

# Desenvolvimento de Aplicações Corporativas com a Plataforma Java EE

Minicurso da 3ª SEMINFO



Antônio Vinícius Menezes Medeiros  
vinyanalista@gmail.com



# Roteiro

- Introdução à Plataforma Java EE
- A aplicação de exemplo "Agenda de Contatos"
- Apresentação das ferramentas utilizadas
- Persistência com JPA
- Validação de dados com Bean Validation
- Lógica de negócios com EJB
- Apresentação com JSF
- Implementação de um CRUD com JSF

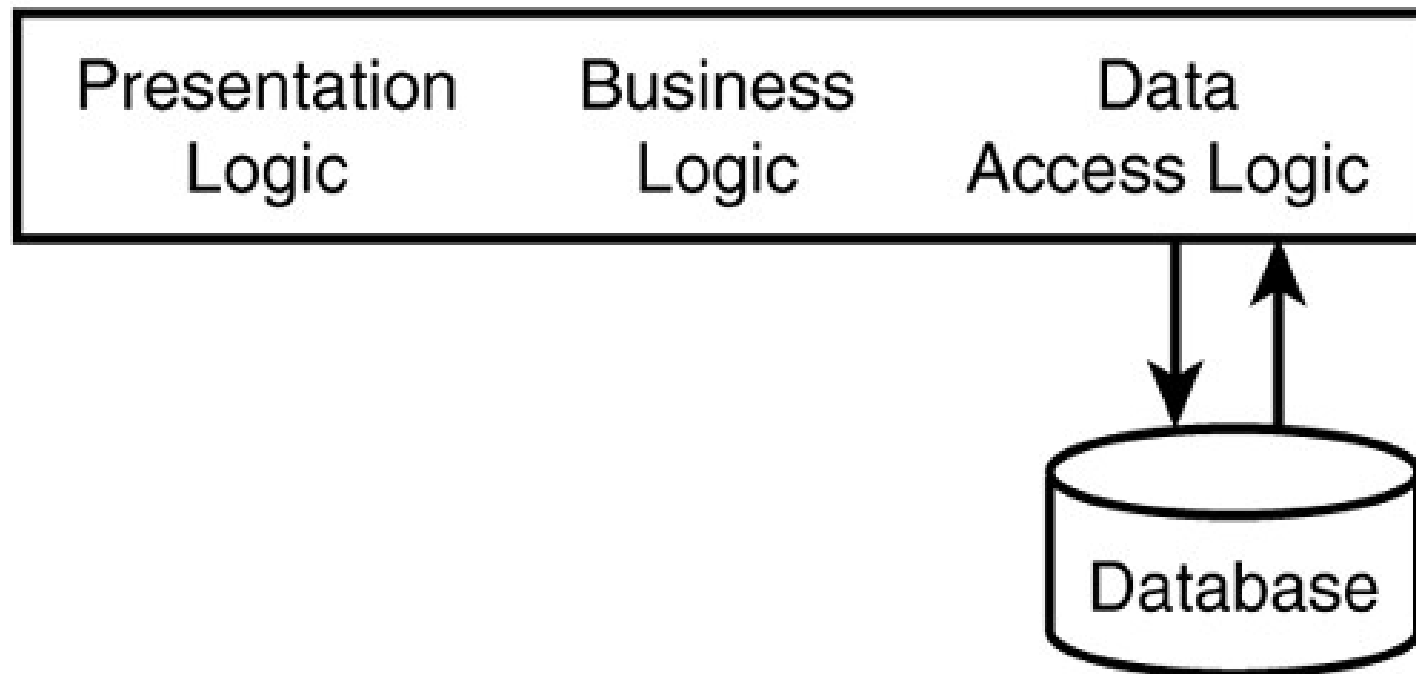
# Roteiro

## ➔ **Introdução à Plataforma Java EE**

- A aplicação de exemplo "Agenda de Contatos"
- Apresentação das ferramentas utilizadas
- Persistência com JPA
- Validação de dados com Bean Validation
- Lógica de negócios com EJB
- Apresentação com JSF
- Implementação de um CRUD com JSF

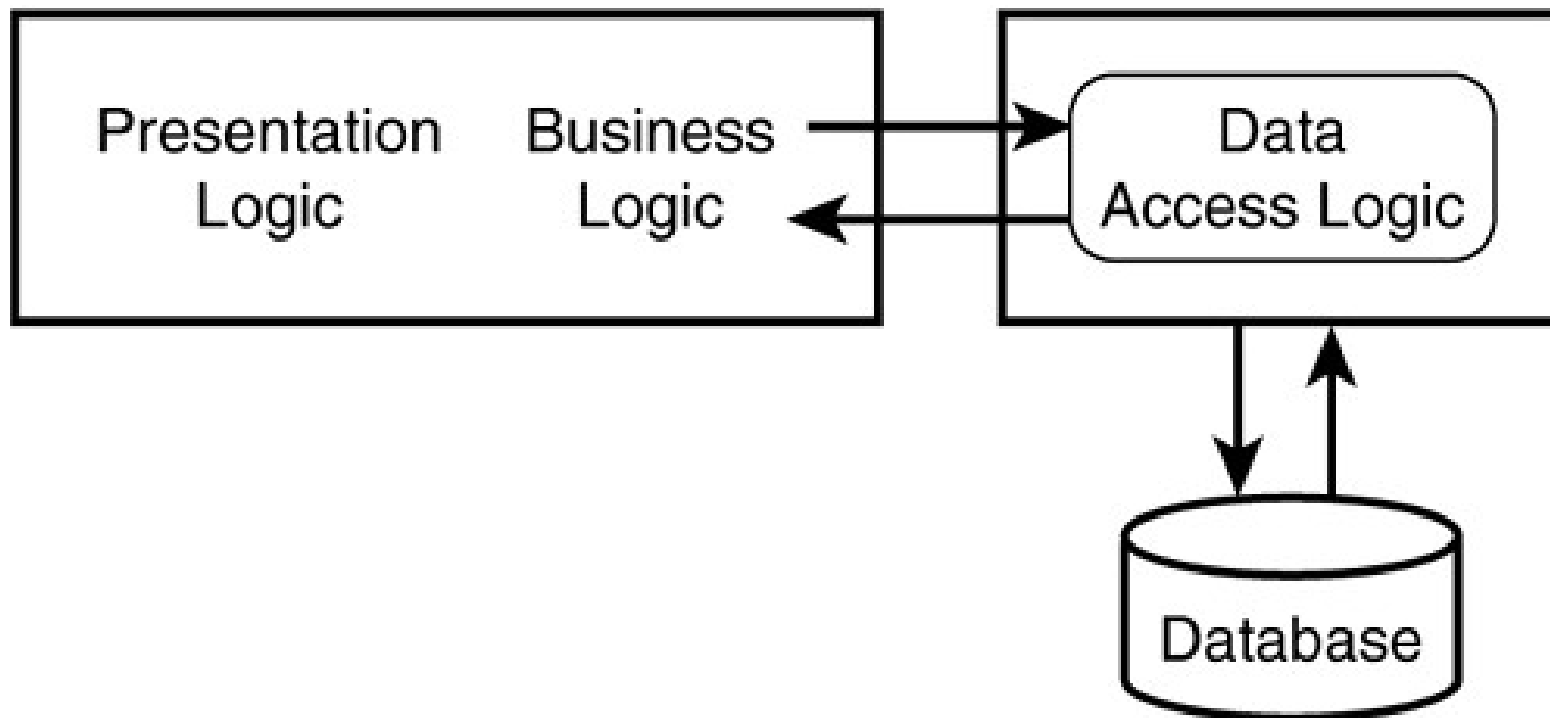
# Aplicações em camadas

- Aplicações monolíticas
  - Apenas uma camada



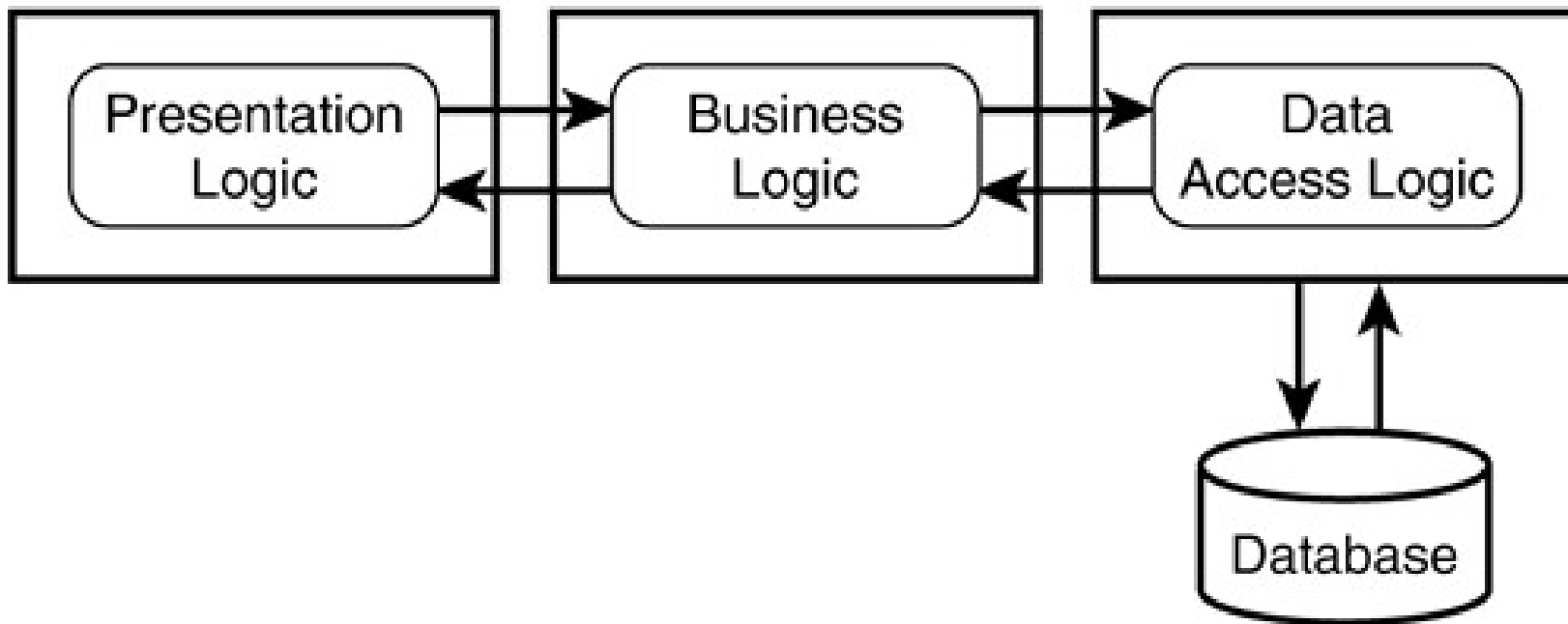
# Aplicações em camadas

- Aplicações em duas camadas



# Aplicações em camadas

- Aplicações em três camadas



# Aplicações em camadas

- Aplicações em  $n$  camadas
  - Mais camadas podem se mostrar necessárias durante o projeto do software
  - Vantagens
    - Melhor distribuição de tarefas
    - Utilização de mais máquinas
  - Desvantagens
    - Implementar a comunicação entre as camadas
    - Perda de performance



# Aplicações em camadas

- Modularização
  - Reduz a complexidade dos programas
  - Redução da dependência entre várias partes do programa
    - Código fracamente acoplado
  - Facilita a manutenção e a evolução do sistema





# Aplicações em camadas

- **Componentes:** unidades de funcionalidade que podem ser usadas em determinado *framework*
  - Facilitam o desenvolvimento, a manutenção e a evolução do *software*
  - Maior independência do *software*
  - Permitem utilização de código de terceiros
    - Redução no custo e no tempo para desenvolver
  - Permitem reutilização de código



# Aplicações corporativas (*Enterprise Applications*)

- Aplicações modernas
  - Apresentam  $n$  camadas
  - Baseadas em componentes
  - Distribuídas
  - Escaláveis
  - Disponíveis "a tempo e a hora"
  - Integradas (em sua maioria) à Internet

# Aplicações corporativas (*Enterprise Applications*)

- Requisitos de um moderno ambiente computacional distribuído:
  - Ciclo de vida
  - Persistência
  - Atribuição de nomes
  - Transações
  - Segurança

# Aplicações corporativas (*Enterprise Applications*)

- Requisitos de um moderno ambiente computacional distribuído:
  - Segurança
    - Autenticação
      - Você é quem diz ser?
    - Autorização
      - Você tem permissão para fazer o que pretende?
    - Confidencialidade
      - Alguém, além de você, pode ver seus dados?
    - Integridade
      - Você está vendo o que está realmente armazenado?

# Aplicações corporativas (*Enterprise Applications*)

- Destacam-se dois *frameworks* visando o desenvolvimento dessas aplicações:

- **.NET**



- Desenvolvido exclusivamente pela Microsoft
    - Disponível apenas para soluções Microsoft
    - Independente de linguagem, em tese

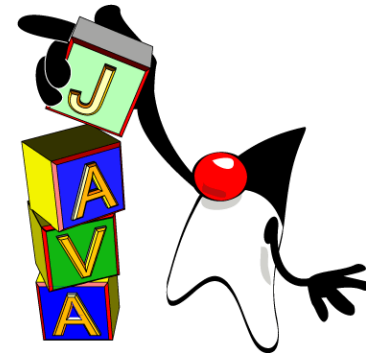
- **Java EE**



- Desenvolvimento coordenado pela Oracle
    - Independente de arquitetura
    - Usa primariamente a linguagem Java

# Plataforma Java EE

- O termo **Java** pode se referir:
  - A uma **linguagem de programação** de alto nível orientada a objetos, que possui uma sintaxe e estilo particulares; ou
  - A uma **plataforma** na qual aplicativos desenvolvidos usando a linguagem Java podem ser executados.
    - Java SE
    - Java EE
    - Java ME
    - JavaFX



# Plataforma Java EE

- **Plataforma Java SE:** fornece os tipos e objetos básicos da linguagem
  - Máquina virtual e ferramentas de desenvolvimento
  - Acesso a arquivos e bancos de dados
  - Rede
  - Segurança
  - Interface gráfica (GUI)
- Versão atual: 7.0



# Plataforma Java EE

- **Plataforma Java EE:** fornece um ambiente comum para a construção de aplicativos corporativos
  - Extensão do Java SE
  - Padronizada por uma especificação
  - Aplicações portáteis, independentes da implementação
- Versão atual: 6.0





# Especificações e implementações de referência (RIs)

- A plataforma Java EE é definida por uma especificação, que traz consigo um conjunto de especificações
  - Java Community Process (JCP)
  - Java Specification Request (JSR)
- Cada especificação deve apresentar uma implementação de referência (RI)
- Implementações devem apresentar, no mínimo, os recursos da RI



# Especificações e implementações de referência (RIs)

- Especificação Java EE

| Especificação | Versão | JSR | RI        | Versão |
|---------------|--------|-----|-----------|--------|
| Java EE       | 6.0    | 316 | GlassFish | 3.1.2  |



- Especificações *web*

| Especificação       | Versão | JSR | RI             | Versão |
|---------------------|--------|-----|----------------|--------|
| JSF                 | 2.0    | 314 | Mojara         | 2.1.16 |
| JSP                 | 2.2    | 245 | GlassFish JSP  | 2.2.6  |
| JSTL                | 1.2    | 52  | GlassFish JSTL | 1.2.1  |
| Servlet             | 3.0    | 315 | GlassFish      | 3.1.2  |
| Expression Language | 1.2    | 245 | GlassFish EL   | 2.2.4  |

# Especificações e implementações de referência (RIs)

- Especificações corporativas

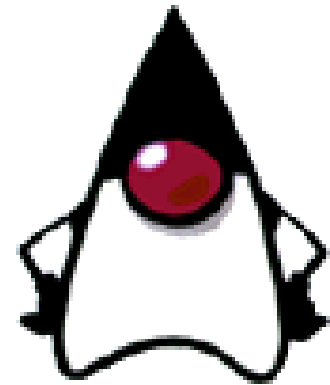
| Especificação | Versão     | JSR        | RI                 | Versão       |
|---------------|------------|------------|--------------------|--------------|
| <b>EJB</b>    | <b>3.1</b> | <b>318</b> | <b>GlassFish</b>   | <b>3.1.2</b> |
| JAF           | 1.1        | 925        | GlassFish JAF      | 1.0          |
| JavaMail      | 1.4        | 919        | Kenai Project      | 1.4.5        |
| JCA           | 1.6        | 322        |                    |              |
| JMS           | 1.1        | 914        | Open MQ            | 4.5.2        |
| <b>JPA</b>    | <b>2.0</b> | <b>317</b> | <b>EclipseLink</b> | <b>2.4.1</b> |
| <b>JTA</b>    | <b>1.1</b> | <b>907</b> |                    |              |



# Especificações e implementações de referência (RIs)

- Especificações de *webservices*

| Especificação         | Versão | JSR | RI             | Versão  |
|-----------------------|--------|-----|----------------|---------|
| JAX-RPC *             | 1.1    | 101 |                |         |
| JAX-WS                | 2.2    | 224 | Metro          | 2.2.1-1 |
| JAXB                  | 2.2    | 222 | Glassfish JAXB | 2.2.6   |
| JAXM                  | 1.0    | 67  | Glassfish SAAJ | 1.3.20  |
| StAX                  | 1.0    | 173 | Sjsxp          | 1.0.2   |
| Web Services          | 1.2    | 109 |                |         |
| Web Services Metadata | 1.1    | 181 |                |         |
| JAX-RS                | 1.0    | 311 | Jersey         | 1.16    |
| JAX-R *               | 1.1    | 93  |                |         |



\* Especificação depreciada



# Especificações e implementações de referência (RIs)

- Outras especificações

| Especificação                                                 | Versão     | JSR        | RI                         | Versão       |
|---------------------------------------------------------------|------------|------------|----------------------------|--------------|
| JACC                                                          | 1.1        | 115        |                            |              |
| <b>Bean Validation</b>                                        | <b>1.0</b> | <b>303</b> | <b>Hibernate Validator</b> | <b>4.3.1</b> |
| Common Annotations                                            | 1.0        | 250        |                            |              |
| Java EE Application Deployment *                              | 1.2        | 88         |                            |              |
| Java EE Management *                                          | 1.1        | 77         |                            |              |
| Java Authentication Service Provider Interface for Containers | 1.0        | 196        |                            |              |
| Debugging Support for Other Languages                         | 1.0        | 45         |                            |              |



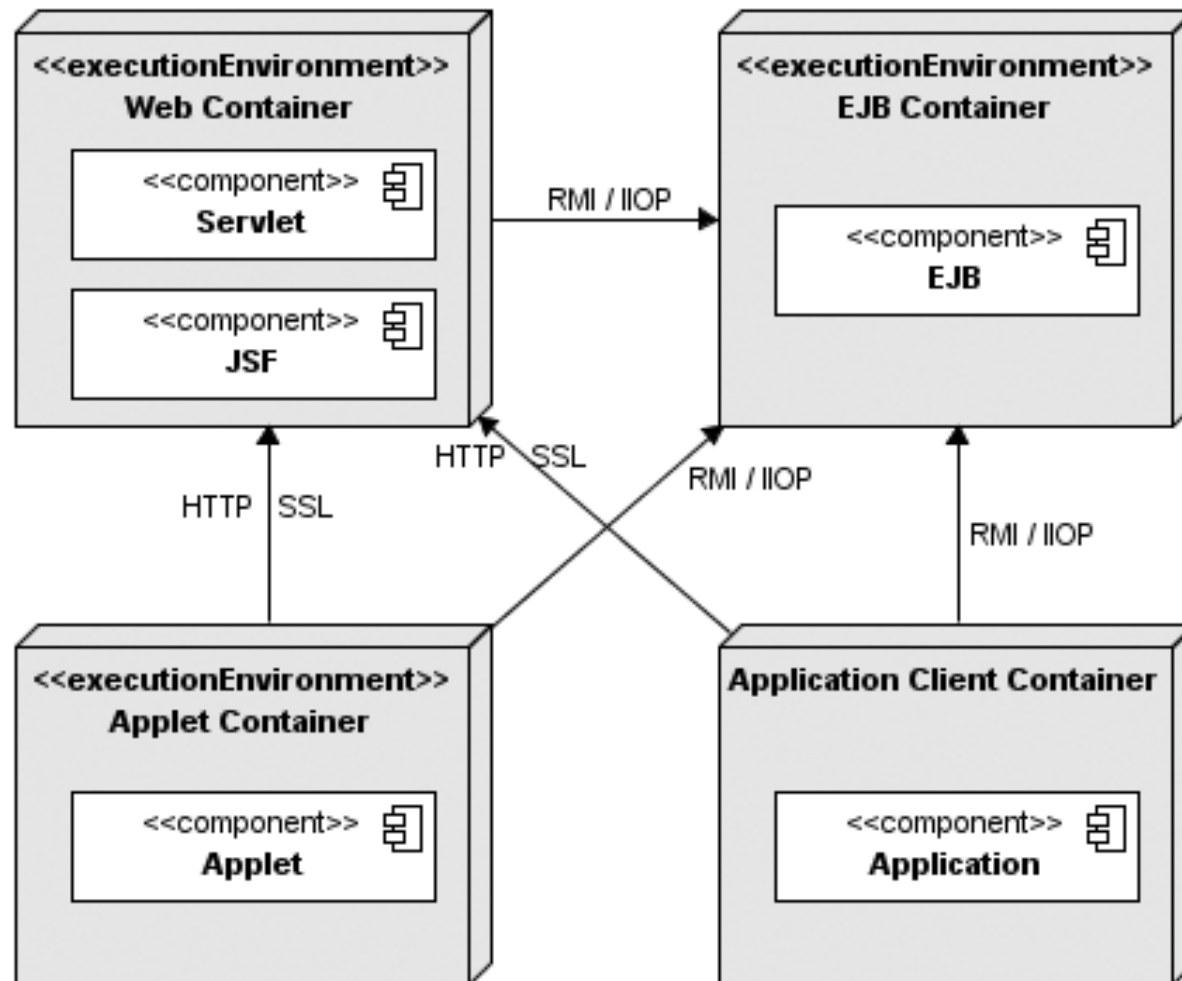
\* Especificação depreciada

# Contêineres e servidores de aplicação

- **Contêineres:** divisões lógicas do ambiente em tempo de execução das aplicações Java EE
  - Possuem regras específicas
  - Suportam determinadas APIs
  - Provêem certos serviços aos seus componentes
    - Gerenciamento do ciclo de vida, comunicação entre componentes, persistência, descoberta de serviços, etc.

# Contêineres e servidores de aplicação

- Contêineres

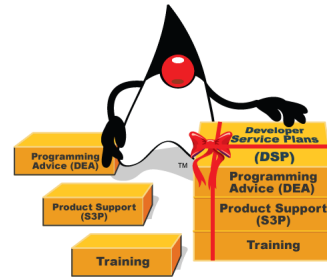


- # Contêineres e servidores de aplicação



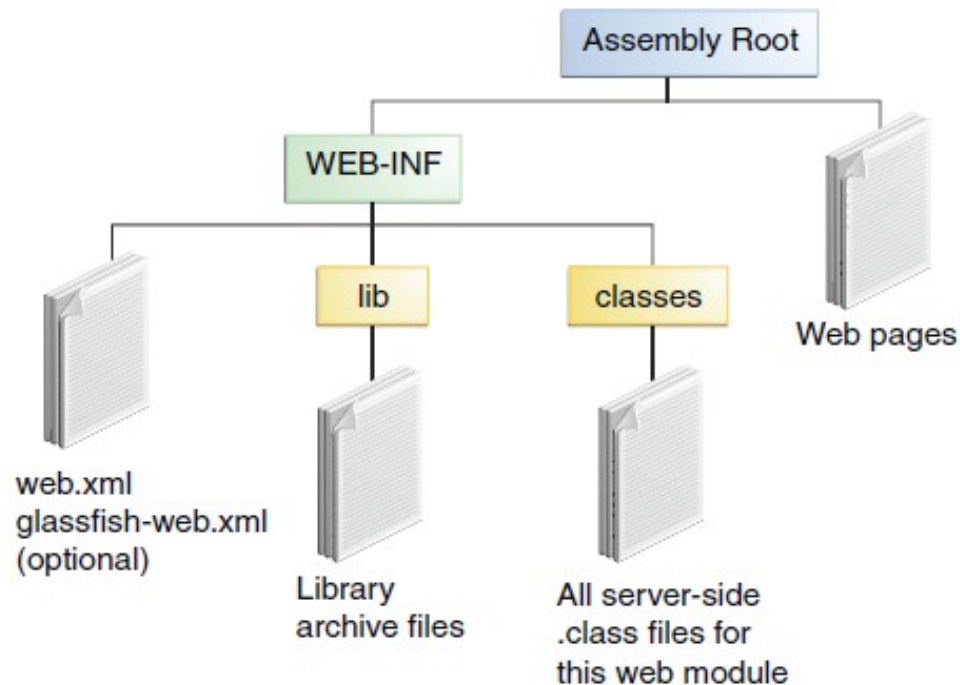
# Contêineres e servidores de aplicação

- Empacotamento
  - **Java SE:** Java Archive (JAR), extensão .jar
    - Aplicações
    - Bibliotecas
  - **Java EE:** JAR, WAR ou EAR
    - Um ou mais componentes Java EE para um mesmo contêiner
    - Opcionalmente, arquivos XML de configuração para aquele contêiner



# Contêineres e servidores de aplicação

- Empacotamento
  - Java EE:
    - Web Archive (WAR): arquivo JAR padrão com extensão .war

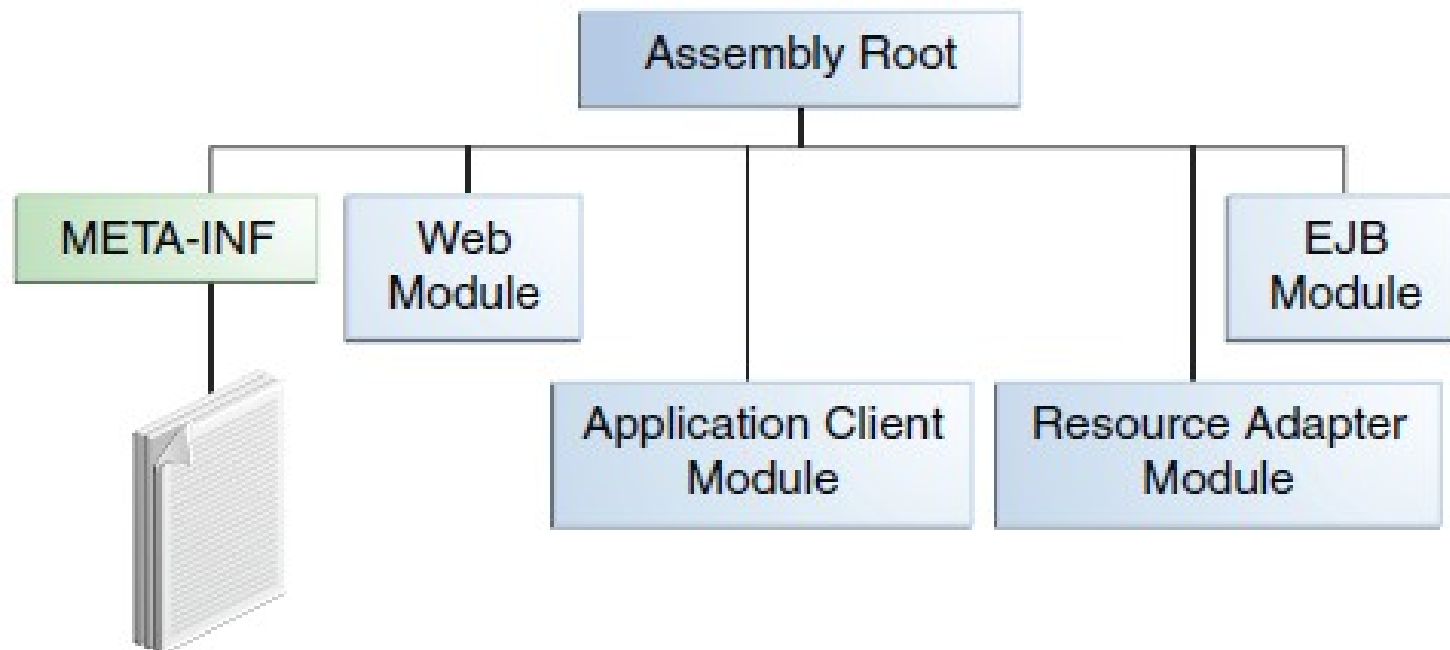
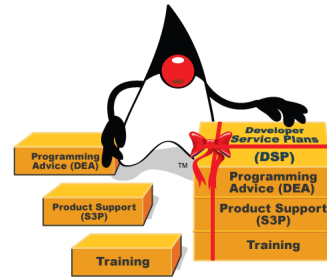


# Contêineres e servidores de aplicação

- Empacotamento

- Java EE:

- Enterprise Archive (EAR): arquivo JAR padrão com extensão .ear



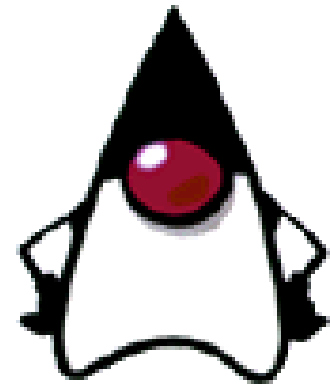
# Contêineres e servidores de aplicação

- **Servidores de aplicação:** fornecem contêineres nos quais os componentes (e a aplicação) serão executados
- **Profiles:**
  - **Web Profile:** implementa as especificações mais usadas pelas aplicações *web*
  - **Full Profile:** implementa todas as 28 especificações exigidas pela Java EE 6

# Contêineres e servidores de aplicação

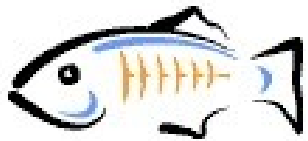
- Especificações do Web Profile

| Especificação       | Versão | JSR | RI             | Versão |
|---------------------|--------|-----|----------------|--------|
| JSF                 | 2.0    | 314 | Mojara         | 2.1.16 |
| JSP                 | 2.2    | 245 | GlassFish JSP  | 2.2.6  |
| JSTL                | 1.2    | 52  | GlassFish JSTL | 1.2.1  |
| Servlet             | 3.0    | 315 | GlassFish      | 3.1.2  |
| Expression Language | 1.2    | 245 | GlassFish EL   | 2.2.4  |
| EJB Lite            | 3.1    | 318 | GlassFish      | 3.1.2  |
| JPA                 | 2.0    | 317 | EclipseLink    | 2.4.1  |
| JTA                 | 1.1    | 907 |                |        |
| Common Annotations  | 1.0    | 250 |                |        |



# Contêineres e servidores de aplicação

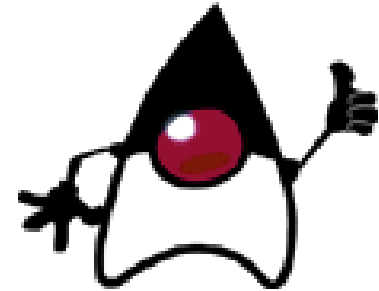
- Servidores de aplicação certificados



GlassFish



APACHE  
GERONIMO



**ORACLE®**  
FUSION MIDDLEWARE  
WEBLOGIC SERVER



Referência: <http://java.sun.com/javaee/overview/compatibility.jsp>

# Roteiro

- ✓ **Introdução à Plataforma Java EE**
- ➔ **A aplicação de exemplo "Agenda de Contatos"**
  - Apresentação das ferramentas utilizadas
  - Persistência com JPA
  - Validação de dados com Bean Validation
  - Lógica de negócios com EJB
  - Apresentação com JSF
  - Implementação de um CRUD com JSF

# A aplicação de exemplo "Agenda de Contatos"

- CRUD de Contatos
  - CRUD (Create, Read, Update and Delete)
- Cada contato possui:
  - ID: número inteiro, sequencial
  - Nome: formado por letras e espaços
  - E-mail (opcional)
  - Telefone (opcional, apenas números)
  - Categoria (opcional)
- Aplicação simples, sem cadastro de usuários



# A aplicação de exemplo "Agenda de Contatos"

## Agenda de Contatos

### Página Inicial

| Contatos |                  |                  |            |  |
|----------|------------------|------------------|------------|--|
| ID       | Nome             | E-mail           | Telefone   |  |
| 1        | Antonio Vinicius | vinyanalista@gm  | 1212341234 |  |
| 2        | Joao             |                  | 5656785678 |  |
| 3        | Jose             | jose@exemplo.com | 1234567890 |  |

 Adicionar  Visualizar  Alterar  Remover

# Roteiro

- ✓ **Introdução à Plataforma Java EE**
- ✓ **A aplicação de exemplo "Agenda de Contatos"**
- **Apresentação das ferramentas utilizadas**
  - Persistência com JPA
  - Validação de dados com Bean Validation
  - Lógica de negócios com EJB
  - Apresentação com JSF
  - Implementação de um CRUD com JSF

# Linux

- AQUI VEM TEXTO

# Linux

- Distribuições Linux = *kernel* + aplicativos

# Linux

- Slax Linux

# Java Development Kit (JDK)



- Conjunto de ferramentas que realizam a compilação, o teste, a depuração e a documentação de programas escritos na linguagem Java
- Traz embutido:
  - Java Runtime Environment (JRE)
  - Versão utilizada: 1.7.0\_45



# Apache Maven



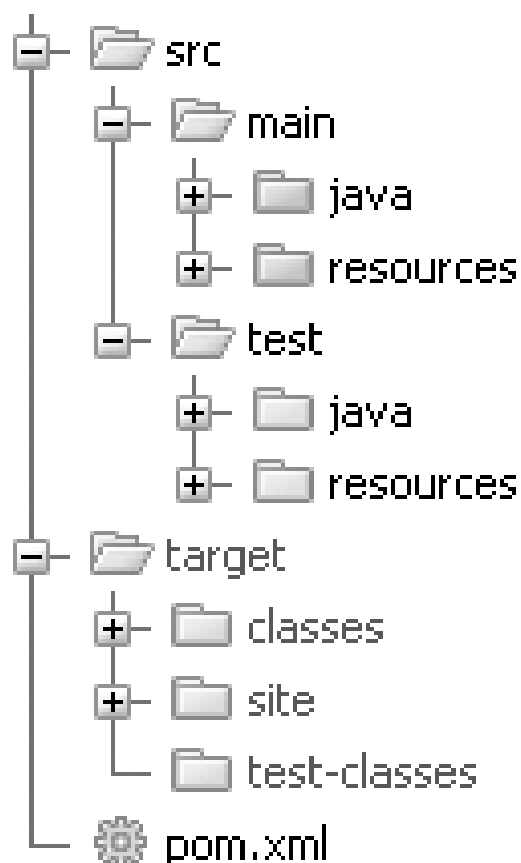
- Ferramenta que automatiza o gerenciamento de projetos em Java
- Auxilia nas etapas de:
  - definição, codificação, compilação, testes, empacotamento, implantação, documentação e distribuição da aplicação.
- Versão utilizada: 3.1.1



# Apache Maven



- Estrutura de diretórios básica de um projeto Maven:





# Apache Maven



- Project Object Model (POM):
  - Arquivo XML que descreve o projeto

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>
    <groupId>br.com.vinyanalista</groupId>
    <artifactId>hello-world</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
```

```
</project>
```

# Apache Maven



- Declaração de dependências
  - Escopos: test, provided, compile, runtime

pom.xml

```
<project>

  <!-- Outras declarações -->

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

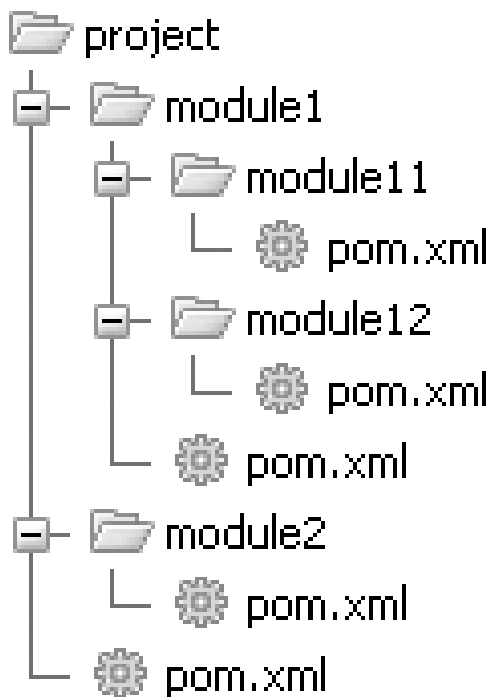
  <!-- Outras declarações -->

</project>
```

# Apache Maven



- Suporte a modularização



pom.xml

```
<project>

    <!-- Outras declarações -->

    <modules>
        <module>module1</module>
        <module>module2</module>
    </modules>

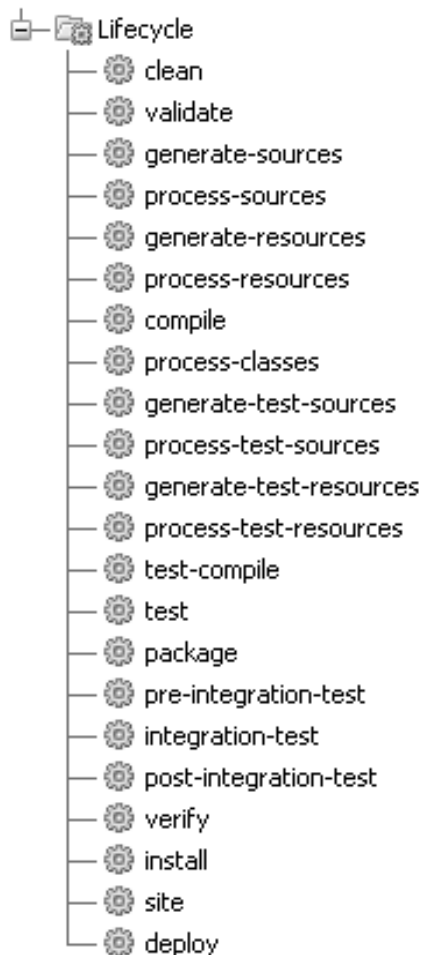
    <!-- Outras declarações -->

</project>
```

# Apache Maven



- Ciclo de vida de um projeto Maven



## pom.xml

```
<project>

    <!-- Outras declarações -->

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
                <inherited>true</inherited>
                <configuration>
                    <source>1.7</source>
                    <optimize>true</optimize>
                    <target>1.7</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <!-- Outras declarações -->

</project>
```

# Apache Maven



- Uso

## Terminal

```
$ mvn clean
```

```
$ mvn compile
```

```
$ mvn test-compile
```

```
$ mvn test
```

```
$ mvn package
```

```
$ mvn install
```

```
$ mvn clean compile package install
```

# Apache Maven



- Projeto de exemplo: Hello World!

HelloWorld.java

```
package br.com.vinyanalista.helloworld;

public class HelloWorld {

    public static String mensagem = "Ola, mundo!";

    public String mensagem2 = "Ola!";

    public static void main(String[] args) {
        System.out.println(mensagem);
    }

}
```

Terminal

```
$ cd codigo/exemplo01
$ mvn clean compile exec:java
```

# JUnit



- Framework que permite a construção e execução de testes de aplicativos desenvolvidos com a linguagem Java.
- Integrado à maioria das IDEs
  - Integrado ao Maven através do plugin Surefire
- Versão utilizada: 4.8.2



- Projeto de exemplo: Hello World!

HelloWorldTest.java

```
package br.com.vinyanalista.helloworld;

import static org.junit.Assert.assertNotNull;

import org.junit.Test;

public class HelloWorldTest {

    @Test
    public void testeDeExemplo() throws Exception {
        HelloWorld hello = new HelloWorld();
        assertNotNull(hello.mensagem2);
    }

}
```

Terminal

```
$ mvn test
```



# JBoss Application Server (JBoss AS)



- Servidor de aplicação de código aberto desenvolvido pela JBoss para a plataforma Java EE
- Certificado pela Oracle nos *profiles* Web e Full Java EE
- Versão utilizada: 7.1.1



# MySQL



- Banco de dados (SGBD) de código aberto desenvolvido pela Oracle
- Se tornou um dos bancos de dados mais utilizados no mundo
  - Integração com PHP
    - Oferecido pela maioria dos serviços de hospedagem de sites
- Versão utilizada: 5.5.27



# XAMPP

- AQUI AINDA VEM TEXTO

# Roteiro

- ✓ **Introdução à Plataforma Java EE**
- ✓ **A aplicação de exemplo "Agenda de Contatos"**
- ✓ **Apresentação das ferramentas utilizadas**
- ➔ **Persistência com JPA**
  - Validação de dados com Bean Validation
  - Lógica de negócios com EJB
  - Apresentação com JSF
  - Implementação de um CRUD com JSF

# Java Persistence API (JPA)

- Bancos de dados relacionais
  - Campo
  - Registro
  - Tabela: formada por linhas (registros) e colunas (campos)
  - Chave primária
  - Chave estrangeira



# Java Persistence API (JPA)

- Linguagens orientadas a objetos
  - Classes
  - Objetos
  - Atributos
  - Métodos
  - Listas



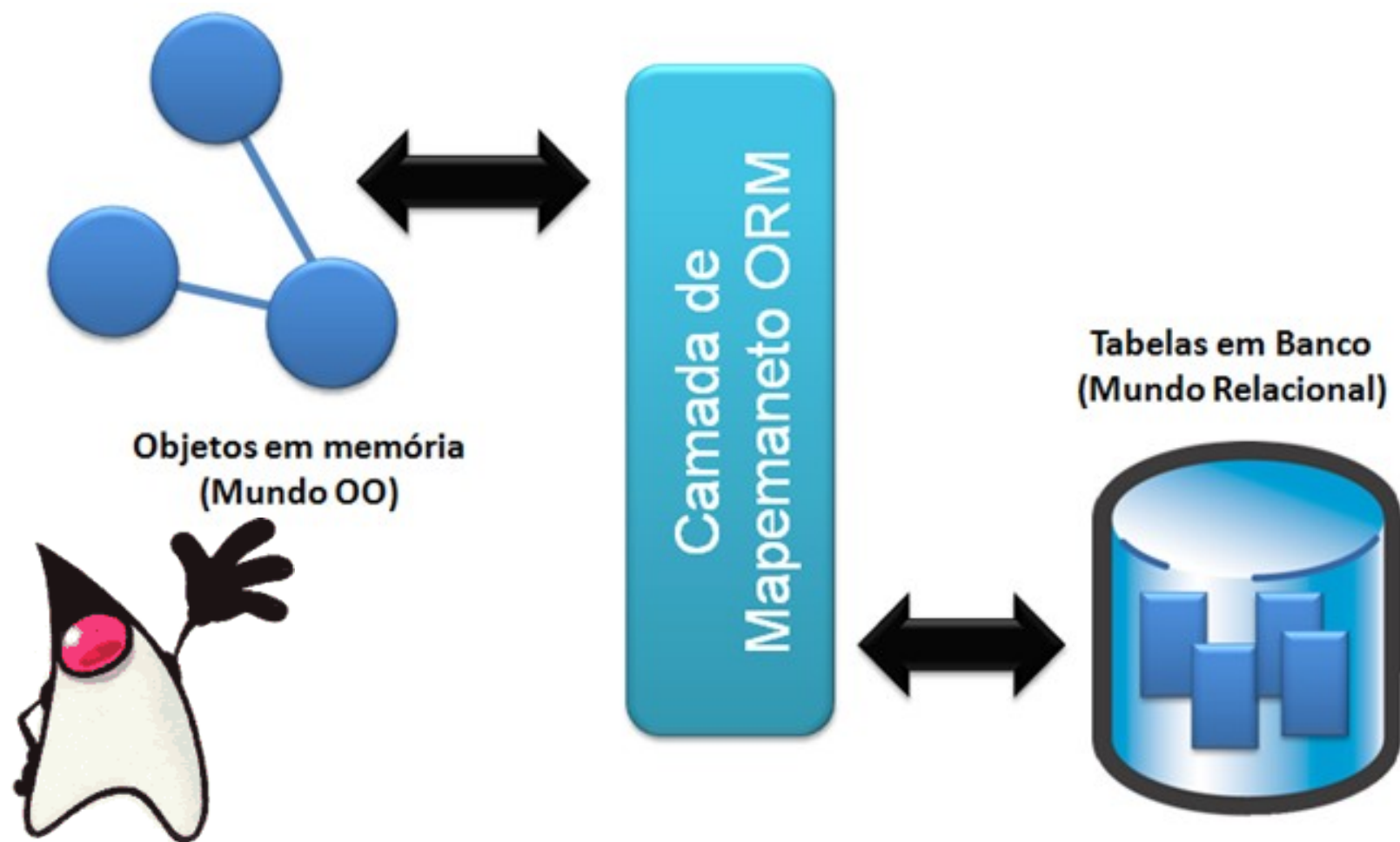
# Java Persistence API (JPA)

- Persistência na linguagem Java
  - Interface Serializable
  - Java Database Connectivity (JDBC)
  - Mapeamento objeto-relacional (ORM)



# Java Persistence API (JPA)

- Mapeamento objeto-relacional (ORM)





# Java Persistence API (JPA)

- Mapeamento objeto-relacional (ORM)
  - Frameworks:
    - JBoss Hibernate
    - EclipseLink (antigamente, Oracle TopLink)
    - Java Data Objects (JDO)
    - Java Persistence API (JPA)



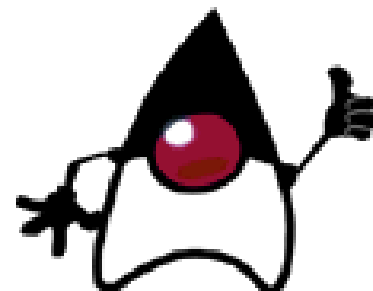
# Java Persistence API (JPA)

- **Java Persistence API (JPA):** padrão de persistência de dados da plataforma Java EE
  - Baseada no modelo ORM
  - Versão atual: 2.0
  - Definida pela JSR 317
  - Disponível para as plataformas Java SE e Java EE



# Java Persistence API (JPA)

- Java Persistence API (JPA)
  - Implementações (*persistence providers*):



eclipse)link  HIBERNATE



# Java Persistence API (JPA)

- Java Persistence API (JPA)
  - Uso em projetos Maven:

pom.xml

```
<project>

  <!-- Outras declarações -->

  <dependencies>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-entitymanager</artifactId>
      <version>4.0.1.Final</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <!-- Outras declarações -->

</project>
```



# Mapeamento de entidades

- Anotações @Entity e @Id

Contato.java

```
import javax.persistence.*;
```

```
@Entity
```

```
public class Contato implements Serializable {
```

```
    @Id
```

```
    private int id;
```

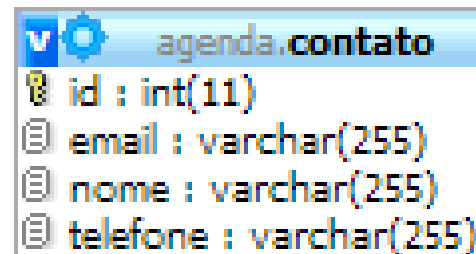
```
    private String nome;
```

```
    private String email;
```

```
    private String telefone;
```

```
    // Getters e setters
```

```
}
```



# Mapeamento de entidades

- Anotações @Entity e @Id

Categoria.java

```
import javax.persistence.*;
```

```
@Entity
```

```
public class Categoria implements Serializable {
```

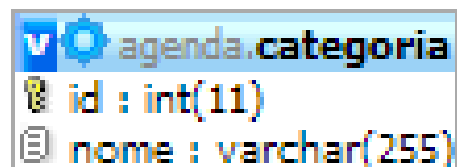
```
    @Id
```

```
    private int id;
```

```
    private String nome;
```

```
    // Getters e setters
```

```
}
```



```
agenda.categoria  
id : int(11)  
nome : varchar(255)
```

# Mapeamento de entidades

- Entity Manager
  - Peça central da API responsável por gerenciar entidades, lendo e gravando no banco de dados
  - CRUD (Create, Read, Update and Delete)
  - JPQL (Java Persistence Query Language)
  - É uma interface definida pela especificação
  - É associado a uma *persistence unit*
    - Conjunto definido pelo banco de dados, configurações de acesso, entidades, etc.
    - Permite a gerência de várias bases de dados

# Mapeamento de entidades

- Entity Manager

JPATest.java

```
import static org.junit.Assert.assertNotNull;

import javax.persistence.*;

import org.junit.Test;

public class JPATest {

    private static EntityManagerFactory emf;

    @Test
    public void atualizarBanco() throws Exception {
        emf = Persistence.createEntityManagerFactory("agenda");
        assertNotNull(emf);
    }

}
```



# Mapeamento de entidades

- Configuração da persistência (Java SE)

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="agenda" transaction-type="RESOURCE_LOCAL">
    <class>br.com.vinyanalista.agenda.modelo.Contato</class>
    <class>br.com.vinyanalista.agenda.modelo.Categoria</class>
    <properties>
      <!-- Configuração JPA -->
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/agenda" />
      <property name="javax.persistence.jdbc.user" value="root" />
      <property name="javax.persistence.jdbc.password" value="" />

      <!-- Configuração do Hibernate -->
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="create-drop" />
    </properties>
  </persistence-unit>
</persistence>
```

# Mapeamento de entidades

- Configuração da persistência (Java SE)

## Terminal

```
$ cd ~/codigo  
$ exemplo02/testar.sh
```

## Navegador

<http://localhost/phpmyadmin>

# Mapeamento de entidades

- Anotação @GeneratedValue

## Contato.java

```
import javax.persistence.*;

@Entity
public class Contato {

    @Id
    @GeneratedValue
    private int id;

    private String nome;

    private String email;

    private String telefone;

    // Getters e setters

}
```

## Categoria.java

```
import javax.persistence.*;

@Entity
public class Categoria {

    @Id
    @GeneratedValue
    private int id;

    private String nome;

    // Getters e setters

}
```

agenda.contato

- id : int(11)
- email : varchar(255)
- nome : varchar(255)
- telefone : varchar(255)

agenda.categoria

- id : int(11)
- nome : varchar(255)

# Mapeamento de entidades

- Anotação @Table

## Contato.java

```
import javax.persistence.*;

@Entity
@Table(name = "age_contato")
public class Contato {

    @Id
    @GeneratedValue
    private int id;

    private String nome;

    private String email;

    private String telefone;

    // Getters e setters

}
```

## Categoria.java

```
import javax.persistence.*;

@Entity
@Table(name = "age_categoria")
public class Categoria {

    @Id
    @GeneratedValue
    private int id;

    private String nome;

    // Getters e setters

}
```

agenda.age\_contato

- id : int(11)
- email : varchar(255)
- nome : varchar(25)
- telefone : varchar(11)

agenda.age\_categoria

- id : int(11)
- nome : varchar(20)

# Mapeamento de entidades

- Anotação @Column

## Contato.java

```
import javax.persistence.*;

@Entity
@Table(name = "age_contato")
public class Contato {

    @Id
    @GeneratedValue
    @Column(name = "con_id")
    private int id;

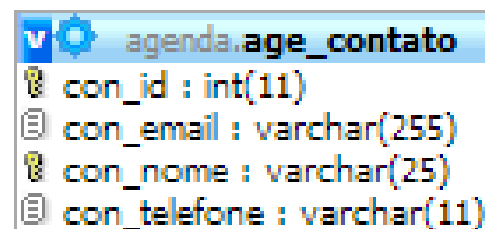
    @Column(name = "con_nome", unique = true, nullable = false)
    private String nome;

    @Column(name = "con_email")
    private String email;

    @Column(name = "con_telefone")
    private String telefone;

    // Getters e setters

}
```



The screenshot shows a database table named 'agenda.age\_contato' with the following columns:

|    | con_id  | con_email    | con_nome    | con_telefone |
|----|---------|--------------|-------------|--------------|
| PK | int(11) |              |             |              |
|    |         | varchar(255) |             |              |
| PK |         |              | varchar(25) |              |
|    |         |              |             | varchar(11)  |

# Mapeamento de entidades

- Anotação @Column

Categoria.java

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name = "age_categoria")
```

```
public class Categoria {
```

```
    @Id
```

```
    @GeneratedValue
```

```
    @Column(name = "cat_id")
```

```
    private int id;
```

```
    @Column(name = "cat_nome", unique = true, nullable = false)
```

```
    private String nome;
```

```
    // Getters e setters
```

```
}
```

```
v agenda.age_categoria  
? cat_id : int(11)  
? cat_nome : varchar(20)
```

# Mapeamento de relacionamentos

- Relacionamento um para um (@OneToOne)

## Contato.java

```
@Entity
public class Contato {

    @Id
    private int id;

    private String nome;
    private String email;
    private String telefone;

    @OneToOne
    @JoinColumn(name = "endereco_id",
        cascade = { CascadeType.REMOVE })
    private Endereco endereco;
}
```

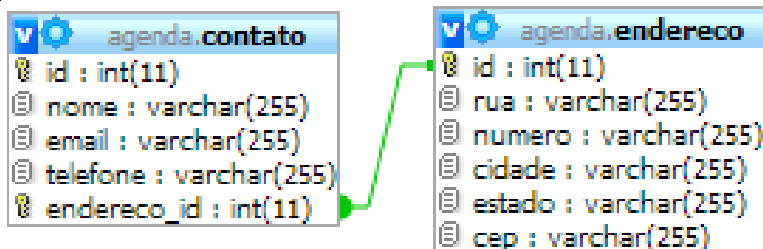
## Endereco.java

```
@Entity
public class Endereco {

    @Id
    @GeneratedValue
    private int id;

    private String rua;
    private String numero;
    private String cidade;
    private String estado;
    private String cep;

    @OneToOne(mappedBy = "endereco")
    private Contato contato;
}
```



# Mapeamento de relacionamentos

- Relacionamento um para muitos (@OneToMany)

## Contato.java

```
@Entity
public class Contato {

    @Id
    private int id;

    private String nome;
    private String email;

    @OneToMany(fetch = FetchType.EAGER, mappedBy = "contato",
        cascade = { CascadeType.REMOVE })
    private List<Telefone> telefones;

}
```

## Telefone.java

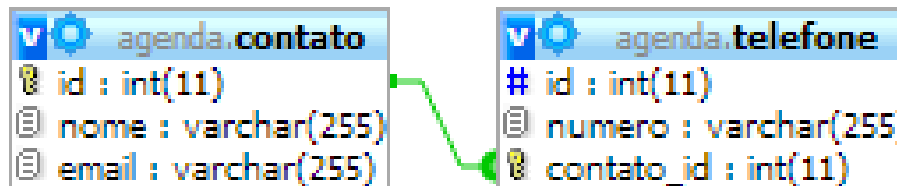
```
@Entity
public class Telefone {

    @Id
    @GeneratedValue
    private int id;

    private String numero;

    @ManyToOne
    private Contato contato;

}
```





# Mapeamento de relacionamentos

- Relacionamento muitos para muitos (@ManyToMany)

## Contato.java

```
@Entity
@Table(name = "age_contato")
public class Contato {

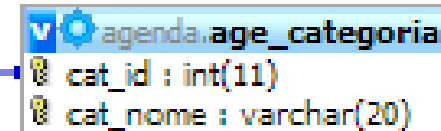
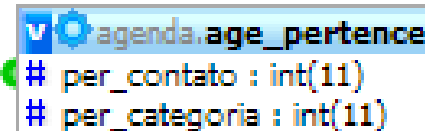
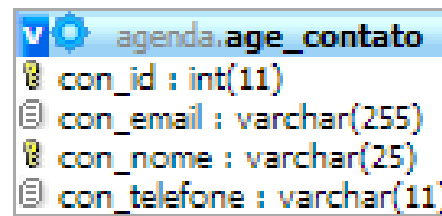
    @Id
    @GeneratedValue
    @Column(name = "con_id")
    private int id;

    @Column(name = "con_nome", unique = true, nullable = false)
    private String nome;

    @Column(name = "con_email")
    private String email;

    @Column(name = "con_telefone")
    private String telefone;

    @ManyToMany
    @JoinTable(name = "age_pertence",
        joinColumns = @JoinColumn(name = "per_contato"),
        inverseJoinColumns = @JoinColumn(name = "per_categoria"))
    private List<Categoria> categorias;
}
```



# Mapeamento de relacionamentos

- Relacionamento muitos para muitos (@ManyToMany)

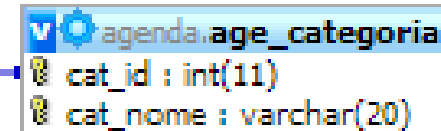
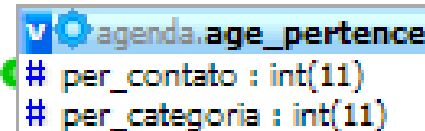
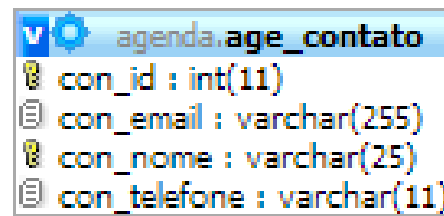
## Categoria.java

```
@Entity
@Table(name = "age_categoria")
public class Categoria {

    @Id
    @GeneratedValue
    @Column(name = "cat_id")
    private int id;

    @Column(name = "cat_nome", unique = true, nullable = false)
    private String nome;

    @ManyToMany(mappedBy = "categorias")
    private List<Contato> contatos;
}
```



## Terminal

```
$ exemplo03/testar.sh
```

## Navegador

```
http://localhost/phpmyadmin
```

# Gerenciando entidades

- Através do Entity Manager

## AbstractPersistenceTest.java

```
public class AbstractPersistenceTest {

    protected static EntityManagerFactory entityManagerFactory;
    protected static EntityManager entityManager;
    protected static EntityTransaction transaction;

    @Before
    public void inicializarEntityManager() throws Exception {
        entityManagerFactory = Persistence.createEntityManagerFactory("agenda");
        entityManager = entityManagerFactory.createEntityManager();
        transaction = entityManager.getTransaction();

        assertNotNull(entityManager); //Teste
        assertNotNull(transaction); //Teste
    }

    @After
    public void initTransaction() {
        if (entityManager != null) entityManager.close();
        if (entityManagerFactory != null) entityManagerFactory.close();
    }

}
```

# Gerenciando entidades

- Através do Entity Manager

EntityManagerTest.java

```
public class EntityManagerTest extends AbstractPersistenceTest {

    @Test
    public void inserirObterContato() throws Exception {
        // Insere contato
        Contato contato = new Contato();
        contato.setNome("Antônio Vinícius");
        contato.setTelefone("1212341234");
        contato.setEmail("vinyanalista@gmail.com");

        transaction.begin();
        entityManager.persist(contato);
        transaction.commit();

        // Obtém contato
        contato = entityManager.find(Contato.class, 1);

        assertNotNull(contato); //Teste
    } //Continua...
```

# Gerenciando entidades

- Através do Entity Manager

EntityManagerTest.java

```
@Test
public void alterarContato() throws Exception {
    // Insere contato

    // Obtém contato
    contato = entityManager.find(Contato.class, 1);

    // Altera contato
    contato.setNome("Vinícius Antônio");

    transaction.begin();
    entityManager.merge(contato);
    transaction.commit();

    // Obtém contato

    assertEquals(contato.getNome(), "Vinícius Antônio"); //Teste
} //Continua...
```

# Gerenciando entidades

- Através do Entity Manager

EntityManagerTest.java

```
@Test
public void removerContato() throws Exception {
    // Insere contato

    // Obtém contato
    contato = entityManager.find(Contato.class, 1);

    // Remove contato
    transaction.begin();
    entityManager.remove(contato);
    transaction.commit();

    // Tenta obter o contato recém removido
    contato = entityManager.find(Contato.class, 1);
    assertNull(contato); //Teste
} //Continua...
```

# Gerenciando entidades

- Através do Entity Manager

## EntityManagerTest.java

```
@Test
public void atribuirCategoria() throws Exception {
    // Insere contato
    // Insere categoria

    // Obtém contato
    contato = entityManager.find(Contato.class, 1);

    // Obtém categoria
    categoria = entityManager.find(Categoria.class, 1);

    // Atribui categoria a contato
    contato.getCategorias().add(categoria);

    transaction.begin();
    entityManager.merge(contato);
    entityManager.merge(categoria);
    transaction.commit();

    contato = entityManager.find(Contato.class, 1);
    assertTrue(!contato.getCategorias().isEmpty()); //Teste
} //Continua...
```

# Gerenciando entidades

- Através do Entity Manager

## EntityManagerTest.java

```
@Test
public void obterContatosDeUmaCategoria() throws Exception {
    // Insere contato 1
    // Insere contato 2
    // Insere categoria

    // Obtém categoria
    categoria = entityManager.find(Categoria.class, 1);

    // Obtém contato 1 e atribui-lhe a categoria
    contato1 = entityManager.find(Contato.class, 1);
    contato1.getCategorias().add(categoria);

    // Obtém contato 2 e atribui-lhe a categoria
    contato2 = entityManager.find(Contato.class, 2);
    contato2.getCategorias().add(categoria);

    // Atribui categoria aos contatos
    transaction.begin();
    entityManager.merge(contato1);
    entityManager.merge(contato2);
    transaction.commit();

    categoria = entityManager.find(Categoria.class, 1);
    assertTrue(!categoria.getContatos().isEmpty()); //Teste
} //Continua...
```



# Gerenciando entidades

- Através do Entity Manager

## EntityManagerTest.java

```
@Test
public void removerContatoDeUmaCategoria() throws Exception {
    // Insere contato 1
    // Insere contato 2
    // Insere categoria
    // Obtém categoria
    // Obtém contato 1 e atribui-lhe a categoria
    // Obtém contato 2 e atribui-lhe a categoria
    // Atribui categoria aos contatos

    // Obtém contato 1 e remove-lhe a categoria
    contato1 = entityManager.find(Contato.class, 1);
    contato1.getCategorias().remove(categoria);

    transaction.begin();
    entityManager.merge(contato1);
    transaction.commit();

    categoria = entityManager.find(Categoria.class, 1);
    assertTrue(!categoria.getContatos().contains(contato1)); // Teste
}

} //Fim da classe EntityManagerTest
```

# Gerenciando entidades

- Através do Entity Manager

Terminal

```
$ exemplo04/testar.sh
```

- Através da JPQL
  - Java Persistence Query Language
  - Oferece mais possibilidades de operações
  - Baseada no SQL convencional
  - Orientada a objetos: entidades e atributos ao invés de tabelas e colunas
  - Independente do banco de dados utilizado

# Gerenciando entidades

- Através da JPQL

JPQLTest.java

```
public class JPQLTest extends AbstractPersistenceTest {

    @Test
    public void obterTodosOsContatos() throws Exception {
        // Insere 2 contatos
        Contato contato1 = new Contato(); contato1.setNome("Antônio Vinícius");
        contato1.setTelefone("1212341234");

        Contato contato2 = new Contato(); contato2.setNome("João");
        contato2.setTelefone("5656785678");

        transaction.begin(); entityManager.persist(contato1);
        entityManager.persist(contato2); transaction.commit();

        // Obtém os contatos inseridos
        String jpql = "select c from Contato c";
        Query query = entityManager.createQuery(jpql);
        List contatos = query.getResultList(); //Advertência

        assertEquals(2, contatos.size()); // Teste
    } //Continua...
```

# Gerenciando entidades

- Através da JPQL

JPQLTest.java

```
@Test
public void typedQuery() throws Exception {
    // Insere 2 contatos

    // Obtém os contatos inseridos
    String jpql = "select c from Contato c";
    TypedQuery<Contato> query = entityManager.createQuery(jpql,
        Contato.class);
    List<Contato> contatos = query.getResultList();

    assertEquals(2, contatos.size()); // Teste
} //Continua...
```

# Gerenciando entidades

- Através da JPQL
  - Tipos de query:
    - **Dynamic queries:** forma mais simples de query, uma String fornecida durante a execução (exemplos anteriores);
    - **Named queries:** possuem um nome, são estáticas e imutáveis;
    - **Native queries:** permitem a execução de SQL, ao invés de JPQL; e
    - **Criteria API:** conceito novo da JPA 2.0.

# Gerenciando entidades

- Através da JPQL

## Contato.java

```
@Entity
@Table(name = "age_contato")
@NamedQuery(name = Contato.LISTAR_TODOS, query = "SELECT c FROM Contato c")
public class Contato implements Serializable {

    public static final String LISTAR_TODOS = "Contato.listarTodos";

}
```

## JPQLTest.java

```
@Test
public void namedQuery() throws Exception {
    // Insere 2 contatos

    // Obtém os contatos inseridos
    TypedQuery<Contato> query = entityManager.createNamedQuery(
        Contato.LISTAR_TODOS, Contato.class);
    List<Contato> contatos = query.getResultList();

    assertEquals(2, contatos.size()); // Teste
}

} //Fim da classe JPQLTest
```

# Gerenciando entidades

- Através da JPQL

Terminal

```
$ exemplo05/testar.sh
```

# Roteiro

- ✓ **Introdução à Plataforma Java EE**
- ✓ **A aplicação de exemplo "Agenda de Contatos"**
- ✓ **Apresentação das ferramentas utilizadas**
- ✓ **Persistência com JPA**
- ➔ **Validação de dados com Bean Validation**
  - Lógica de negócios com EJB
  - Apresentação com JSF
  - Implementação de um CRUD com JSF



# Validação de dados

- Qualquer aplicação deve se preocupar com a integridade dos dados
- Tradicionalmente feita através de rotinas de validação
  - Difícil de desenvolver e manter
  - Aumenta o código e a complexidade do sistema



# Bean Validation API

- **Bean Validation API:** padrão de validação de dados da plataforma JEE
  - Versão atual: 1.0
  - Definida pela JSR 303
  - Disponível para as plataformas JSE e JEE
  - Integrada com a especificação JPA



# Bean Validation API

- Bean Validation API
  - Implementações:



# Bean Validation API

- Bean Validation API
  - Uso em projetos Maven:

pom.xml

```
<project>

  <!-- Outras declarações -->

  <dependencies>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-validator</artifactId>
      <version>4.2.0.Final</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <!-- Outras declarações -->

</project>
```



# Bean Validation API

- Anotações

Categoria.java

```
import javax.validation.constraints.*;

import org.hibernate.validator.constraints.NotBlank;

@Entity
public class Categoria implements Serializable {

    private int id;

    @NotNull
    @NotBlank
    @Size(min = 1, max = 20)
    @Pattern(regexp = "[A-Za-z ]*")
    private String nome;

    private List<Contato> contatos = new ArrayList<Contato>();

}
```

# Bean Validation API

- Anotações

## Contato.java

```
import javax.validation.constraints.*;

import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotBlank;

@Entity
public class Contato implements Serializable {

    private int id;

    @NotNull
    @NotBlank
    @Size(min = 1, max = 25)
    @Pattern(regexp = "[A-Za-z ]*")
    private String nome;

    @Email
    private String email;

    @Size(max = 11)
    @Digits(fraction = 0, integer = 12)
    private String telefone;

    private List<Categoria> categorias = new ArrayList<Categoria>();
}
```

# Bean Validation API

- Anotações

## Outras anotações disponíveis

```
@AssertFalse  
boolean suportado;
```

```
@AssertTrue  
boolean ativo;
```

```
@Min(5)  
@Max(10)  
int quantidade;
```

```
@DecimalMin("5.00")  
@DecimalMax("30.00")  
BigDecimal desconto;
```

```
@Past  
Date aniversario;
```

```
@Future  
Date dataDoEvento;
```

```
@Null  
String stringQueNaoDeveSerUtilizada;
```

**Referência:** <http://docs.oracle.com/javaee/6/tutorial/doc/gircz.html>

# Bean Validation API

- Personalizando as mensagens de erro

Categoria.java

```
import javax.validation.constraints.*;

import org.hibernate.validator.constraints.NotBlank;

@Entity
public class Categoria implements Serializable {

    private int id;

    @NotNull(message = "Preenchimento obrigatório!")
    @NotBlank(message = "Preenchimento obrigatório!")
    @Size(min = 1, max = 20, message = "Não deve ultrapassar 20 caracteres!")
    @Pattern(regexp = "[A-Za-z ]*", message = "Deve conter apenas letras
    maiúsculas e minúsculas, sem acentos ou cedilha, e espaços.")
    private String nome;

    private List<Contato> contatos = new ArrayList<Contato>();

}
```



# Bean Validation API

- Testando a Bean Validation API

ValidationTest.java

```
public class ValidationTest extends AbstractPersistenceTest {  
  
    protected static ValidatorFactory validatorFactory;  
    protected static Validator validator;  
  
    @Before  
    public void inicializarValidator() throws Exception {  
        validatorFactory = Validation.buildDefaultValidatorFactory();  
        validator = validatorFactory.getValidator();  
    } //Continua...
```

# Bean Validation API

- Testando a Bean Validation API

## ValidationTest.java

```
@Test
public void validarContatoSemTratarExcecao() throws Exception {
    // Tenta inserir contato
    Contato contato = new Contato();
    contato.setNome("João 18");
    contato.setTelefone("12345467");
    contato.setEmail("joao_email");

    transaction.begin();
    entityManager.persist(contato);
    transaction.commit();

    // Obtém os contatos inseridos
    String jpql = "select c from Contato c";
    TypedQuery<Contato> query = entityManager.createQuery(jpql,
        Contato.class);
    List<Contato> contatos = query.getResultList();

    assertEquals(1, contatos.size()); // Teste
} //Continua...
```

# Bean Validation API

- Testando a Bean Validation API

## ValidationTest.java

```
@Test
public void validarContatoTratandoExcecao() throws Exception {
    // Tenta inserir contato

    try {
        transaction.begin();
        entityManager.persist(contato);
        transaction.commit();
    } catch (ConstraintViolationException excecaoDeValidacao) {
        transaction.rollback();
        System.out.println("Erros:");
        Set<ConstraintViolation<?>> erros = excecaoDeValidacao
            .getConstraintViolations();
        for (ConstraintViolation<?> erro : erros) {
            System.out.println("-" + erro.getMessage());
        }
    }

    // Obtém os contatos inseridos

    assertEquals(1, contatos.size()); // Teste
}

// Fim da classe ValidationTest
```

# Bean Validation API

- Testando a Bean Validation API

Terminal

```
$ exemplo06/testar.sh
```

# Roteiro

- ✓ **Introdução à Plataforma Java EE**
- ✓ **A aplicação de exemplo "Agenda de Contatos"**
- ✓ **Apresentação das ferramentas utilizadas**
- ✓ **Persistência com JPA**
- ✓ **Validação de dados com Bean Validation**
- ➔ **Lógica de negócios com EJB**
  - Apresentação com JSF
  - Implementação de um CRUD com JSF

# Enterprise Java Beans (EJB)

- Lógica de negócios
  - Separação entre as camadas de persistência e apresentação, concentra o processamento da aplicação
    - Integra bancos de dados a diversos sistemas/clientes
  - Modela as ações (verbos) da aplicação
    - Listar, inserir, alterar, remover, processar, registrar, autenticar, permitir, negar, imprimir, reservar, comprar, enviar, compartilhar, etc.



# Enterprise Java Beans (EJB)

- **Enterprise Java Beans (EJB):**  
componentes executados no servidor que encapsulam a lógica de negócios da aplicação, gerenciando também transações e segurança.
  - Versão atual: 3.1
  - Definida pela JSR 318
  - Disponível para as plataformas JSE (cliente, remoto) e JEE (local, servidor)
  - Implementação de referência: GlassFish



# Enterprise Java Beans (EJB)

- Enterprise Java Beans (EJB):
  - Invocação remota de métodos
  - Injeção de dependências
  - Gerenciamento de ciclo de vida e de estado
  - *Pooling*
  - Envio de mensagens assíncronas (e-mails, por exemplo)
  - Agendamento de tarefas
  - Persistência
  - Transações, segurança e concorrência





# Enterprise Java Beans (EJB)

- Enterprise Java Beans (EJB):
  - Integrados transparentemente a outras tecnologias das plataformas JSE e JEE:
    - Java Database Connectivity (JDBC)
    - Java Persistence API (JPA)
    - Java Transaction API (JTA)
    - Java Messaging Service (JMS)
    - JavaMail
    - Java Authentication and Authorization Service (JAAS)
    - Java Naming and Directory Interface (JNDI)
    - Remote Method Invocation (RMI)



# Enterprise Java Beans (EJB)

- Enterprise Java Beans (EJB)
  - Uso em projetos Maven:

pom.xml

```
<project>

  <!-- Outras declarações -->

  <dependencies>
    <dependency>
      <groupId>org.glassfish</groupId>
      <artifactId>javax.ejb</artifactId>
      <version>3.1</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <!-- Outras declarações -->

</project>
```



# Tipos de EJBs

- **Session beans (SBs):** encapsulam o mais alto nível da lógica de negócios, sendo por isso os mais importantes (e mais comuns) EJBs
  - Implementam a lógica de negócios do sistema propriamente dita: transações, segurança, concorrência, etc.
- **Message-driven beans (MDBs):** consumidores de mensagens assíncronos, são executados quando uma mensagem é recebida pelo servidor
  - Implementam comunicação com outros sistemas via envio e recebimento de mensagens pelo JMS

# Tipos de EJBs

- **Session beans (SBs):**
  - **Stateless session beans:** não armazenam estado associado a clientes ou seções
    - Uma instância "sobrevive" a apenas um requisição
    - Destinados à execução de tarefas que são concluídas com uma única requisição
    - Exemplo: Data Access Objects (DAOs)

# Tipos de EJBs

- **Session beans (SBs):**
  - **Stateful session beans:** armazenam estado associado a determinados cliente e sessão
    - Uma instância destinada a uma sessão só pode ser utilizada enquanto esta estiver ativa
    - Destinados à execução de tarefas que devem ser realizadas por etapas
    - Devem ser utilizados com cautela, pois demandam alto consumo de recursos do servidor
    - Exemplo: carrinho de compras (vendas)

# Tipos de EJBs

- **Session beans (SBs):**
  - **Singleton session beans:** armazenam estado associado à aplicação
    - São instanciados apenas uma vez pelo servidor de aplicação, e permanecem disponíveis a todos os clientes enquanto a aplicação estiver em execução
    - O servidor de aplicação garante o acesso concorrente aos *singletons* da aplicação
    - Exemplo: variáveis globais, de configuração da aplicação

# Tipos de EJBs

- Exemplos:

ContatoDAO.java

```
import javax.ejb.Stateless;

@Stateless
public class ContatoDAO implements Serializable {

    private static final long serialVersionUID = 1L;

    @PersistenceContext(unitName = "agenda")
    private EntityManager entityManager;

    public Contato inserir(Contato contato) {
        entityManager.persist(contato);
        return contato;
    }

    public Contato buscar(Integer id) {
        return entityManager.find(Contato.class, id);
    } //Continua...
```

# Tipos de EJBs

- Exemplos:

ContatoDAO.java

```
public void atualizar(Contato contato) {  
    entityManager.merge(contato);  
}  
  
public List<Contato> listarTodos() {  
    return entityManager.createNamedQuery(Contato.LISTAR_TODOS,  
        Contato.class).getResultList();  
}  
  
public void remover(Contato contato) {  
    entityManager  
        .remove(entityManager.find(Contato.class,  
            contato.getId()));  
}  
  
} //Fim da classe ContatoDAO
```



# Acesso remoto

- Anotação @Remote

## ContatoDAORemote.java

```
import javax.ejb.Remote;

@Remote
public interface ContatoDAORemote {

    public Contato inserir(Contato contato);
    public Contato buscar(Integer id);
    public void atualizar(Contato contato);
    public List<Contato> listarTodos();
    public void remover(Contato contato);

}
```

## ContatoDAO.java

```
import javax.ejb.Stateless;

@Stateless
public class ContatoDAO implements ContatoDAORemote, Serializable {

}
```

# Acesso remoto

- **Java Naming and Directory Interface (JNDI):** é uma API presente nas plataformas JSE e JEE que permite o acesso a serviços de diretórios
  - Objetos e recursos são associados a nomes pelos quais podem ser localizados e obtidos remotamente
  - Independente da implementação
    - Lightweight Directory Access Protocol (LDAP)
    - Domain Name System (DNS)
    - Common Object Request Broker Architecture (CORBA)



# Acesso remoto

- Implementação do cliente

Cliente.java

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class Cliente {
    public static void main(String[] args) throws Exception {
        // Cria uma nova instância do contato
        Contato contato = new Contato();
        contato.setNome("Jose");
        contato.setTelefone("1212345678");

        // Obtém o DAO
        ContatoDAORemote dao = obterDaoRemoto();

        // Insere o contato na base de dados
        contato = dao.inserir(contato);
    } //Continua...
```

# Acesso remoto

- Implementação do cliente

Cliente.java

```
private static ContatoDAORemote obterDaoRemoto() throws NamingException {
    Hashtable<String, String> jndiProperties = new Hashtable<String, String>();
    jndiProperties.put(Context.URL_PKG_PREFIXES,
        "org.jboss.ejb.client.naming");
    Context context = new InitialContext(jndiProperties);

    String appName = "";
    String moduleName = "exemplo07";
    String distinctName = "";
    String beanName = ContatoDAO.class.getSimpleName();
    String viewClassName = ContatoDAORemote.class.getName();

    return (ContatoDAORemote) context.lookup("ejb:" + appName + "/"
        + moduleName + "/" + distinctName + "/" + beanName + "!"
        + viewClassName);

    // ejb:exemplo07/ContatoDAO!br.com.vinyanalista.agenda.dao.ContatoDAORemote
}
} //Fim da classe Cliente
```

# Acesso remoto

- Configuração da persistência (no servidor)

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="agenda" transaction-type="JTA">
    <jta-data-source>java:/agenda</jta-data-source>
    <properties>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="create-drop" />
    </properties>
  </persistence-unit>
</persistence>
```

## Maiores informações:

<https://docs.jboss.org/author/display/AS71/EJB+invocations+from+a+remote+client+using+JNDI>

<http://www.vinyanalista.com.br/blog/2012/07/19/configurando-um-datasource-do-mysql-no-jboss-as/>

# Acesso remoto

- Implementação do cliente

## Terminal

```
$ exemplo07/servidor/implantar.sh  
$ exemplo07/cliente/executar.sh
```

## Navegador

<http://localhost/phpmyadmin>

# Roteiro

- ✓ **Introdução à Plataforma Java EE**
- ✓ **A aplicação de exemplo "Agenda de Contatos"**
- ✓ **Apresentação das ferramentas utilizadas**
- ✓ **Persistência com JPA**
- ✓ **Validação de dados com Bean Validation**
- ✓ **Lógica de negócios com EJB**
- ➔ **Apresentação com JSF**
  - Implementação de um CRUD com JSF

# Interfaces *web* na plataforma Java EE

- A informação processada deve ser retornada ao usuário
  - Aplicações *desktop*
  - Aplicações *web* em um navegador
  - Aplicações para dispositivos móveis
- **Aplicações *web***
  - World Wide Web (WWW)
  - Hypertext Markup Language (HTML)
  - Hypertext Transfer Protocol (HTTP)
  - Páginas estáticas ou dinâmicas





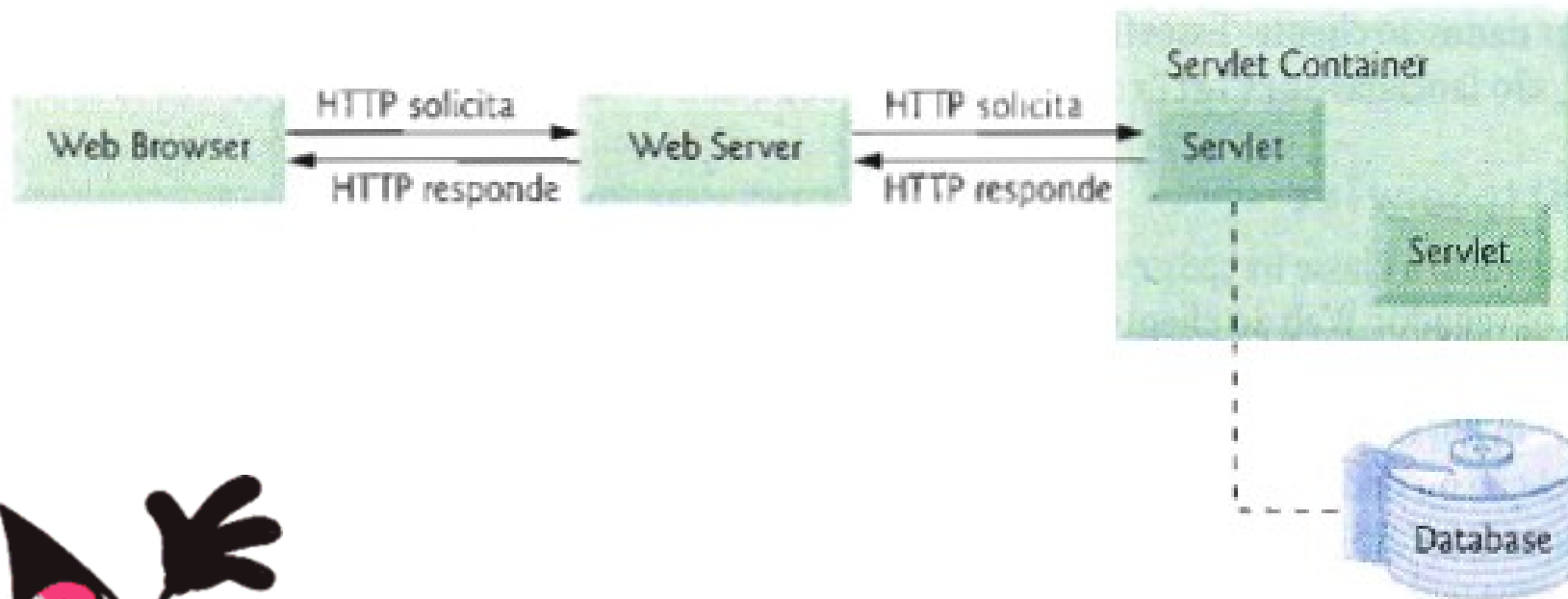
# Interfaces *web* na plataforma Java EE

- **Servlet API (1996):** estende os recursos de servidores cujas aplicações atendem por um modelo de requisição-resposta
  - Normalmente utilizado para abstrair o protocolo HTTP em servidores *web*
  - Alternativa aos scripts CGI (Common Gateway Interface)
  - Traz recursos da linguagem Java
    - Interação com banco de dados
    - Acesso remoto
    - Independência de plataforma



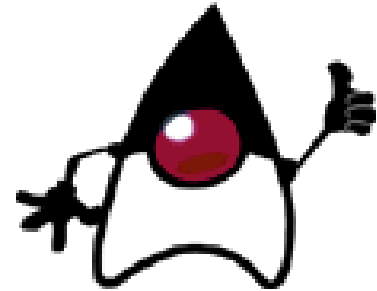
# Interfaces *web* na plataforma Java EE

- Servlet API



# Interfaces *web* na plataforma Java EE

- Servlet API
  - Versão atual: 3.0
  - Definida pela JSR 315
  - Implementação de referência: GlassFish



**GlassFish**

# Interfaces *web* na plataforma Java EE

- Servlet API
  - Uso em projetos Maven:

pom.xml

```
<project>

  <!-- Outras declarações -->

  <dependencies>
    <dependency>
      <groupId>org.glassfish</groupId>
      <artifactId>javax.servlet</artifactId>
      <version>3.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <!-- Outras declarações -->

</project>
```



# Interfaces *web* na plataforma Java EE

- Servlet API

## ListaContatosServlet.java

```
package br.com.vinyanalista.agenda.web.servlets;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ListaContatosServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest requisicao,
        HttpServletResponse resposta) throws ServletException,
        IOException {
        resposta.setContentType("text/html");
        PrintWriter saida = resposta.getWriter();

        saida.println("<html>");
        saida.println("<head>");
        saida.println("<title>Agenda de Contatos</title>");
        saida.println("</head>");
        saida.println("<body>"); //Continua...
```

# Interfaces *web* na plataforma Java EE

- Servlet API

## ListaContatosServlet.java

```
saida.println("<h1>Agenda de Contatos</h1>");
saida.println("<hr />");

List<Contato> contatos = new ArrayList<Contato>();

Contato contato1 = new Contato();
contato1.setNome("Antonio Vinicius");
contato1.setTelefone("1212341234");
contato1.setEmail("vinyanalista@gmail.com");
contatos.add(contato1);

Contato contato2 = new Contato();
contato2.setNome("Joao");
contato2.setTelefone("5656785678");
contatos.add(contato2);

Contato contato3 = new Contato();
contato3.setNome("Jose");
contato3.setTelefone("1234567890");
contato3.setEmail("jose@exemplo.com");
contatos.add(contato3); //Continua...
```

# Interfaces *web* na plataforma Java EE

- Servlet API

## ListaContatosServlet.java

```
for (Contato contato : contatos) {
    saida.println("<tr>");
    saida.println("<td>" + contato.getNome() + "</td>");
    saida.println("<td>" + contato.getTelefone() + "</td>");
    saida.println("<td>" + contato.getEmail() + "</td>");
    saida.println("</tr>");
}

saida.println("</table>");
saida.println("<hr />");
saida.println("<i>Desenvolvimento de Aplicações Corporativas "
    + "com a Plataforma Java EE 6</i>");
saida.println("</body>");
saida.println("</html>");

saida.close();
}
//Fim da classe ListaContatosServlet
```

# Interfaces *web* na plataforma Java EE

- Servlet API

## WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">

  <display-name>exemplo08</display-name>

  <servlet>
    <servlet-name>lista-contatos</servlet-name>
    <servlet-class>br.com.vinyanalista.agenda.web.servlets.ListaContatosServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>lista-contatos</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

</web-app>
```



# Interfaces *web* na plataforma Java EE

- Servlet API

Terminal

```
$ exemplo08/implantar.sh
```

Navegador

<http://localhost/exemplo08>

## Agenda de Contatos

---

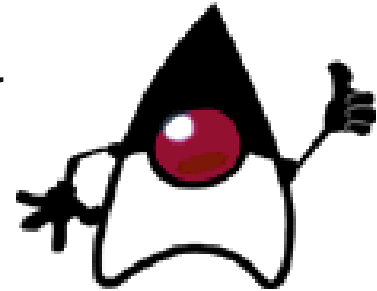
| Nome             | Telefone   | E-mail                 |
|------------------|------------|------------------------|
| Antonio Vinicius | 1212341234 | vinyanalista@gmail.com |
| Joao             | 5656785678 | null                   |
| Jose             | 1234567890 | jose@exemplo.com       |

---

*Desenvolvimento de Aplicações Corporativas com a Plataforma Java EE 6*

# Interfaces *web* na plataforma Java EE

- **JavaServer Pages (1999):** extensão da Servlet API que separa apresentação e conteúdo dinâmico da página
  - Escrito, em maior parte, com HTML
  - Versão atual: 2.2
  - Definida pela JSR 245
  - Implementação de referência:



**GlassFish**

# Interfaces *web* na plataforma Java EE

- JavaServer Pages
  - Uso em projetos Maven:

pom.xml

```
<project>

  <!-- Outras declarações -->

  <dependencies>
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.2</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <!-- Outras declarações -->

</project>
```



# Interfaces *web* na plataforma Java EE

- JavaServer Pages

index.jsp

```
<%@ page import="java.util.ArrayList"%>
<%@ page import="java.util.List"%>
<%@ page import="br.com.vinyanalista.agenda.modelo.Contato"%>
<html>
<head>
<title>Agenda de Contatos</title>
</head>
<body>
  <h1>Agenda de Contatos</h1>
  <hr />
  <%
    List<Contato> contatos = new ArrayList<Contato>();

    Contato contato1 = new Contato();
    contato1.setNome("Antonio Vinicius");
    contato1.setTelefone("1212341234");
    contato1.setEmail("vinyanalista@gmail.com");
    contatos.add(contato1); //Continua...
```

# Interfaces *web* na plataforma Java EE

- JavaServer Pages

index.jsp

```
Contato contato2 = new Contato();
contato2.setNome("Joao");
contato2.setTelefone("5656785678");
contatos.add(contato2);

Contato contato3 = new Contato();
contato3.setNome("Jose");
contato3.setTelefone("1234567890");
contato3.setEmail("jose@exemplo.com");
contatos.add(contato3);
```

%>

```
<table border='1'>
  <tr>
    <th>Nome</th>
    <th>Telefone</th>
    <th>E-mail</th>
  </tr><!-- Continua... -->
```

# Interfaces *web* na plataforma Java EE

- JavaServer Pages

index.jsp

```
<%
    for (Contato contato : contatos) {
%>
<tr>
    <td><%=contato.getNome()%></td>
    <td><%=contato.getTelefone()%></td>
    <td><%=contato.getEmail()%></td>
</tr>
<%
    }
%>

</table>
<hr />
<i>Desenvolvimento de Aplicações Corporativas com a
Plataforma Java
    EE 6</i>
</body>
</html><!-- Fim da página index.jsp -->
```

# Interfaces *web* na plataforma Java EE

- JavaServer Pages

## WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">

  <display-name>exemplo09</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

## Terminal

```
$ exemplo09/implantar.sh
```

## Navegador

```
http://localhost/exemplo09
```

# Interfaces *web* na plataforma Java EE

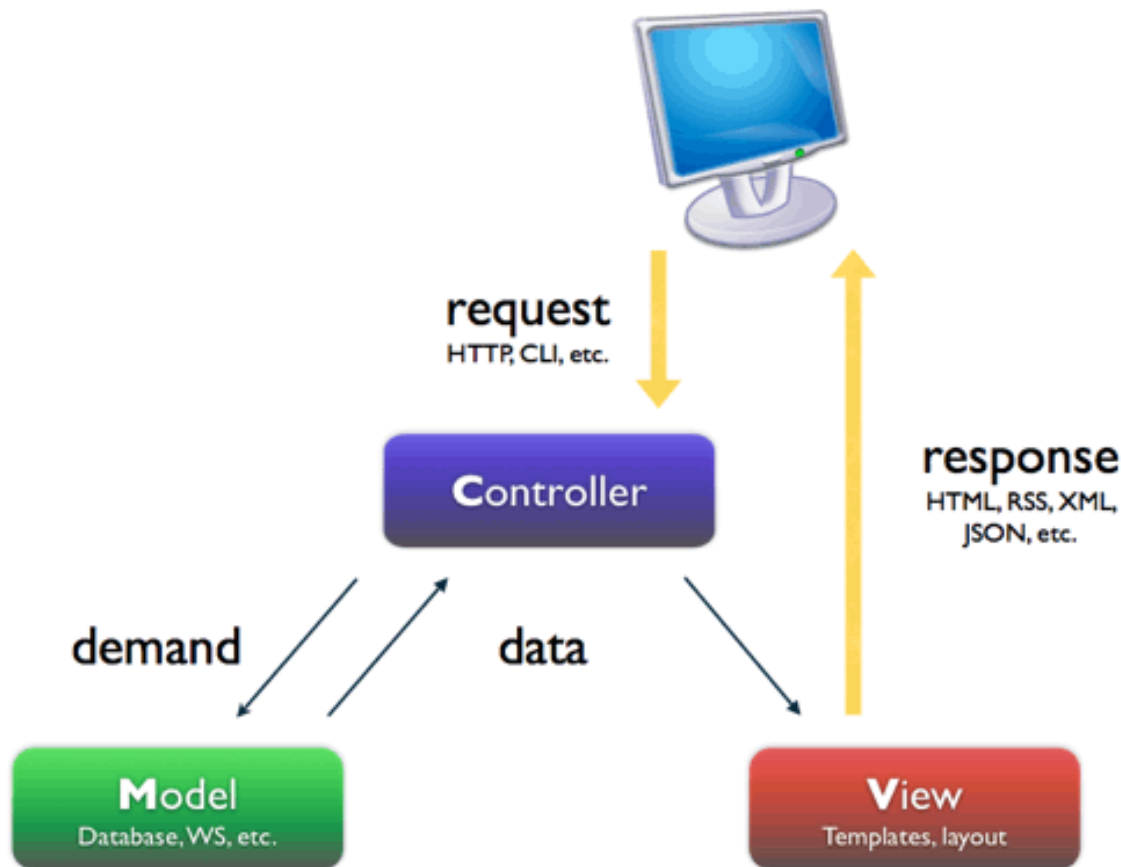
- **JavaServer Faces (2001):** *framework* que permite a criação de interfaces *web* através de componentes reutilizáveis
  - Especificação inspirada em vários *frameworks web* de código aberto
    - Spring, Struts, etc.
  - Baseada no modelo MVC
  - Interface *web* preferida da Java EE 6
  - Versão atual: 2.0
  - Definida pela JSR 314





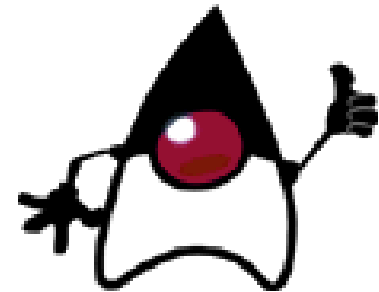
# Interfaces *web* na plataforma Java EE

- JavaServer Faces
  - Modelo Model View Controller (MVC):



# Interfaces *web* na plataforma Java EE

- JavaServer Faces
  - Implementações:



# Interfaces *web* na plataforma Java EE

- JavaServer Faces
  - Uso em projetos Maven:

pom.xml

```
<project>

  <!-- Outras declarações -->

  <dependencies>
    <dependency>
      <groupId>javax.faces</groupId>
      <artifactId>jsf-api</artifactId>
      <version>2.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <!-- Outras declarações -->

</project>
```



# Interfaces *web* na plataforma Java EE

- JavaServer Faces
  - Recursos principais:
    - Permite criar páginas *web*
    - Inserir componentes nas páginas através da utilização de *tags*
    - Associar de componentes em uma página a dados armazenados no servidor
      - Associar eventos dos componentes a códigos executados no servidor
      - Salvar e restaurar o estado da aplicação entre requisições do cliente
      - Reusar, estender e personalizar componentes



# Interfaces *web* na plataforma Java EE

- JavaServer Faces

ListaContatosBean.java

```
package br.com.vinyanalista.agenda.web.mb;

import javax.faces.bean.ManagedBean;

@ManagedBean
public class ListaContatosBean {

    public List<Contato> getContatos() {
        List<Contato> contatos = new ArrayList<Contato>();

        Contato contato1 = new Contato();
        contato1.setNome("Antonio Vinicius");
        contato1.setTelefone("1212341234");
        contato1.setEmail("vinyanalista@gmail.com");
        contatos.add(contato1);

        // Adiciona mais dois contatos...

        return contatos;
    }
}
```

# Interfaces *web* na plataforma Java EE

- JavaServer Faces

index.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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:f="http://java.sun.com/jsf/core">
<h:head>
  <title>Agenda de Contatos</title>
</h:head>
<h:body>
  <h1>Agenda de Contatos</h1>
  <hr /><!-- Continua... -->
```

# Interfaces *web* na plataforma Java EE

- JavaServer Faces

index.xhtml

```
<h:dataTable value="#{listaContatosBean.contatos}" var="contato">
  <h:column>
    <f:facet name="header">Nome</f:facet>
    <h:outputText value="#{contato.nome}" />
  </h:column>
  <h:column>
    <f:facet name="header">Telefone</f:facet>
    <h:outputText value="#{contato.telefone}" />
  </h:column>
  <h:column>
    <f:facet name="header">E-mail</f:facet>
    <h:outputText value="#{contato.email}" />
  </h:column>
</h:dataTable>
<hr />
<i>Desenvolvimento de Aplicações Corporativas com a Plataforma Java
  EE 6</i>
</h:body>
</html><!-- Fim da página index.xhtml -->
```

# Interfaces *web* na plataforma Java EE

- JavaServer Faces

## WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">

  <display-name>exemplo10</display-name>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.xhtml</welcome-file>
  </welcome-file-list>
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>

</web-app>
```



# Interfaces *web* na plataforma Java EE

- JavaServer Faces

Terminal

```
$ exemplo10/implantar.sh
```

Navegador

<http://localhost/exemplo10>

## Agenda de Contatos

---

| Nome             | Telefone   | E-mail                 |
|------------------|------------|------------------------|
| Antonio Vinicius | 1212341234 | vinyanalista@gmail.com |
| Joao             | 5656785678 |                        |
| Jose             | 1234567890 | jose@exemplo.com       |

---

*Desenvolvimento de Aplicações Corporativas com a Plataforma Java EE 6*

# Interfaces *web* na plataforma Java EE

- JavaServer Faces
  - *Frameworks* de componentes:



RichFaces



ICEfaces  
ICE19C62



APACHE MyFaces  
Trinidad

# Interfaces *web* na plataforma Java EE

- RichFaces
  - Surgiu a partir do *framework* Ajax4jsf, criado na época do surgimento do AJAX e da popularização do JSF
  - Provê componentes complementares aos padrões definidos pela especificação JSF
  - Facilita a utilização de AJAX em componentes específicos ou na página como um todo
  - Permite mudar facilmente a aparência da aplicação através de *skins*
  - Validação no cliente, baseada na Bean Validation
  - Permite o desenvolvimento de componentes próprios através do Component Development Kit (CDK)
  - Versão atual: 4.3.0

# Interfaces *web* na plataforma Java EE

- RichFaces
  - Uso em projetos Maven:

pom.xml

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.richfaces</groupId>
      <artifactId>richfaces-bom</artifactId>
      <version>4.3.0</version>
      <scope>import</scope>
      <type>pom</type>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>org.richfaces.ui</groupId>
    <artifactId>richfaces-components-ui</artifactId>
  </dependency>
  <dependency>
    <groupId>org.richfaces.core</groupId>
    <artifactId>richfaces-core-impl</artifactId>
  </dependency>
</dependencies>
```



# Interfaces *web* na plataforma Java EE

- RichFaces

index.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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:f="http://java.sun.com/jsf/core"
      xmlns:a4j="http://richfaces.org/a4j"
      xmlns:rich="http://richfaces.org/rich">
<h:head>
  <title>Agenda de Contatos</title>
  <style>
    label {
      font-weight: bold;
    }
  </style>
</h:head>
<h:body>
  <h1>Agenda de Contatos</h1>
  <hr /><!-- Continua... -->
```

# Interfaces *web* na plataforma Java EE

- RichFaces

index.xhtml

```
<h:panelGrid columns="2">
  <h:form>
    <rich:extendedDataTable value="#{listaContatosBean.contatos}"
      var="contato" selection="#{listaContatosBean.selecao}"
      selectionMode="single">
      <a4j:ajax execute="@form" event="selectionchange"
        listener="#{listaContatosBean.atualizarContatoSelecionado}"
        render=":contato_dados" />
      <rich:column>
        <f:facet name="header">Nome</f:facet>
        <h:outputText value="#{contato.nome}" />
      </rich:column>
      <rich:column>
        <f:facet name="header">Telefone</f:facet>
        <h:outputText value="#{contato.telefone}" />
      </rich:column>
      <rich:column>
        <f:facet name="header">E-mail</f:facet>
        <h:outputText value="#{contato.email}" />
      </rich:column>
    </rich:extendedDataTable>
  </h:form><!-- Continua... -->
```

# Interfaces *web* na plataforma Java EE

- RichFaces

index.xhtml

```
<a4j:outputPanel id="contato_dados">
    <rich:panel header="Dados do contato"
        rendered="#{not empty listaContatosBean.contatoSelecionado}">
        <h:panelGrid columns="2">

            <h:outputLabel value="ID:" for="contato_id" />
            <h:outputText id="contato_id"
                value="#{listaContatosBean.contatoSelecionado.id}" />

            <h:outputLabel value="Nome:" for="contato_nome" />
            <h:outputText id="contato_nome"
                value="#{listaContatosBean.contatoSelecionado.nome}" />

            <h:outputLabel value="Telefone:" for="contato_telefone" />
            <h:outputText id="contato_telefone"
                value="#{listaContatosBean.contatoSelecionado.telefone}" />

            <h:outputLabel value="E-mail:" for="contato_email" />
            <h:outputText id="contato_email"
                value="#{listaContatosBean.contatoSelecionado.email}" />

        </h:panelGrid>
    </rich:panel>
</a4j:outputPanel><!-- Continua... -->
```

# Interfaces *web* na plataforma Java EE

- RichFaces

index.xhtml

```
</h:panelGrid>
<hr />
<i>Desenvolvimento de Aplicações Corporativas com a Plataforma Java
    EE 6</i>
</h:body>
</html><!-- Fim da página index.xhtml -->
```



# Interfaces *web* na plataforma Java EE

- RichFaces

ListaContatosBean.java

```
package br.com.vinyanalista.agenda.web.mb;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import javax.faces.event.AjaxBehaviorEvent;

import org.richfaces.component.UIExtendedDataTable;

@ManagedBean
@ViewScoped
public class ListaContatosBean implements Serializable {
    private static final long serialVersionUID = 1L;

    public List<Contato> getContatos() {

        // Fornece a lista de contatos

    } //Continua...
```

# Interfaces *web* na plataforma Java EE

- RichFaces

ListaContatosBean.java

```
private Collection<Object> selecao;  
  
public Collection<Object> getSelecao() {  
    return selecao;  
}  
  
public void setSelecao(Collection<Object> selecao) {  
    this.selecao = selecao;  
}  
  
private Contato contatoSelecionado = null;  
  
public Contato getContatoSelecionado() {  
    return contatoSelecionado;  
} //Continua...
```

# Interfaces *web* na plataforma Java EE

- RichFaces

ListaContatosBean.java

```
public void atualizarContatoSelecionado(AjaxBehaviorEvent evento) {
    UIExtendedDataTable dataTable = (UIExtendedDataTable) evento
        .getComponent();
    Object linhaAtual = dataTable.getRowKey();
    contatoSelecionado = null;
    for (Object linhaSelecionada : selecao) {
        dataTable.setRowKey(linhaSelecionada);
        if (dataTable.isRowAvailable()) {
            contatoSelecionado = (Contato) dataTable.getRowData();
        }
    }
    dataTable.setRowKey(linhaAtual);
}

} //Fim da classe ListaContatosBean
```

# Interfaces *web* na plataforma Java EE

- RichFaces

## WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  id="WebApp_ID" version="3.0">

  <display-name>exemplo11</display-name>

  <!-- Configuração do JSF -->

  <context-param>
    <param-name>org.richfaces.enableControlSkinning</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <param-name>org.richfaces.skin</param-name>
    <param-value>blueSky</param-value>
  </context-param>

</web-app>
```

# Interfaces *web* na plataforma Java EE

- RichFaces

Terminal

```
$ exemplo11/implantar.sh
```

Navegador

<http://localhost/exemplo11>

## Agenda de Contatos

| Nome             | Telefone   | E-mail           |
|------------------|------------|------------------|
| Antonio Vinicius | 1212341234 | vinyanalista@gm  |
| Joao             | 5656785678 |                  |
| Jose             | 1234567890 | jose@exemplo.com |

### Dados do contato

ID: 0  
Nome: Antonio Vinicius  
Telefone: 1212341234  
E-mail: vinyanalista@gmail.com

*Desenvolvimento de Aplicações Corporativas com a Plataforma Java EE 6*

# Roteiro

- ✓ **Introdução à Plataforma Java EE**
- ✓ **A aplicação de exemplo "Agenda de Contatos"**
- ✓ **Apresentação das ferramentas utilizadas**
- ✓ **Persistência com JPA**
- ✓ **Validação de dados com Bean Validation**
- ✓ **Lógica de negócios com EJB**
- ✓ **Apresentação com JSF**
- ➔ **Implementação de um CRUD com JSF**





HANDS ON!!!



# Implementação de um CRUD com JSF

- HANDS ON!

Terminal

```
$ final/implantar.sh
```

Navegador

<http://localhost/agenda>


## Agenda de Contatos


### Página Inicial

| Contatos |                  |                  |            |  |
|----------|------------------|------------------|------------|--|
| ID       | Nome             | E-mail           | Telefone   |  |
| 1        | Antonio Vinicius | vinyanalista@gm  | 1212341234 |  |
| 2        | Joao             |                  | 5656785678 |  |
| 3        | Jose             | jose@exemplo.com | 1234567890 |  |
|          |                  |                  |            |  |

 Adicionar

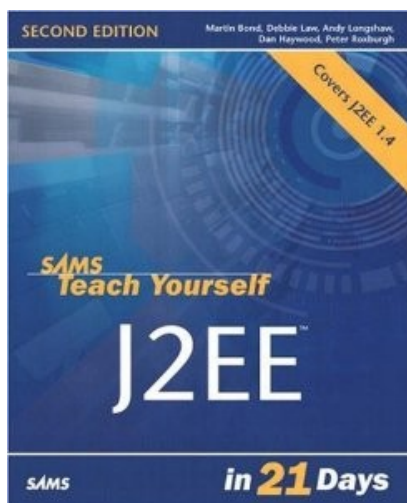
 Visualizar

 Alterar

 Remover



# Referências

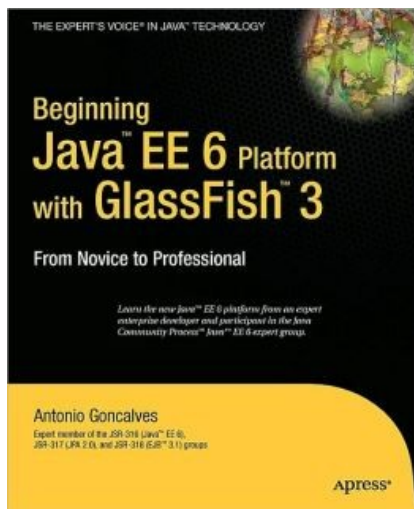


BOND, M. et al; *Sams Teach Yourself J2EE in 21 Days*. 2. ed. ISBN 0672325586. Indianapolis: Sams Publishing, 2004.

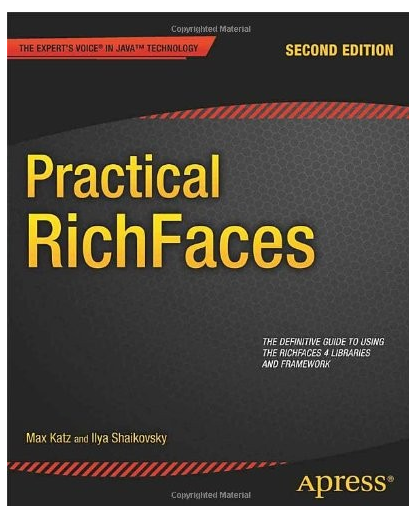


DEITEL, H. M.; DEITEL, P. J. *Java – Como Programar*. 6. ed. ISBN 8576050196. São Paulo: Prentice Hall, 2005.

# Referências



GONÇALVES, A. *Beginning Java EE 6 Platform with GlassFish 3: From Novice to Professional*. 2. ed. ISBN 9781430219545. New York: Apress, 2009.



KATZ, M.; SHAIKOVSKY, I. *Practical RichFaces*. 2. ed. ISBN 9781430234494. New York: Apress, 2011.

# Referências

- Outras referências:
  - Apostilas da Caelum de Java
    - <http://www.caelum.com.br/apostilas/>
  - Tutorial oficial da plataforma Java EE 6
    - <http://docs.oracle.com/javaee/6/tutorial/doc/>
  - Documentação do JBoss AS 7.1
    - <https://docs.jboss.org/author/display/AS71/Documentation>
  - *Site* do autor
    - <http://www.vinyanalista.com.br/>

OBRIGADO!!!



Antônio Vinícius Menezes Medeiros  
vinyanalista@gmail.com

