

## Unidade III

### Componentes no Java EE

- Java EE
- JSF
- EJB
- JPA

## Java EE

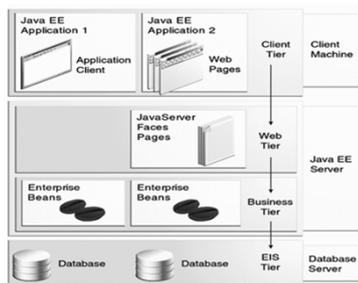


- Plataforma Java Enterprise Edition
  - Adiciona ao Java suporte para:
    - Desenvolvimento de Aplicações Web: JSP, *Servlets* e JSF
    - Componentes de Negócio: EJB
    - Interconexão com Sistemas Legados: Java *Connectors*
  - Fornece diversas APIs para:
    - Comunicação: JMS, JavaMail
    - Gerenciamento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

2

## Java EE

### Arquitetura do Java EE



© <http://java.sun.com>

3

## 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

4

## 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

5

## Java EE

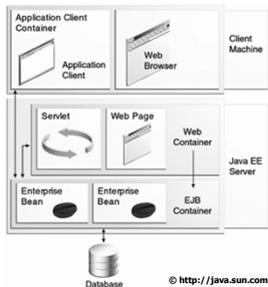
### Servidor Java EE

- 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

6

## Java EE

### ■ Contêineres do Servidor Java EE



7

## 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

8

## 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*

9

## 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)

10

## 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

11

## 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*)

12

## 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

13

## 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

14

## 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

15

## 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

16

## Java EE

- Implantação de aplicações corporativas
  - Arquivos EAR são carregados no servidor Java EE, que abre o pacote e coloca a aplicação em execução
    - Conteúdo de arquivos é WAR implantado no contêiner Web
    - Componentes EJB contidos nos 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

17

## 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

18

## JSF

- Componentes JSF
  - O JSF fornece um conjunto de componentes comumente usados em páginas Web: link, tabela, botão, combo box, caixa 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

19

## JSF

- Modelo de eventos
  - Uma ação na página Web – como clicar um botão ou selecionar uma opção em um *combo box* – 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*

20

## JSF

- *Facelets*
  - Forma padrão de implementar *views* no JSF2.0
  - Baseado em 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

21

## JSF

- Escopo dos *Beans* Gerenciados JSF

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

22

## JSF

- Exemplo: *Facelet* index.xhtml

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core">
  <head>
    <title>Adivinhe o número</title>
  </head>
  <body>
    <h5>Entre com um número de 0 a 9:</h5>
    <h:form>
      <h:inputText id="userNumber" size="2"
        maxlength="2" value="#{UserNumberBean.userNumber}" />
      <h:commandButton id="submit" value="Enviar"
        action="response" />
    </h:form>
  </body>
</html>
```

O *facelet* possui um formulário com um campo de texto, no qual deve ser digitado um número. Ao clicar no botão Enviar, o número é armazenado no atributo *userNumber* de *UserNumberBean* e o navegador exibe o *facelet response*.

23

## JSF

- Exemplo: *Facelet* response.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
  <head>
    <title>Resposta</title>
  </head>
  <body>
    <h4>
      <h:outputText escape="false" value=
        "#{UserNumberBean.response}"/>
    </h4>
    <h:form>
      <h:commandButton id="back" value="Voltar" action="index" />
    </h:form>
  </body>
</html>
```

O *facelet* de resposta tem um campo de texto no qual é mostrado o valor retornado pelo método *getResponse* do *bean* gerenciado. O botão *Voltar* retorna ao *facelet index*.

24

## JSF

### ■ Exemplo: *Bean* gerenciado *UserNumberBean*

```
@ManagedBean(name="UserNumberBean")
@SessionScoped // sessão --> uma instância por usuário
public class UserNumberBean {
    Integer randomInt; // armazena número sorteado
    Integer userNumber; // armazena número digitado pelo usuário

    public UserNumberBean() { // construtor; sorteia número aleatório
        Random randomGR = new Random();
        randomInt = new Integer(randomGR.nextInt(10));
        System.out.println("Número sorteado: " + randomInt);
    }

    public void setUserNumber(Integer userNumber) {
        this.userNumber = userNumber; // armazena o número digitado
    }
    ...
}
```

25

## JSF

```
...
public Integer getUserNumber() {
    return userNumber; // retorna o último número digitado
}

public String getResponse() {
    if ((userNumber != null) && (userNumber.compareTo(randomInt)
        == 0)) { // se número digitado == gerado
        FacesContext context = FacesContext.getCurrentInstance();
        HttpSession session = (HttpSession)context.getExternalContext().
            getSession(false); // obtém o objeto de sessão
        session.invalidate(); // destrói a sessão
        return "Você acertou!"; // resposta = acertou
    } else { // resposta = errou
        return "<p>Desculpe, o número não é " + userNumber + "</p>"
            + "<p>Tente novamente</p>";
    }
}
}
```

26

## 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

27

## 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

28

## EJB

### ■ Tipos de *Enterprise Beans*

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

29

## 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

30

## EJB

- Estado dos *Session Beans*
  - *Stateless Session Bean*
    - Não possui estado que o ligue a um cliente
    - Instâncias diferentes são equivalentes se inativas
  - *Stateful Session Bean*
    - Armazena estado durante a sessão de um cliente (entre invocações sucessivas)
    - O estado não é persistido (é transiente)

31

## EJB

- Exemplo de *Stateless Session Bean*

```
@Stateless
public class HelloWorldSessionBean{
    public String hello(){
        return "Hello World";
    }
}
```

- A anotação `@Stateless` indica que a classe anotada é um *Stateless Session Bean*

32

## EJB

- *Stateless Session Bean* com suporte a clientes remotos

- Anotando a interface com `@Remote`

```
@Remote
public interface Hello{
    public String hello();
}
```

```
@Stateless
public class HelloWorldSessionBean implements Hello{
    public String hello(){
        return "Hello World";
    }
}
```

33

## EJB

- *Stateless Session Bean* com suporte a clientes remotos

- Anotando classe com `@Remote`

```
public interface Hello{
    public String hello();
}
```

```
@Stateless
@Remote(Hello.class)
public class HelloWorldSessionBean implements Hello{
    public String hello(){
        return "Hello World";
    }
}
```

34

## EJB

- Ciclo de vida dos *Stateless Session Beans*



35

## EJB

- Exemplo de *Stateful Session Bean*

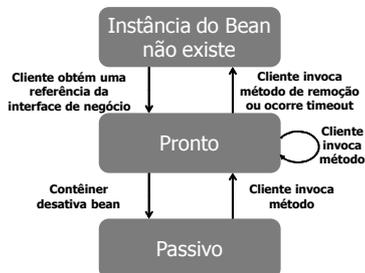
```
@Stateful
@Remote(Count.class)
public class CountBean implements Count {
    private int count;
    public int getCount() { return this.count; }
    public void setCount(int count) { this.count = count; }
    @Remove
    public void remove() {}
}
```

- A anotação `@Stateful` indica que a classe é um *Stateful Session Bean*
- A anotação `@Remove` indica que o *bean* deve ser removido após a execução do método anotado

36

## EJB

### ■ Ciclo de vida dos *Stateful Session Beans*



37

## EJB

### ■ *Message-Driven Beans*

- Consomem mensagens de uma fila ou associadas a um determinado tópico
- Podem receber mensagens de uma ou de múltiplas fontes
- Não possuem estado nem interfaces remotas
- Beans são anotados com `@MessageDriven`
- Sua interface depende do serviço de mensagens utilizado
  - Geralmente é usado JMS (*Java Message Service*)

38

## EJB

### ■ Exemplo de *Message-Driven Bean*

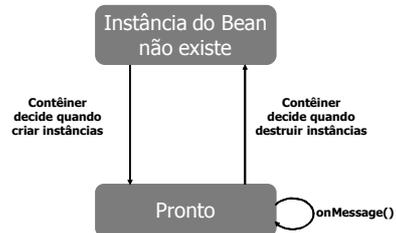
```
@MessageDriven(mappedName="jms/Queue")
public class SimpleMessageBean implements MessageListener {
    public void onMessage (Message msg) {
        // utiliza a mensagem
    }
}
```

- O *bean* utiliza o JMS, que requer a implementação da interface `javax.jms.MessageListener`
- A fila de mensagens especificada na anotação `@MessageDriven` é mantida num provedor JMS

39

## EJB

### ■ Ciclo de vida dos *Message-Driven Beans*



40

## EJB

### ■ *Entity Beans*

- São POJOS (*Plain Old Java Objects*)
- Permitem o acesso compartilhado a dados armazenados em um BD
- 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
- Utilizam a JPA (*Java Persistence API*) para controlar a persistência dos dados

41

## JPA

### ■ *Java Persistence API*

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

42

## 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, identificada pela anotação `@Id`
  - Chaves compostas devem ser representadas por uma classe Java em separado

43

## JPA

### ■ Exemplo de Entidade JPA

```
@Entity
public class Conta {
    @Id
    private Long numConta;
    private String nomeTitular;
    private long saldo;
    public Conta() {} //Construtor sem argumentos é obrigatório
    public void setNumConta(Long numConta) {
        this.numConta = numConta;
    }
    public Long getNumConta() { return this.numConta; }
    //Métodos getters e setters para os outros atributos
}
```

44

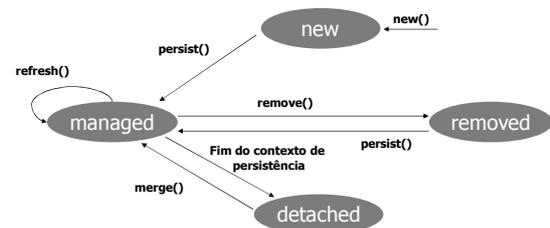
## JPA

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

45

## JPA

### ■ Ciclo de vida das entidades



46

## JPA

- Estados
  - **new**: Estado da entidade após ser criada.
  - **managed**: Entidade persistida, com id associado a um contexto de persistência.
  - **removed**: marcada para ser removida do BD.
  - **detached**: Entidade possui um id persistente mas não possui um contexto de persistência.
- Operações
  - **new()**: Cria nova entidade
  - **persist()**: Persiste uma entidade
  - **refresh()**: Atualiza o estado de uma entidade
  - **merge()**: Sincroniza entidade desacoplada

47

## 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 dependências, utilizando a anotação `@PersistenceContext`

48