      </form>
    <% } else { %>
      <p>Promoção encerrada em <%= promocao.getDataFimPromo() %>.</p>
    <% } %>
  </body>
</html>
```

A página *index.jsp* permite que clientes de uma operadora de cartão participem de uma promoção.

O *bean promocao* armazena os códigos premiados e a data de encerramento da promoção.

Se a promoção ainda estiver em vigor, será exibido um formulário para envio do código da compra.

Uma mensagem será exibida se a promoção já tiver sido encerrada.



# JSP

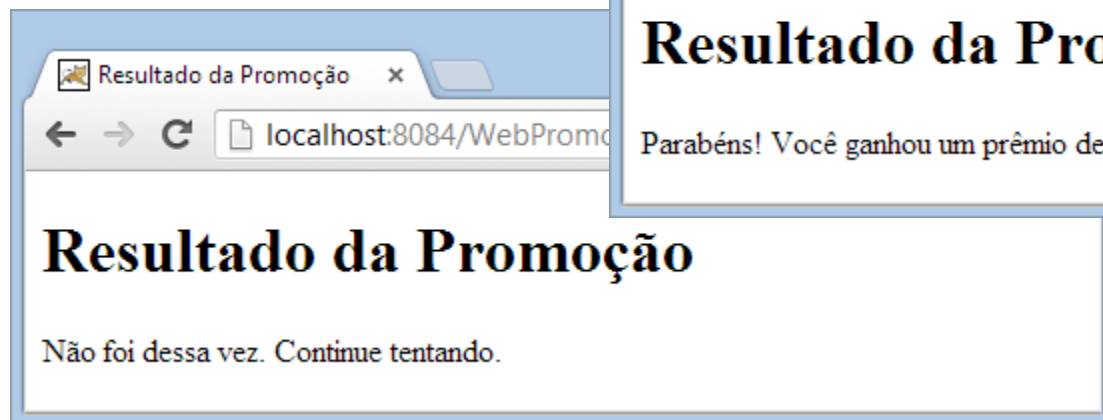
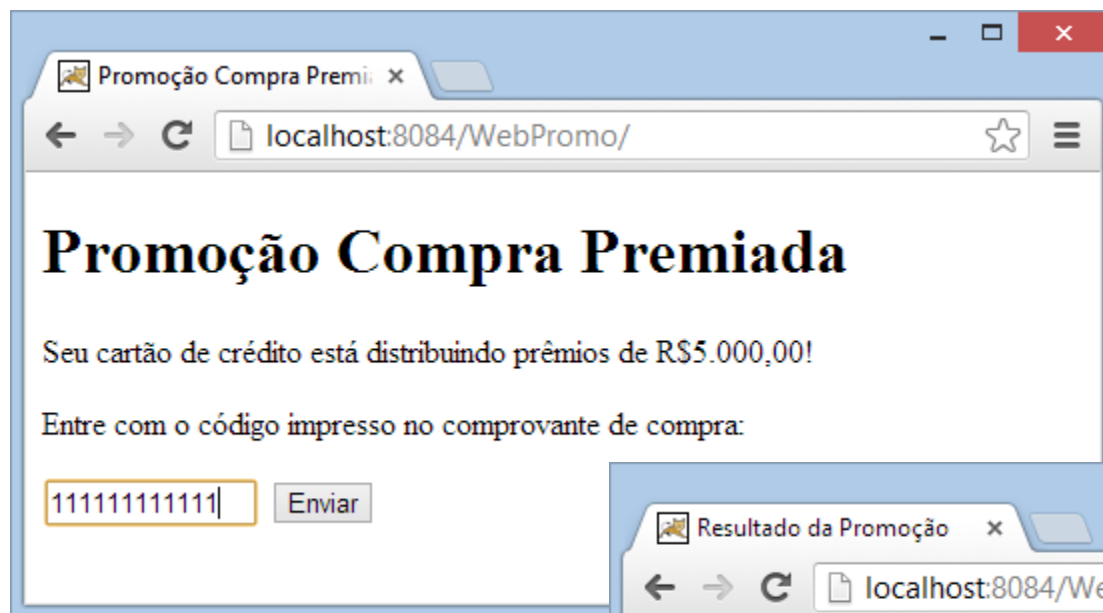
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Resultado da Promoção</title>
  </head>
  <jsp:useBean id="promocao" scope="application" class="beans.Promocao" />
  <body>
    <% if (request.getParameter("codigo").length() != 12) { %>
      <h1>Erro</h1>
      <p>Código inválido. O código deve ter 12 caracteres.</p>
    <% } else { %>
      <h1>Resultado da Promoção</h1>
      <% if (promocao.premiados.contains(request.getParameter("codigo"))) { %>
        <p>Parabéns! Você ganhou um prêmio de R$5.000,00!</p>
      <% } else { %>
        <p>Não foi dessa vez. Continue tentando.</p>
      <% }
    } %>
  </body>
</html>
```

A página *result.jsp* exibe o resultado da promoção.

Primeiramente o código é validado.

Verifica-se então se o código é premiado ou não.

# JSP



# JSP

- JSTL (*JSP Standard Tag Library*)
  - Provê *tags* para adicionar lógica de programação às páginas JSP, de modo a evitar a mistura de código Java com HTML
  - Bibliotecas nativas JSTL:
    - core: construções básicas, como condições e laços.
    - xml: para manipulação de XML.
    - sql: para acesso a bancos de dados via SQL.
    - fmt: para formatação, internacionalização, etc.

# JSP

- JSTL – Principais *tags*:
  - Importação de biblioteca de *tags* (**taglib**):  
`<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>`
  - Variáveis (**set**):  
`<c:set var="x" value="{y+z}" />`
  - Saída de dados (**out**):  
`<c:out value="{y+z}" />`
  - Condição (**if**):  
`<c:if test="{x} >= 100">`  
...  
`</c:if>`

# JSP

- JSTL – Principais tags (cont.):
  - Laço (forEach):
    - Numérico:

```
<c:forEach var="i" begin="1" end="100" step="5">
...
</c:forEach>
```
    - Com lista de itens:

```
<c:forEach var="item" items="${bean.lista}">
...
</c:forEach>
```

# JSP

- JSTL – Principais tags (cont.):
  - Seleção (**choose**):

```
<c:choose>  
  <c:when test="${x==1}">  
    ...  
  </c:when>  
  <c:when test="${x==2}">  
    ...  
  </c:when>  
  <c:otherwise>  
    ...  
  </c:otherwise>  
</c:choose>
```

# JSP

- JSTL – Objetos Implícitos:
  - Mapas de valores sem escopo:
    - initParam
    - param
    - header
    - cookie
  - Mapas de arrays:
    - paramValues
    - headerValues
  - Mapas de valores em um escopo particular:
    - pageScope
    - requestScope
    - sessionScope
    - applicationScope
  - Contexto da página:
    - pageContext

# JSP

- JSLT – Acesso a mapas:
  - Formas válidas de acesso a mapa de *strings*:

`${param["codigo"]}`

`${param['codigo']}`

`${param.codigo}`

- Exemplo de acesso a mapa de *arrays*:

`<c:forEach var='parameter' items='${paramValues}'>`

`<p><b><c:out value='${parameter.key}'/></b>:`

`<c:forEach var='value' items='${parameter.value}'>`

`<c:out value='${value}'/>`

`</c:forEach>`

`</c:forEach>`



# JSP

- *Expression Language* (EL)
  - Introduzida a partir do JSP 2.0
  - Permite o fácil acesso a JavaBeans e a suas propriedades, avaliação de condições, etc.
  - Podem ser usadas ao longo da página ou como valores de atributos de algumas *tags*
  - Acesso a variáveis simples: `${var}`
  - Acesso a propriedades de *beans*: `${bean.prop}`
  - Condições lógicas: `<c:if test="${var!=1}" > ... </c:if>`

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Promoção Compra Premiada</title>
  </head>
  <jsp:useBean id="promocao" class="beans.Promocao" scope="application" />
  <body>
    <h1>Promoção Compra Premiada</h1>
    <c:if test="${!promocao.encerrada}" >
      <p>Seu cartão de crédito está distribuindo prêmios de R$5.000,00!</p>
      <p>Entre com o código impresso no comprovante de compra:</p>
      <form name="Codigo" action="result.jsp" method="GET">
        <input type="text" name="codigo" value="" size="12" />
        <input type="submit" value="Enviar" />
      </form>
    </c:if>
    <c:if test="${promocao.encerrada}">
      <p>Promoção encerrada em ${promocao.dataFimPromo}.</p>
    </c:if>
  </body>
</html>

```

Página inicial da promoção do cartão, agora usando JSTL.

O *bean* *promocao* armazena os códigos premiados e a data de encerramento da promoção.

Se a promoção ainda estiver em vigor, será exibido um formulário para envio do código da compra.

Uma mensagem será exibida se a promoção já tiver sido encerrada.

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Resultado da Promoção</title>
  </head>
  <jsp:useBean id="promocao" scope="application" class="beans.Promocao" />
  <body>
    <c:if test="${param.codigo.length() != 12}">
      <h1>Erro</h1>
      <p>Código inválido. O código deve ter 12 caracteres.</p>
    </c:if>
    <c:if test="${param.codigo.length() == 12}">
      <h1>Resultado da Promoção</h1>
      <c:set var="ganhou" value="false" />
      <c:forEach var="cod" items="${promocao.premiados}" >
        <c:if test="${cod.equals(param.codigo)}" >
          <p>Parabéns! Você ganhou um prêmio de R$5.000,00!</p>
          <c:set var="ganhou" value="true" />
        </c:if>
      </c:forEach>
      <c:if test="${!ganhou}" >
        <p>Não foi dessa vez. Continue tentando.</p>
      </c:if>
    </c:if>
  </body>
</html>

```

A página de resultado,  
agora com JSTL.

Primeiramente o  
código é validado.

O laço verifica se o  
código é premiado.

Exibe uma mensagem  
se não for premiado.

# Servlets

- *Servlets* são classes Java que implementam a interface `javax.servlet.Servlet`
- A interface `Servlet` possui métodos para:
  - Inicialização do Servlet: método `init()`
  - Destruição do Servlet: método `destroy()`
  - Obtenção de configuração: `getServletConfig()`
  - Informações sobre o Servlet: `getServletInfo()`
  - Responder solicitações dos clientes: `service()`

# Servlets

- Implementações padrão da interface `javax.servlet.Servlet`
  - `GenericServlet`
    - Independente de protocolo
    - Desenvolvedor precisa tratar as requisições
  - `HttpServlet`
    - Recebe requisições através do protocolo HTTP
    - É acessado através de uma URL

# Servlets

- Servlets HTTP
  - Agem como *plug-ins* para o servidor Web
  - O servidor Java EE associa uma URL ao *servlet*
  - Não é necessário nenhum suporte especial no navegador do cliente para exibir a página
  - Ao receber uma requisição HTTP, o *servlet* executa o método correspondente ao tipo da requisição
    - GET: executa o método doGet()
    - POST: executa o método doPost()
    - etc.
  - Os métodos do *servlet* geram dinamicamente a página Web que será enviada ao cliente

# Servlets

- Exemplo de Servlet
  - Suponha que, no exemplo anterior em JSP, queremos verificar a resposta da promoção do cartão utilizando um *servlet* HTTP
  - Para que isso ocorra, a *action* especificada no formulário JSP deve apontar para a URL do *servlet*
  - A URL do *servlet* é definida na anotação `@WebServlet`
  - O *servlet* implementa o método `doGet()`, no qual verifica o código digitado e gera a resposta
  - O código digitado pelo usuário na página JSP é obtido chamando o método `request.getParameter()`
  - A saída exibida no cliente é gerada utilizando o `PrintWriter out`, obtido com o parâmetro `response`

```

@WebServlet(name = "result", urlPatterns = {"/result"})
public class Result extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<!DOCTYPE html><html><head>");
            out.println("<title>Resultado da Promoção</title></head><body>");
            String codigo = request.getParameter("codigo").toUpperCase();
            if (codigo.length() != 12) {
                out.println("<h1>Erro</h1>");
                out.println("<p>Código inválido. O código deve ter 12 caracteres.</p>");
            } else {
                out.println("<h1>Resultado da Promoção</h1>");
                if (beans.Promocao.premiados.contains(codigo)) {
                    out.println("<p>Parabéns! Você ganhou um prêmio de R$5.000,00!</p>");
                } else {
                    out.println("<p>Não foi dessa vez. Continue tentando.</p> ");
                }
            }
            out.println("</body></html>");
        } finally { out.close(); }
    }
}

```



# JSF

- *Java Server Faces* é um *framework* para construção de aplicações Web em Java
  - JSF é baseado em componentes para Web
  - Adota o padrão Modelo-Visão-Controlador
  - Utiliza JavaBeans gerenciados (com injeção de dependência) e é integrado a JSP e Servlets
  - Incorpora o conceito de eventos na navegação pela Web, com tratamento no servidor
  - Provê ainda APIs para controle de navegação na Web, validação de entrada, conversão de valores e suporte a localização e acessibilidade

# JSF

- Componentes JSF
  - O JSF fornece um conjunto de componentes comumente usados em páginas Web: link, tabela, botão, lista, área de texto, etc.
  - Há várias bibliotecas de componentes – comerciais ou gratuitas – disponíveis para uso (MyFaces, RichFaces, WoodStock, etc.)
  - Componentes são representados como *tags* em uma página JSP e posteriormente convertidos para o código HTML equivalente

# JSF

- Principais Componentes do JSF

- Formulário:

- `<h:form>`

- Entrada de dados:

- `<h:inputText>`

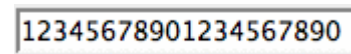
- `<h:inputSecret>`

- `<h:inputTextarea>`

- Saída de dados:

- `<h:outputText>`

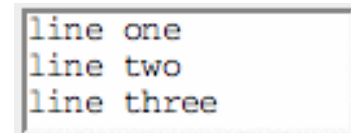
Formulário HTML



12345678901234567890



\*\*\*\*\*



line one  
line two  
line three

Texto na página HTML

# JSF

- Principais Componentes do JSF

- Mensagens de erro:

- `<h:message>`

- `<h:messages>`

- Botão:

- `<h:commandButton>`

- Link:

- `<h:commandLink>`

- Tabela:

- `<h:dataTable>`

Amount  Amount: 'too much' is not a number.  
Example: 99

- Preencha corretamente o nome.
- Preencha corretamente o CPF.

press me

register

Name	
Washington, George	<a href="#">Delete</a>
Jefferson, Thomas	<a href="#">Delete</a>
Lincoln, Abraham	<a href="#">Delete</a>
Roosevelt, Theodore	<a href="#">Delete</a>

# JSF

- Principais Componentes do JSF

- Caixas de seleção:

- `<h:selectBooleanCheckbox>`

- `<h:selectManyCheckbox>`

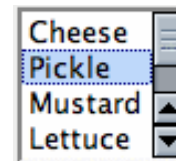
Receive email: ☒

☒ Cheese ☐ Pickle ☒ Mustard ☐ Lettuce ☐ Onions

- Listas:

- `<h:selectOneListbox>`

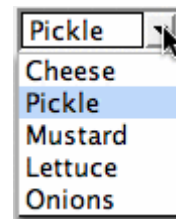
- `<h:selectManyListbox>`



- Menus:

- `<h:selectOneMenu>`

- `<h:selectManyMenu>`



- Listas:

- `<h:selectOneRadio>`

☐ Cheese ☒ Pickle ☐ Mustard ☐ Lettuce ☐ Onions

# JSF

- Modelo de eventos
  - Uma ação na página Web – como clicar um botão ou selecionar um item em uma lista – resulta em um evento
  - O evento pode ser associado a um método de um *bean* gerenciado, executado no servidor
  - Retorno do método tratador do evento pode determinar o fluxo de navegação da aplicação
  - O uso de eventos torna o desenvolvimento de aplicações Web semelhante ao *desktop*

# JSF

- *Facelets*
  - Forma padrão de implementar *views* no JSF2.0
  - Descritos na linguagem XHTML
  - Permite construir uma árvore de componentes e referenciar *beans* gerenciados JSF
- *Beans* Gerenciados
  - São *beans* usados na aplicação JSF
  - Seu ciclo de vida é gerenciado pelo servidor Java EE

# JSF

- Escopo dos *Beans* Gerenciados JSF

Escopo	Ciclo de Vida
<i>Application</i>	A mesma instância é compartilhada por todos os usuários da aplicação
<i>Session</i>	Uma instância por sessão (cada usuário possui a sua instância particular)
<i>Conversation</i>	Instâncias criadas/destruídas pela aplicação para diferenciar janelas/abas
<i>Request</i>	Instância criada/destruída a cada requisição enviada ao servidor
<i>Dependent</i>	Instância criada/removida quando o objeto que a referencia é criado/removido



# JSF

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Projeto de Lei de Iniciativa Popular</title>
  </h:head>
  <h:body>
    <h1>Projeto de Lei de Iniciativa Popular</h1>
    <p>Solicitamos o seu apoio para o projeto de lei de iniciativa popular
    que torna a segunda e a terça-feira de carnaval feriados nacionais.</p>
    <p>O texto do projeto está disponível <a href="img001.jpg">aqui</a>.</p>
    <p>Entre com os seus dados no formulário abaixo e apoie o projeto.</p>
    <h:form>
      <p>Nome: <h:inputText id="nome" size="50" maxlength="100"
        value="#{apoiador.nome}" /></p>
      <p>CPF: <h:inputText id="cpf" size="14" maxlength="14"
        value="#{apoiador.cpf}" /></p>
      <h:commandButton id="submit" value="Apoiar"
        action="#{projeto.apoiar(apoiador)}" />
      <h:messages style="color:red" />
    </h:form>
  </h:body>
</html>
```

O *facelet* index.xhtml permite que pessoas apoiem um projeto de lei de iniciativa popular.

O formulário tem campos para entrada do nome e do CPF.

Ao clicar no botão, os valores dos campos são setados no *bean apoiador* e o método *apoiar()* do *bean projeto* é executado. 57

# JSF

```
@ManagedBean
@SessionScoped
public class Apoiador {
    private String nome;
    private String cpf;

    public Apoiador() {}

    public Apoiador(String nome, String cpf) {
        this.nome = nome;
        this.cpf = cpf;
    }

    public String getNome() { return nome; }

    public void setNome(String nome) { this.nome = nome; }

    public String getCpf() { return cpf; }

    public void setCpf(String cpf) { this.cpf = cpf; }
}
```

O bean *Apoiador* tem escopo de sessão e possui atributos e os respectivos *getters* e *setters* para armazenar o nome e o CPF de um apoiador do projeto.

O bean *Projeto* tem escopo de aplicação.

```
@ManagedBean
@ApplicationScoped
public class Projeto {
    private ArrayList<Apoiador> apoiadores = new ArrayList<Apoiador>();

    public ArrayList<Apoiador> getApoiadores() {
        return apoiadores;
    }

    public String apoiar(Apoiador pessoa) {
        boolean erro = false;
        if (pessoa.getNome() == null || pessoa.getNome().isEmpty()) {
            FacesMessage fm = new FacesMessage("Preencha corretamente o nome.");
            FacesContext.getCurrentInstance().addMessage("nome", fm);
            erro = true;
        }
        if (pessoa.getCpf() == null || pessoa.getCpf().isEmpty()) {
            FacesMessage fm = new FacesMessage("Preencha corretamente o CPF.");
            FacesContext.getCurrentInstance().addMessage("cpf", fm);
            erro = true;
        }
        if (erro)
            return null;
        apoiadores.add(new Apoiador(pessoa.getNome(), pessoa.getCpf()));
        return "agradecimento";
    }
}
```

Ele mantém na memória da aplicação uma lista com os apoiadores do projeto.

O método *apoiar()* verifica se todos os campos foram preenchidos. Em caso de não preenchimento, mensagens de erro serão exibidas na página index.

Se os campos tiverem sido preenchidos corretamente, os dados do apoiador serão adicionados à lista de apoiadores e a página de agradecimento será exibida.

# JSF

O *facelet* `agradecimento.xhtml` exibe uma mensagem de agradecimento pelo apoio ao projeto.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:c="http://java.sun.com/jsp/jstl/core">
  <h:head>
    <title>Projeto de Lei de Iniciativa Popular</title>
  </h:head>
  <h:body>
    <h1>Projeto de Lei de Iniciativa Popular</h1>
    <p>Prezado(a) <h:outputText value="#{apoiodor.nome}" />,
    agradecemos pelo seu apoio.</p>
    <p>Segue abaixo a lista de apoiadores do projeto:</p>
    <ol>
      <c:forEach var="pessoa" items="#{projeto.apoiadores}" >
        <li><h:outputText value="#{pessoa.nome}" /></li>
      </c:forEach>
    </ol>
  </h:body>
</html>
```

No final da página é exibida uma lista com os nomes dos apoiadores do projeto.

# JSF

Projeto de Lei de Iniciativa Popular

Solicitamos o seu apoio para o projeto de lei de iniciativa popular que torna a segunda e a terça-feira de carnaval feriados nacionais.

O texto do projeto está disponível [aqui](#).

Entre com os seus dados no formulário abaixo e apoie o projeto.

Nome:

CPF:

• Preencha corretamente o CPF.

Projeto de Lei de Iniciativa Popular

Prezado(a) Beltrano, agradecemos pelo seu apoio.

Segue abaixo a lista de apoiadores do projeto:

1. Fulano
2. Beltrano

# EJB

- *Enterprise JavaBeans*
  - Integra um modelo de componentes de negócio à arquitetura Java EE
  - Cria uma camada composta de *beans* especializados, não-gráficos
  - *Beans* rodam em servidores Java EE
- Componentes EJB
  - São objetos Java escaláveis e reutilizáveis
  - Utilizam anotações/arquivos XML para informar ao contêiner como devem ser gerenciados

# EJB

- Comunicação
  - EJBs interagem com clientes remotos através de interfaces/Beans anotados com `@Remote`
  - *Beans* podem ser acessados remotamente por:
    - Aplicações Java usando RMI/IIOP
    - Aplicações CORBA usando IIOP
    - Clientes Web via páginas JSP ou *Servlets*
  - Clientes locais podem interagir com os EJBs utilizando injeção de dependência ou interfaces/ *beans* anotados com `@Local`

# EJB

- Tipos de *Enterprise Beans*
  - *Session Beans*
    - Executam uma tarefa durante uma sessão de interação entre o *Bean* o cliente
  - *Message-Driven Beans*
    - São consumidores de mensagens JMS
    - Mensagens tratadas ao serem recebidas
  - *Entity Beans*
    - Representam dados armazenados em BDs
    - Persistência transparente



# EJB

- *Session Bean*

- Representam um cliente em particular no servidor
  - ou seja, o *bean* não é compartilhado entre os clientes
- O cliente invoca métodos do *bean* para acessar o servidor – o *bean* age como uma extensão do cliente
- Pode ser acessado remotamente quando possui a anotação `@Remote` na classe do *bean* ou em uma interface que ela implementa

# EJB

- Tipos de *Session Beans*
  - *Stateless Session Bean*
    - Não possui estado que o ligue a um cliente
    - Instâncias diferentes são equivalentes
    - Classe recebe a anotação `@Stateless`
  - *Stateful Session Bean*
    - Armazena estado durante a sessão de um cliente (entre invocações sucessivas)
    - O estado não é persistido (é transiente)
    - Classe recebe a anotação `@Stateful`

# EJB

- *Message-Driven Beans*
  - Consomem mensagens de uma fila ou associadas a um determinado tópico
  - Podem receber mensagens de uma ou de várias fontes
  - Não possuem estado nem interfaces remotas
  - Classe recebe a anotação @MessageDriven
  - Sua interface depende do serviço de mensagens utilizado

# EJB

- *Message-Driven Beans* (cont.)
  - O JMS (*Java Message Service*) é o serviço de mensagens mais usado para implementar *Message-Driven Beans*
  - JMS é implementado por diversos servidores de aplicação Java EE
  - Requer que o *bean* implemente a interface *javax.jms.MessageListener*
  - O método *onMessage()* é executado a cada mensagem recebida, devendo efetuar o tratamento adequado da mensagem

# EJB

- *Entity Beans*
  - São POJOS (*Plain Old Java Objects*)
  - Permitem o acesso compartilhado a dados armazenados em um banco de dados
  - Dados são materializados na forma de objetos (mapeamento objeto-relacional)
    - *Bean* = tabela de um BD relacional
    - Instância do *Bean* = linha da tabela
    - Identificador do *Bean* = chave primária
  - A JPA (*Java Persistence API*) é usada para controlar a persistência dos dados da aplicação

# JPA

- *Java Persistence API*
  - Modelo simplificado e leve de persistência
  - Pode ser utilizado tanto em containers JavaEE quanto em aplicações JavaSE
  - Permite utilização de herança e polimorfismo
  - Permite criação de testes independentes do container quando utilizado com JavaEE
  - Possui anotações para definição de mapeamento objeto-relacional
  - Principais implementações da JPA
    - *Hibernate*
    - *Oracle TopLink*
    - *EclipseLink*

# JPA

- Entidade
  - No JPA uma entidade é um objeto comum Java (um POJO) que pode ser gravado pelo mecanismo de persistência
  - Uma classe que representa uma entidade é anotada com `@Entity`
  - Toda entidade deve possuir um construtor sem argumentos
  - Toda entidade deve possuir uma chave primária, simples ou composta, indicada pela anotação `@Id`
  - Chaves compostas devem ser representadas por uma classe Java em separado

# JPA

- Anotações de Mapeamento
  - Gerenciam o mapeamento de entidades para o banco de dados, definem restrições, consultas, ...
    - @Table
    - @Column
    - @Id
    - @NotNull
    - @Size
    - @GeneratedValue
    - @NamedQueries
    - ...



# JPA

- Anotações de Relacionamento
  - Especificam relacionamentos entre entidades e determinam a cardinalidade da relação
    - @OneToOne
    - @OneToMany
    - @ManyToOne
    - @ManyToMany

# JPA

- *Entity Manager*
  - Controla o ciclo de vida das entidades
  - Possui métodos para buscar, salvar, remover e atualizar estado das entidades
  - Referência para o *Entity Manager* é obtida com injeção de dependência, utilizando a anotação `@PersistenceContext`
  - Principais métodos:
    - **`persist()`**: persiste uma instância de entidade
    - **`refresh()`**: atualiza estado da entidade a partir do BD
    - **`remove()`**: remove uma instância da entidade
    - **`create[Named,Native]Query()`**: criam consultas no BD

# JPA

- Exemplo de uso da JPA

```
@Entity
@Table(name = "APOIADORES")
@NamedQueries({
    @NamedQuery(name = "Apoiadores.findAll",
        query = "SELECT a FROM Apoiadores a")})
public class Apoiadores implements Serializable {
    @Size(max = 60) @Column(name = "NOME")
    private String nome;
    @Id @NotNull @Size(max = 14) @Column(name = "CPF")
    private String cpf;

    public Apoiadores() {}
    public Apoiadores(String cpf) { this.cpf = cpf; }

    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getCpf() { return cpf; }
    public void setCpf(String cpf) { this.cpf = cpf; }
}
```

A entidade *Apoiadores* é mapeada para a tabela com o mesmo nome, que é mantida no banco de dados.

Os atributos são mapeados para as colunas da tabela. O CPF é o identificador (chave primária).

```
@Local
public interface SessionBeanLocal {

    void apoiar(String nome, String cpf);
    List<Apoiadores> getApoiadores();
}
```

A interface do EJB de sessão define as assinaturas dos métodos de negócio.

```
@Stateless
public class SessionBean implements SessionBeanLocal {
    @PersistenceContext(unitName = "ProjetoLeiEJB-ejbPU")
    private EntityManager em;

    @Override
    public void apoiar(String nome, String cpf) {
        Apoiadores apoiador = new Apoiadores();
        apoiador.setNome(nome);
        apoiador.setCpf(cpf);
        em.persist(apoiador);
    }

    @Override
    public List<Apoiadores> getApoiadores() {
        return em.createNamedQuery("Apoiadores.findAll").getResultList();
    }
}
```

O EJB de sessão implementa a interface.

Ele possui uma referência para o gerenciador de entidades.

O método *apoiar()* persiste os dados do apoiador no banco de dados.

O método *getApoiadores()* recupera a lista completa de apoiadores do BD.

```

@ManagedBean
@ApplicationScoped
public class Projeto {
    @EJB
    ejb.SessionBeanLocal session;

    public String apoiar(Apoiador pessoa) {
        boolean erro = false;
        if (pessoa.getNome() == null || pessoa.getNome().isEmpty()) {
            FacesMessage fm = new FacesMessage("Preencha corretamente o nome.");
            FacesContext.getCurrentInstance().addMessage("nome", fm);
            erro = true;
        }
        if (pessoa.getCpf() == null || pessoa.getCpf().isEmpty()) {
            FacesMessage fm = new FacesMessage("Preencha corretamente o CPF.");
            FacesContext.getCurrentInstance().addMessage("cpf", fm);
            erro = true;
        }
        if (erro)
            return null;
        session.apoiar(pessoa.getNome(), pessoa.getCpf());
        return "agradecimento";
    }

    public List<ejb.Apoiadores> getApoiadores() {
        return session.getApoiadores();
    }
}

```

A partir da camada Web, um *bean* JSP pode obter uma referência para o EJB de sessão e acessar a camada de negócios. Note que, diferentemente do exemplo anterior, os dados não são mantidos na camada Web.

O método *apoiar()* valida os dados e invoca o método do EJB de sessão que faz a persistência.

O método *getApoiadores()* recupera a lista de apoiadores a partir do EJB de sessão.

# Segurança

- Autenticação
  - A especificação do Java EE não define como a autenticação de usuários é realizada
  - As seguintes APIs do Java podem ser usadas:
    - JNDI (*Java Naming and Directory Interface*)
    - JAAS (*Java Authentication and Authorization Service*)
  - Alguns servidores de aplicação fornecem mecanismos proprietários de autenticação
  - Mecanismos de autenticação podem ser implementados na própria aplicação

# Segurança

- Autorização
  - Baseada em papéis (*roles*): grupos funcionais de usuários; ex.: operador, gerente, supervisor, etc.
  - Definida através de anotações no código do *bean*
    - `@RolesAllowed("PAPEL-1", ...)`: define quais papéis são exigidos para execução de um método ou *bean*
    - `@PermitAll`: pode ser executado por qualquer papel
    - `@RunAs("PAPEL-3")`: especifica o papel com o qual um método/*bean* invocará métodos de outros *beans*
  - Exceção *EJBAccessException* indica que um método não-autorizado foi invocado

# Segurança

- Autorização (cont.)
  - Os papéis são especificados no arquivo ejb-jar.xml

```
<ejb-jar version="3.0">  
  <assembly-descriptor>  
    ...  
    <security-role>  
      <description>Auditor do sistema</description>  
      <role-name>AUDITOR</role-name>  
    </security-role>  
    ...  
  </assembly-descriptor>  
</ejb-jar>
```



# Segurança

- Autorização (cont.)
  - O arquivo ejb-jar.xml também pode definir os papéis exigidos para execução de métodos

```
...  
<method-permission>  
  <role-name>AUDITOR</role-name>  
  <method>  
    <ejb-name>MeuBean</ejb-name>  
    <method-name>auditData</method-name>  
  </method>  
</method-permission >  
...
```