Java Enterprise Edition 5 (Java EE 5)

Prof. Rogério Ranthum

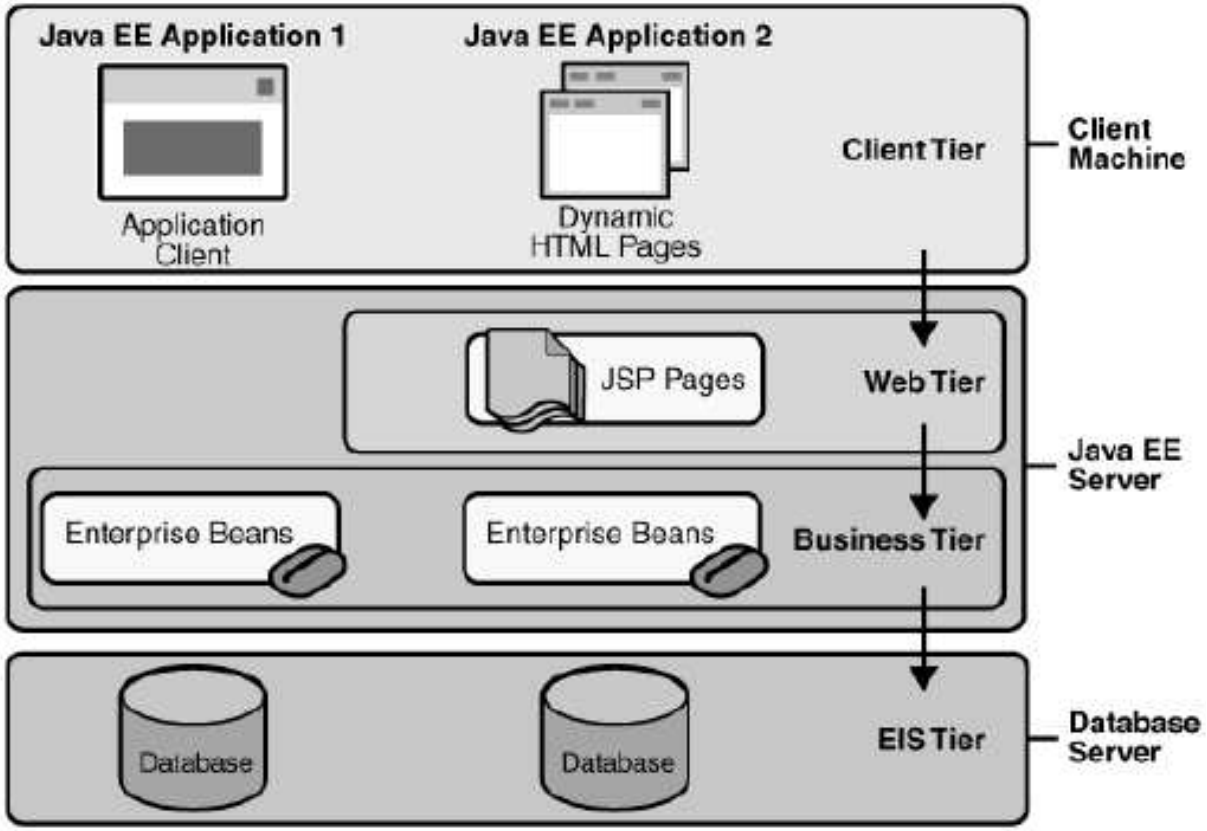
rogerio@pg.cefetpr.br

APLICAÇÕES DISTRIBUÍDAS

- A plataforma JEE utiliza um modelo de aplicação distribuída multicamada.
- A lógica da aplicação é dividida em componentes de acordo com a sua função.
- Os vários componentes que constituem uma aplicação JEE podem ser instalados em diferentes equipamentos.

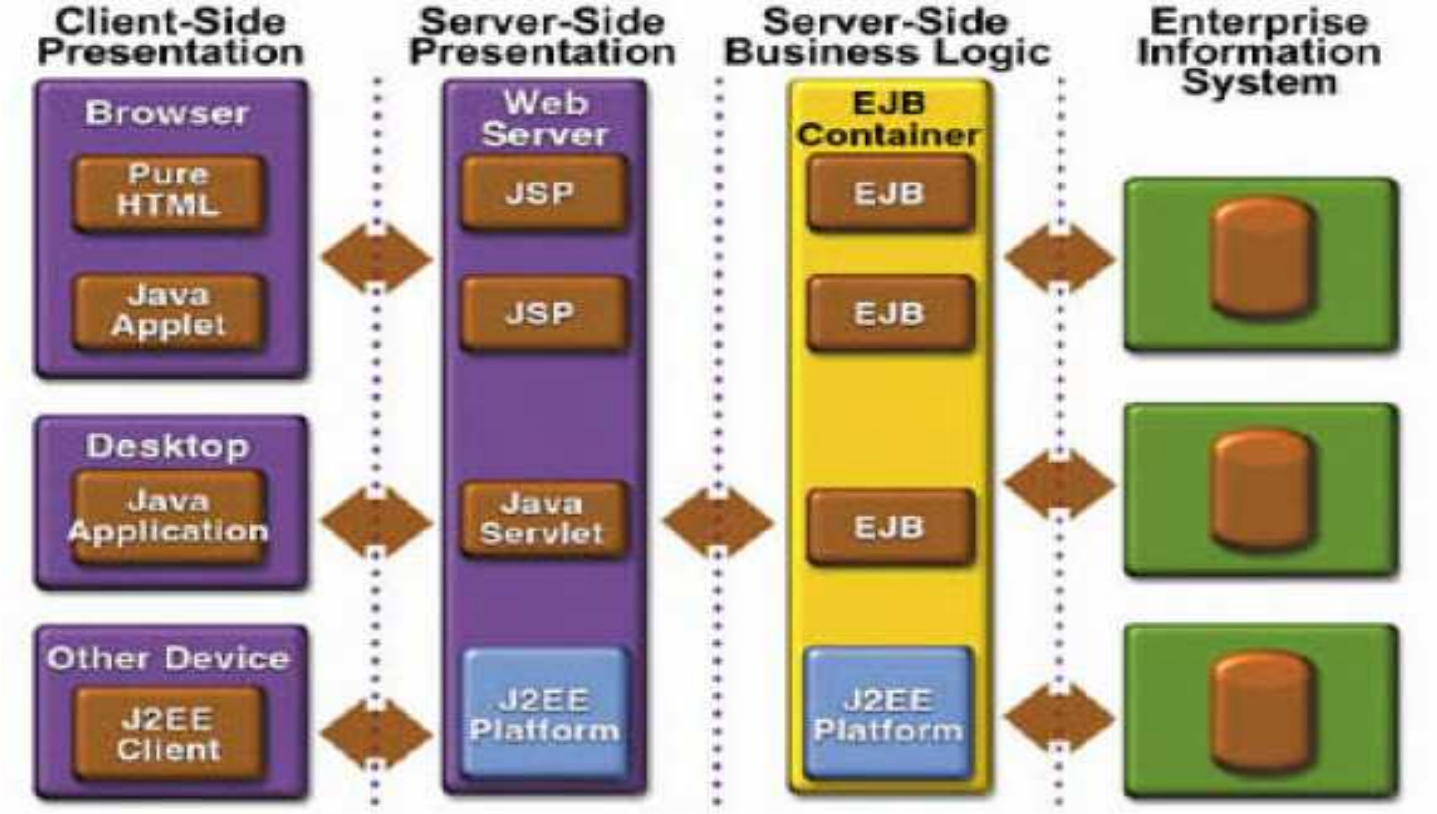
APLICAÇÕES DISTRIBUÍDAS

- Nível Físico



APLICAÇÕES DISTRIBUÍDAS

- Nível Lógico



MODELO JEE

- O modelo de aplicação JEE divide as aplicações corporativas em três partes:
  - Componentes
  - Containers
  - Conectores

COMPONENTES JEE

- As aplicações JEE são compostas de componentes.
- Um componente JEE é uma unidade de software funcional independente que é montada em uma aplicação JEE com seus arquivos e classes relacionados e que se comunica com outros componentes.

COMPONENTES JEE

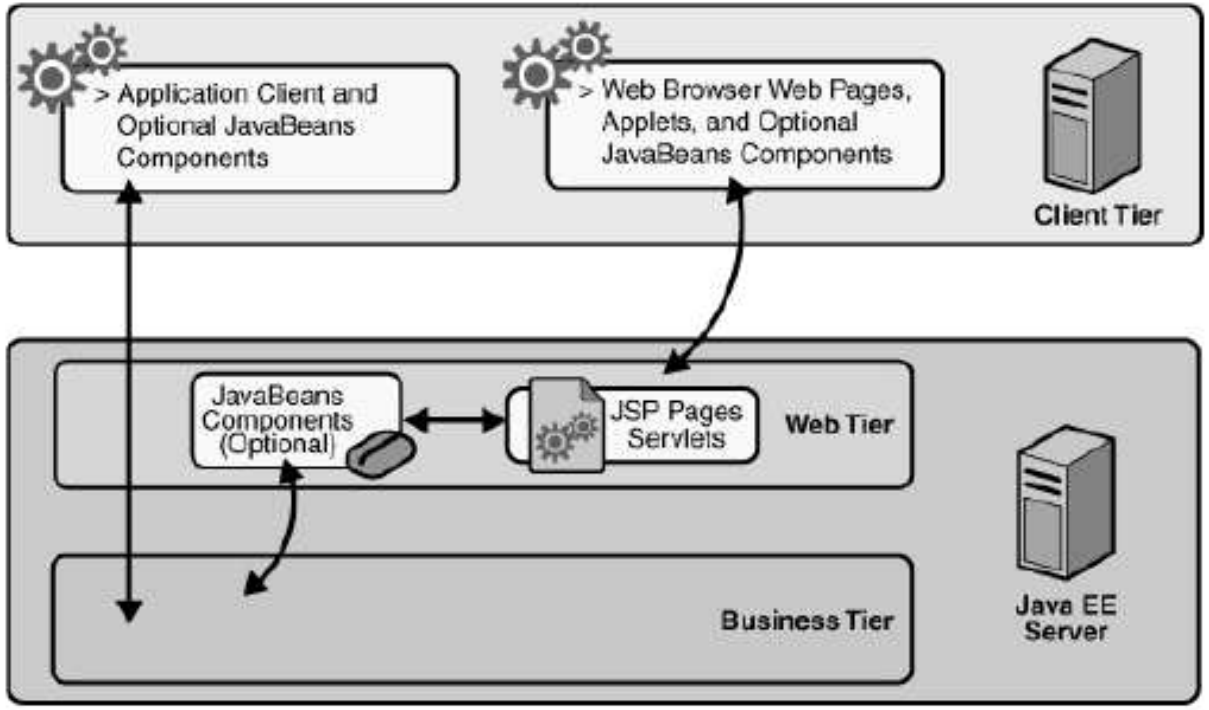
- A especificação JEE define três tipos de componentes:
  - Clientes JEE
  - Componentes WEB
  - Componentes de Negócios

CLIENTES JEE

- Podem ser um cliente Web (HTML, XML, Applets, etc.) ou um cliente de aplicação (graphical stand-alone application).

COMPONENTES WEB

- Podem ser servlets ou páginas JSP.

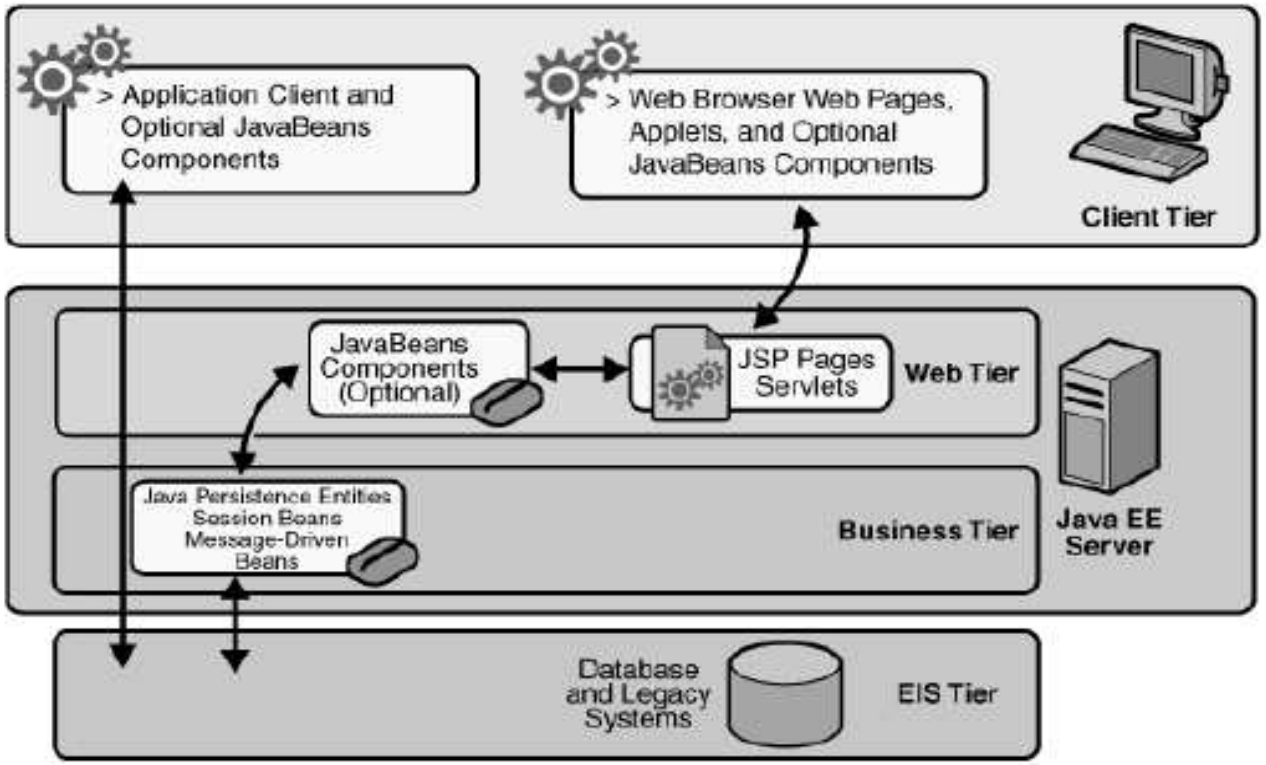


COMPONENTES DE NEGÓCIOS

- A lógica de negócios que soluciona e atende às necessidades de um domínio de negócios
- específico (bancárias, de varejo, financeiras, etc.) é tratado por enterprise java beans executando na camada de negócios.

COMPONENTES DE NEGÓCIOS

- Camada de Negócios



COMPONENTES JEE

- São responsáveis por tratar:
  - Apresentação
  - Lógica de Negócio
  - Acesso as Informações

CAMADA EIS

- Representa o software de EIS (Enterprise Information Systems) e inclui sistemas de infra-estrutura empresarial como sistemas de banco de dados, ERP, processamento de transações em mainframe, sistemas de informações legados, etc.

CONTAINER JEE

- Os Containers ficam localizados entre os componentes e os clientes, provendo serviços transparentes para ambos, incluindo serviços transacionais e pooling de recursos (reuso).

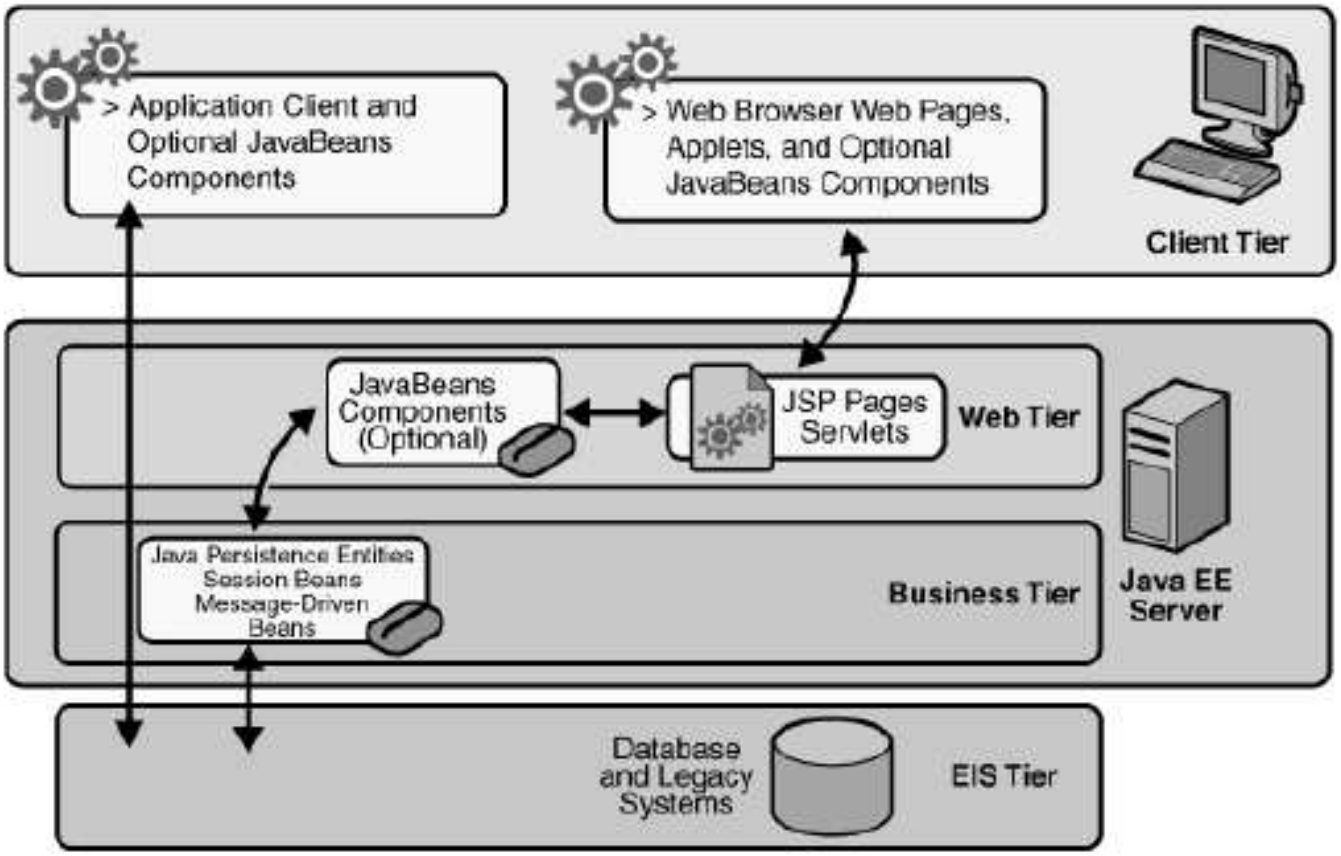
CONTAINER JEE

- Responsável por tratar:
  - Concorrência
  - Consistência
  - Segurança
  - Disponibilidade
  - Escalabilidade
  - Administração
  - Integração
  - Distribuição

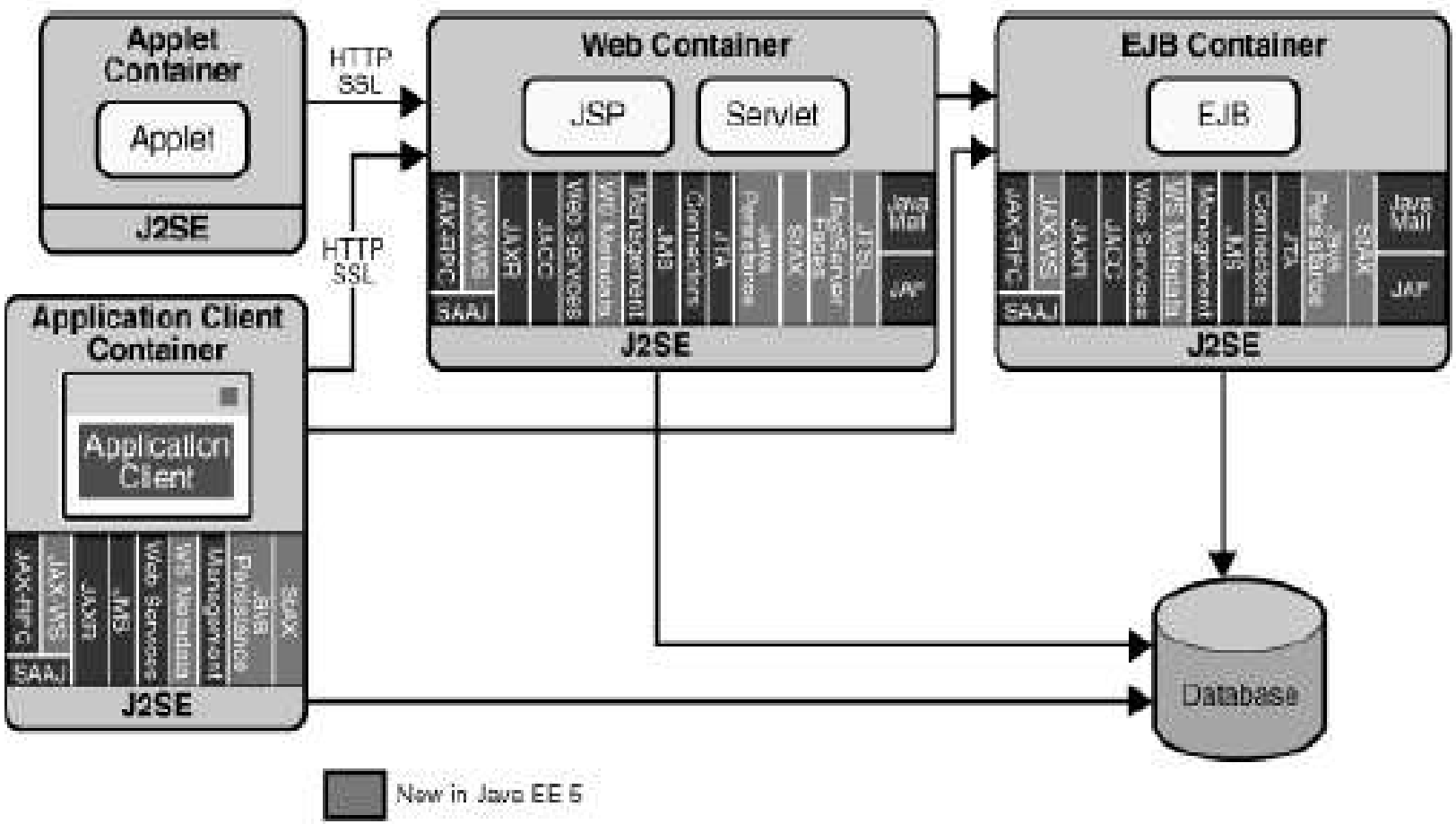
CONTAINER JEE

- São classificados em dois tipos:
- CONTAINER WEB
  - Gerencia a execução dos componentes JSP e Servlet.
    - Jakarta Tomcat, Caucho Resin, etc.
- CONTAINER EJB
  - Gerencia a execução dos Enterprise Beans para aplicações JEE.
    - JBoss
    - Sun Application Server

CONTAINER JEE



API´s JEE


Applet Container
Applet
J2SE
HTTP
SSL
HTTP
SSL
Web Container
JSP
Servlet
J2SE
EJB Container
EJB
J2SE
Application Client Container
Application Client
J2SE
Database
Now in Java EE 5

APIs JEE

- Java Database Connectivity (JDBC)
- Java Servlets
- Java Server Pages (JSP)
- Enterprise JavaBeans (EJB)
- Java Messaging Services (JMS)
- Java Transaction API (JTA)
- Java Naming and Directory Interface (JNDI)
- JavaMail
- Java Beans Activation Framework
- Java API for XML Processing (JAXP)
- JEE Connector Architecture
- Java Authentication and Autorization Service (JAAS)

BENEFÍCIOS

- O padrão da plataforma JEE traz inúmeros benefícios:
  - Arquitetura e desenvolvimento simplificado;
  - Escalabilidade para atender a variações da demanda;
  - Integração com sistemas de informação pré-existentes
  - Flexibilidade na escolha de servidores, ferramentas e componentes;
  - Modelo de segurança flexível

Enterprise Java Beans (EJB)

- Arquitetura para componentes no lado servidor;
- Possibilita e simplifica o processo de construir aplicações de objetos distribuídos
- Usando EJB, pode-se escrever aplicações escaláveis, robustas e seguras sem escrever sua própria infra-estrutura complexa para objetos distribuídos
- EJB é desenvolvido para prover portabilidade e reusabilidade qualquer que seja o vendedor de serviços corporativos da camada do meio, ou seja, da middlleware.

Definição EJB

- “Enterprise JavaBeans (EJB) technology defines a model for the development and deployment of reusable Java server components”.
- “Components are pre-developed pieces of application code that can be assembled into working application systems”. (Sun)

Introdução ao EJB

- Especificação EJB 1.0, 1.1, 2.0, 3.0
- Tecnologia de componentes Server Side
- Evolução dos Monitores de Transações
- Suporte a Objetos Distribuídos CORBA, DCOM e RMI
- Independência de Dados (JDBC)
- Padrão de Fato da Indústria

Introdução ao EJB

- Características desejáveis
  - Redução da complexidade de sistemas
  - Maior reuso de componentes
  - Facilidade de alteração de componentes
  - Maior portabilidade e interoperabilidade
  - Distribuição do processamento
  - Centralização da gerência e controles

Introdução ao EJB

- Servicos e Funcionalidades

Introdução ao EJB

- Quem é quem em Enterprise JavaBeans
  - O Provedor de Beans;
  - O Provedor de Container;
  - O Provedor de Servidor;
  - O Montador de Aplicações;
  - O Disponibilizador (Deployer)
  - O Administrador do Sistema

Introdução ao EJB

- O Provedor de Beans
- Características:
  - Parte que Fornece os Enterprise Beans;
  - Pode ser uma Empresa Vendedora de Componentes;
  - Pode ser uma Equipe de Desenvolvimento;
  - Pode ser um Programador Individual.

Introdução ao EJB

- O Provedor de Container e Servidores EJB
- Container EJB x Servidor EJB
- Características:
  - Gerenciamento de Transações Distribuídas;
  - Controle de Segurança;
  - Gerenciamento de Recursos e Gerenciamento do Ciclo de Vida dos Componentes
  - Controle de Persistência;
  - Acessibilidade Remota e Suporte a Múltiplos Clientes;
  - Implícita Localização dos Componentes.

Introdução ao EJB

- Quem é quem para o EJB.
  - O Disponibilizador de EJB;
  - O Administrador do Sistema.

Introdução ao EJB

- Características / Considerações:
- São diferentes dos JavaBeans.
  - Seguem o modelo de componentes da Sun:
  - Especificação (APIs, Encapsulamento, hiding, etc) regras para construir, gerenciar e manter os componentes.
  - Interação entre os componentes (interfaces)
  - Tipos : Client-side componentes (JavaBeans) e server-side componentes (Enterprise JavaBeans)
  - São instalados, acessados e executados dentro do Container do EJB Server
- Em resumo: Permite aos desenvolvedores pensar além dos componentes e suas interações, e desenvolverem estas classes para implementação de todas as funcionalidades da aplicação

Tipos de Beans EJB

- Enterprise Beans
  - Session Beans
    - Session Beans Statefull
    - Session Beans Stateless
  - Entity Beans
    - Entity Bean BMP (Bean-Managed Persistence)
    - Entity Bean CMP (Container-Managed
    - Persistence)
  - Mensage-Driven Bean

Session Beans

- Características
  - São usados para descrever interações entre beans.
  - Representam através dos seus métodos o Workflow do componente.
  - Workflow - descreve um fluxo completo de como fazer uma tarefa
  - Podem ser Stateless ou Stateful, mas não podem ser persistentes (entity beans).

Session Beans

- Stateless
  - São simples e leves, fáceis de serem desenvolvidos e eficientes, genéricos e reutilizáveis.
  - São uma coleção de serviços (métodos) podem ser relacionados mas não interdependentes.
  - São um conjunto de procedimentos, serviços de software que não necessitam variáveis de instância ou qualquer outro tipo de estado interno.
  - Podem ser utilizados por exemplo para gerar relatórios, processamento batch.

Session Beans

- Stateful
  - É uma extensão da aplicação do cliente, executa tarefas mas mantém o estado do Objeto.
  - conversationalstate: porque representa o state do objeto através dos métodos invocados pelo cliente
  - conversationalstate pode ser compartilhado (leitura e alteração) através de todos os métodos do Statefull Bean
  - Estão entre os Stateless e os entity beans

Entity Beans

- Descrevem objetos do mundo real (lugar, pessoa ou coisa) com comportamentos, estados e características.
- São beans (componentes) persistentes.
- Representam os beans no banco de dados.
- Provêem uma interface para facilitar o desenvolvedor criar, modificar e apagar dados de um banco de dados.
- Encapsulam os dados e as regras de negócios.

Entity Beans

- Container-Managed Persistence
  - A persistência é automaticamente controlada pelo EJB Container.
  - O container deve saber quais campos podem e precisam persistir, faz o mapeamento com o banco de dados se necessário e cuida da inserção, atualização e deleção no banco de dados.
  - Mais simples pois o desenvolvedor pode se preocupar mais com a lógica do negócio e deixa a cargo do container as questões relativas a persistência.
  - A desvantagem é que o container precisa de um sofisticado mecanismo de controle e mapeamento dos objetos para bancos de dados relacionais.

Entity Beans

- Bean-managed persistence
  - Mais complicado que o Container-manage porque é preciso escrever explicitamente dentro da classe EJB.
  - Torna mais flexível a utilização de beans que envolvam vários joins entre tabelas e bancos de dados heterogêneos e integração com sistemas legados.
  - São obrigatoriamente utilizados quando o container é insuficiente ou inadequado para fazer o serviço de persistência.
  - Os métodos do bean contém os comandos SQL necessários para implantação da persistência.

Entity Beans x Session Beans

- Session Beans
  - Representa um cliente especifico
  - Não persistente
  - Pode ser transacional
  - Representa a lógica propriamente dita do Bean.
- Entity Beans
  - Clientes dividem uma instância
  - Persistente
  - São sempre transacionais
  - Pode ser uma classe que faz o mapeamento a dados persistentes.

Vantagens do EJB

- Estabelece papeis bem definidos para o desenvolvimento (EJB Developer, EJB Deployer, EJB Container Vendor, EJB Server Vendor, Application Developer).
- Gerência de Transações automática
- Suporte a transações distribuídas
- Portabilidade entre fabricante de EJB Server e JEE compatíveis.
- Escalabilidade
- Integração com RMI e CORBA.

Criando um EJB - Session Bean

- Cada enterprise bean requer as seguintes classes
  - uma remote interface;
  - uma home interface;
  - a classe do enterprise bean;
  - Servlet de acesso;

Criando a Remote Interface

- Uma interface remota define os métodos de negócio que um cliente pode chamar.
- Os métodos de negócio são implementados no código da classe do enterprise bean.
- O código fonte da classe Converter é o seguinte:

Criando a Remote Interface

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;


public interface Converter extends EJBObject {


public double dollarToYen(double dollars) throws RemoteException;


public double yenToEuro(double yen) throws RemoteException;
}
```

Criando a Home Interface

- Uma home interface define os métodos que permitem que o cliente crie, localize, ou remova um enterprise bean.
- A interface ConverterHome contém um único método create que retorna um objeto do tipo da interface remota.
- Abaixo está o código fonte da classe ConverterHome.

Criando a Home Interface

```
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;


public interface ConverterHome extends EJBHome {


 Converter create() throws RemoteException, CreateException;


}
```

Criando a Classe do Enterprise Bean

- O enterprise bean do nosso exemplo é um stateless session bean, chamado de ConverterEJB.
- Este bean implementa dois métodos de negócio dollarToYen e yenToEuro, que a remote interface Converter define.
- E também implementa o método create que a home interface define, e no enterprise bean é chamado de ejbCreate.
- O código fonte da classe ConveterEJB está listado abaixo.

Criando a Classe do Enterprise Bean

```
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class ConverterEJB implements SessionBean {

public double dollarToYen(double dollars) {

return dollars * 121.6000;
}

public double yenToEuro(double yen) {

return yen * 0.0077;
}
```

Criando a Classe do Enterprise Bean

```
public ConverterEJB() {}

public void ejbCreate() {}

public void ejbRemove() {}

public void ejbActivate() {}

public void ejbPassivate() {}

public void setSessionContext(SessionContext sc) {}
}
```

Criando Aplicação JEE:Session Bean – Stateless - Remote

- Projeto:
  - Nome do Projeto: ConverterApp (Aplicação JEE)
  - Enterprise Java Bean: ConverterBean;
  - Servlet: ConverterServlet

Criando Aplicação JEE:Session Bean – Stateless - Remote

- Passo:
  - New → File → New Project → Enterprise → Enterpise Application;
    - Nome do Projeto: ConverterApp
      - Estrutura:
        - Enterprise Application;
        - EJB Module;
        - Web Module.

Criando o Enterprise Bean (EJB)

- Session Bean Stateless : ConverterBean
  - Gerar as classes bean e suas interfaces;
  - Adicionar os métodos de negócio no EJB
    - Em ConverterApp-EJBModule → Botão da direita do mouse → New → Session Bean
      - Nome: Converter;
      - Package: converter;
      - Opções: Stateless e Remote
        - ConverterBean.java,
        - ConverterRemote.java,
        - ConverterRemoteBussiness.java,
        - ConverterRemoteHome.java.

Adicionando Métodos de Negócio.

- ConverterSB → Botão da Direita do mouse → Add → Bussines Method;
  - Nome Campo: dollarToYen;
  - Tipo de retorno: BigDecimal;
  - Parâmetros: dollars, BigDecimal
  - Adicionar o método yenToEuro, seguindo os passos anteriores, parêmetro yen.
- Abrir ConverterRemoteBusiness: verificar métodos criados automaticamente.

Implementar a Classe Bean

- Abrir a classe: ConverterBean.java
  - Declarar as variáveis
  - Implementar os dois métodos.

Implementar a Classe Bean

```
public class ConverterBean implements converter.ejb.Converter {
    private BigDecimal euroRate = new BigDecimal("0.0070");
    private BigDecimal yenRate = new BigDecimal("112.58");

    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);

        return result.setScale(2, BigDecimal.ROUND_UP);
    }
    public BigDecimal yenToEuro(BigDecimal yen) {
        BigDecimal result = yen.multiply(euroRate);

        return result.setScale(2, BigDecimal.ROUND_UP);
    }
}
```

Criando o Cliente Web (Servlet)

- Em ConverterApp-WebModule: botão da direita do mouse → New → Servlet;
- Nome: ConverterServlet;
- Package: converter;
- Manter opções default e na próxima página finalizar o processo.

Localizando a Home Interface

- Abrir a classe ConverterServlet;
- Clicar com o botão direito do mouse em qualquer lugar da classe → Enterprise Resources → Call Enterprise Bean;
- Na caixa de diálogo selecione ConverterSB e clique em OK.

Localizando a Home Interface

- Nesta etapa a IDE adiciona o método lookupConverterBean, no servlet;
- Registra a referência ao bean no deployment descriptor do modulo web.

Localizando a Home Interface

- Finalidades do código lookup:
  - Criar contexto naming inicial:
    - javax.naming.Context c = new javax.naming.InitialContext();
  - Obter ambiente para o contexto naming para o cliente web e retornar o limite para o objeto ejb/CoverterBean:
    - Object remote = c.lookup((“java:comp/env/ejb/ConverterBean”);
  - Narrow referência para o objeto ConverterRemoteHome:
    - converter.ConverterRemoteHome rv = (converter.ConverterRemoteHome) javax..rmi.PortableRemoteObject.narrow(remote, converter.ConverterRemoteHome.class);
  - Criar instância do EJB ConverterBean:
    - return rv.create();

Invocando os métodos de negócio, através da construção do servlet.

Código do ConverterServlet.java

```
out.println("<h1><b><center>Converter</center></b></h1>");
out.println("<hr>");
out.println("<p>Enter an amount to convert:</p>");
out.println("<form method=\"get\">");
out.println("<input type=\"text\"
name=\"amount\" size=\"25\">");
out.println("<br>");
out.println("<p>");
out.println("<input type=\"submit\" value=\"Submit\">");
out.println("<input type=\"reset\" value=\"Reset\">");
out.println("</form>");
String amount = request.getParameter("amount");
```

Invocando os métodos de negócio, através da construção do servlet.

```
if ( amount != null && amount.length() > 0 ) {
try {
converter.ConverterRemote converter;
converter = lookupConverterBean();
java.math.BigDecimal d = new java.math.BigDecimal(amount);
out.println("<p>");
out.println("<p>");
out.println(amount + " Dollars are "+ converter.dollarToYen(d) + " Yen.");
out.println("<p>");
out.println(amount + " Yen are " + converter.yenToEuro(d) + " Euro.");
converter.remove();
} catch (Exception e){
out.println(“Não pode localizar ou executar EJB!");
}
}
```

Especificando a URL default da aplicação

- No projeto ConverterApp → botão da direita do mouse → Properties → Run;
- No campo Relative URL digitar: /ConverterServlet e OK;
- Executar o projeto.

Criando um Entity Bean – Bean Managed Persistence

- Projeto: SavingsAccount;
- File → New Project → Enterprise → Enterprise Apliccation;
- Botão da Direita em SavingsAccont-EJBModule → New → Entity Bean;
  - EJB Name: SavingsAccount;
  - Package: bank;
  - Primary Key Class: String;
  - Persistence Type: Bean;
  - Create Interface: Romote e Local

Criando um Entity Bean – Bean Managed Persistence

- Database Lookup:
  - No corpo da classe bean, botão da direita do mouse → Enterprise Resources → Use Database.
  - Gera método DataSource;
  - Criar métodos MakeConnection() e ReleaseConnection();

Criando um Entity Bean – Bean Managed Persistence.

- Métodos de Acesso ao Banco:
  - Declarar campos da tabela;
  - Gerar Gets/Sets;
  - Em cada método get, botão da direita do mouse → EJB Methods → Local e Remote;
  - Adiconar métodos de acesso ao banco, do arquivo SavingsAccoutBean.java
  - Gera métodos EjbCreate, EjbPostCreat, EjbLoad e EjbRemove.

Criando um Entity Bean – Bean Managed Persistence.

- Gerar Métodos Finder
  - Botão da direita na classe Bean → Add Finder Method;
    - Gerar os métodos findLastName e FindInRange;
    - Implementar o método FindByPrimaryKey

Criando um Entity Bean – Bean Managed Persistence.

- Gerar os Bussines Methods
  - Gerar o método Exception
    - Em cima do projeto, botão da direita do mouse → New File/Folder → Java Class → Java Exception;
      - Classe: InsufficientBalanceException;
      - Package: bank;
  - No corpo da classe bean, botão da direita do mouse → Add Bussines Methods:
    - Criar o métodos debit e credit;

Criando um Entity Bean – Bean Managed Persistence.

- Home Methods
  - No corpo da classe bean, botão da direita do mouse → Add Home Methods
    - Gerar o método: chargeForLowBalance;

Criando um Entity Bean – Bean Managed Persistence.

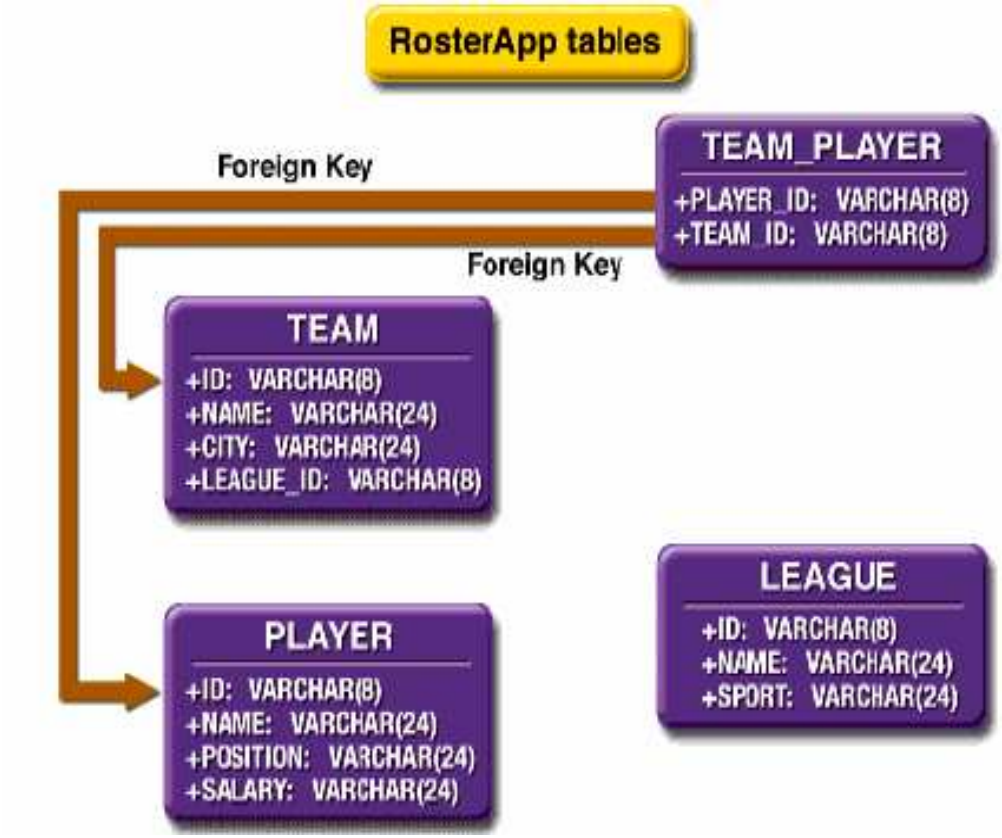
- Criar banco de dados;
- Deploy da aplicação;
  - Configurar data source e pool conection no server;
- Rodar o cliente:
  - Abrir projeto;
  - Resolver problemas com referências;
  - Rodar cliente.

Criando um Entity Bean – Container-Managed Persistence.

- Criando o projeto: File → New → Project;
  - Enterprise → EJB Module → Next:
    - Name: Roster.

Criando um Entity Bean – Container-Managed Persistence.

- Criar BD: Gerar através de script.



Criando um Entity Bean – Container-Managed Persistence.

- Gerando o CMP – Entity Bean:
  - No projeto botão da direita do mouse → New → CMP Entity Bean From Database.
    - JDBC connection, selecione seu BD.
    - Package : team;
    - Na proxima tela selecione todas as tabelas.
    - Presione Add e termine.

Criando um Entity Bean – Container-Managed Persistence.

- Métodos de Acesso: Criando as cardinalidades:
  - Na janela do projeto, expanda configuration files;
  - Duplo clique em ejb-jar.xml;
  - Clique em CMP Relationships
  - Selecione TeamPlayer → Multiplicity → Many to Many;
  - Encontre a regra PlayerId e altere o nome do campo de teamId para teams.
  - Encontre a regra TeamId e altere o nome do campo de playerId para players.
  - No CMP relationship: TeamBean—LeagueBean
    - Encontre a regra TeamBean e altere o nome do campo para league;
    - Enconte a regra LeagueID e altere o nome do campo para teams

Criando um Entity Bean – Container-Managed Persistence.

- Finder e Select Methods:
  - Duplo clique em: ejb-jar.xml, na aba General;
  - Expanda a seção PlayerEB expanda Finder Methods → Add;
  - Utilize a tabela a seguir para incluir novos Finder Methods.

Criando um Entity Bean – Container-Managed Persistence.

- Tabela:

Name	Cardinality	EJBQL	Parameters
findAll	Many	`select object(p) from Player p`	none
findByCity	Many	`select distinct object(p) from Player p, in (p.teams) as t where t.city = ?1`	String city
findByHigherSalary	Many	`select distinct object(p1) from Player p1, Player p2 where p1.salary > p2.sal- ary and p2.name = ?1`	String name
findByLeague	Many	`select distinct object(p) from Player p, in (p.teams) as t where t.league = ?1`	team.League Local league
findByPositionAnd- Name	Many	`select distinct object(p) from Player p where p.position = ?1 and p.name = ?2`	String position, String name
findBySalaryRange	Many	`select distinct object(p) from Player p where p.salary between ?1 and ?2`	double low, double high

Criando um Entity Bean – Container-Managed Persistence.

- Tabela Continuação:

Name	Cardinality	EJBQL	Parameters
findBySport	Many	`select distinct object(p) from Player p, in (p.teams) as t where t.league.sport = ?1`	`String sport`
`findByTest`	Many	`select distinct object(p) from Player p where p.name = ?1`	`String parm1, String parm2, String parm3`
`findNotOnTeam`	Many	`select object(p) from Player p where p.teams is empty`	`none`

Criando um Entity Bean – Container-Managed Persistence.

- Em PlayerEB feche o editor, expanda CMP Select Method → Add.
- Utilize a tabela a seguir.

Criando um Entity Bean – Container-Managed Persistence.

- Tabela:

Name	Return Type	EJBQL	Parameters
ejbSelectLeagues	java.util.Collection	select distinct t.league from Player p, in (p.teams) as t where p = ?1	team.PlayerLocal player
ejbSelectSports	java.util.Collection	select distinct t.league.sport from Player p, in (p.teams) as t where p = ?1	team.PlayerLocal player

Criando um Entity Bean – Container-Managed Persistence.

- Helper Classes:
  - Estas classes são necessárias para desenvolver o EJB;
  - Copiar os arquivos a pasta util, da pasta do professor para a pasta /src/java, do seu projeto;
  - Irá aparecer em Sources Package a package util.

Criando um Entity Bean – Container-Managed Persistence.

- Business Method:
  - Na classe PlayerBean → botão da direita → EJB Methods → Add Business Method:
    - Field: getLeagues;
    - Type: Collection;
    - Exception: Finder Exception
    - Edite o método da seguinte forma:

```
public Collection getLeagues() throws FinderException {
PlayerLocal player =
(PlayerLocal)context.getEJBLocalObject();
return ejbSelectLeagues(player);
}
```

Criando um Entity Bean – Container-Managed Persistence.

- Business Method:
  - Na classe PlayerBean → botão da direita → EJB Methods → Add Business Method:
    - Field: getSports;
    - Type: Collection;
    - Exception: Finder Exception
    - Edite o método da seguinte forma:

```
public Collection getSports() throws FinderException {
PlayerLocal player =
(team.PlayerLocal)context.getEJBLocalObject();
return ejbSelectSports(player);
}
```

Criando um Entity Bean – Container-Managed Persistence.

- Business Method:
  - Na classe TeamBean → botão da direita → EJB Methods → Add Business Method:
    - Field: addPlayer;
    - Type: void;
    - Parameter Type: PlayerLocal
    - Parameter Name: player
    - Edite o método da seguinte forma:

```
public void addPlayer(PlayerLocal player) {
Debug.print("TeamBean addPlayer");
try {
Collection players = getPlayers();
players.add(player);
} catch (Exception ex) {throw new EJBException(ex.getMessage()); }
}
```

Criando um Entity Bean – Container-Managed Persistence.

- Business Method:
  - Na classe TeamBean → botão da direita → EJB Methods → Add Business Method:
    - Field: dropPlayer;
    - Type: void;
    - Parameter Type: PlayerLocal
    - Parameter Name: player
    - Edite o método da seguinte forma:

```
public void dropPlayer(PlayerLocal player) {
    Debug.print("TeamBean dropPlayer");
    try {
        Collection players = getPlayers();
        players.remove(player);
        } catch (Exception ex) {
                    throw new EJBException(ex.getMessage());
        }
    }
```

Criando um Entity Bean – Container-Managed Persistence.

- Business Method:
  - Na classe TeamBean → botão da direita → EJB Methods → Add Business Method:
    - Field: getCopyOfPlayers;
    - Type: ArrayList;
    - Edite o método da seguinte forma:

```
public ArrayList getCopyOfPlayers() {
    Debug.print("TeamBean getCopyOfPlayers");
    ArrayList playerList = new ArrayList();
    Collection players = getPlayers();
    Iterator i = players.iterator();
    while (i.hasNext()) {
        PlayerLocal player = (PlayerLocal) i.next();
        PlayerDetails details =
        new PlayerDetails(player.getId(), player.getName(),
        player.getPosition(), 0.0);
        playerList.add(details);
    }
    return playerList;
    }
```

Criando um Entity Bean – Container-Managed Persistence.

- Business Method:
  - Na classe LeagueBean → botão da direita → EJB Methods → Add Business Method:
    - Field: addTeam;
    - Type: void;
    - Parameter Type: TeamLocal
    - Parameter Name: team
    - Edite o método da seguinte forma:

```
public void addTeam(team.TeamLocal team) {
    Debug.print("TeamBean addTeam");
    try {
        Collection teams = getTeams();
        teams.add(team);
    } catch (Exception ex) {
        throw new EJBException(ex.getMessage());
    }
    }
```

Criando um Entity Bean – Container-Managed Persistence.

- Business Method:
  - Na classe LeagueBean → botão da direita → EJB Methods → Add Business Method:
    - Field: dropTeam;
    - Type: void;
    - Parameter Type: TeamLocal
    - Parameter Name: team
    - Edite o método da seguinte forma:

```
public void dropTeam(team.TeamLocal team) {
    Debug.print("TeamBean dropTeam");
    try {
        Collection teams = getTeams();
        teams.remove(team);
    } catch (Exception ex) {
        throw new EJBException(ex.getMessage());
    }
    }
```

Criando um Entity Bean – Container-Managed Persistence.

- Criando o Session Bean RosterBean
  - Botão da Direita em Roster → New → Sesion Bean
    - Nome: Roster
    - Package: roster
    - Gerar interface remote e home
  - Botão da direita no corpo do RosterBean → Enterprise Resource → Call Enterprise Bean
    - Selecione LeagueEB;
    - Repita os passos para gerar o código lookup para PlayerEB e TeamEB

Criando um Entity Bean – Container-Managed Persistence.

- Declarar as seguintes variáveis:
  - private PlayerLocalHome playerHome = null;
  - private TeamLocalHome teamHome = null;
  - private LeagueLocalHome leagueHome = null;
- Alterar os métdos ejbCreate, ejbActivate e ejbPassivate para pegar e atualizar as referencias dos beans.

Criando um Entity Bean – Container-Managed Persistence.

- ejbCreate:

```
public void ejbCreate() {
    Debug.print("RosterBean ejbCreate");
    playerHome = lookupPlayerBean();
    teamHome = lookupTeamBean();
    leagueHome = lookupLeagueBean();
}
```

Criando um Entity Bean – Container-Managed Persistence.

- ejbActivate:

```
public void ejbActivate() {
    Debug.print("RosterBean ejbCreate");
    playerHome = lookupPlayerBean();
    teamHome = lookupTeamBean();
    leagueHome = lookupLeagueBean();
}
```

Criando um Entity Bean – Container-Managed Persistence.

- ejbPassivate:

```
public void ejbPassivate() {
    playerHome = null;
    teamHome = null;
    leagueHome = null;
}
```

Criando um Entity Bean – Container-Managed Persistence.

- Criar Métodos de negócio (Business Methods), para acessar o Entity Bean;
  - Copiar os métodos do RosterBean
    - Inicia em testFinder e termina em copyPlayerToDetails
    - Substituir o conteúdo do RosterRemoteBusiness, com o conteúdo do mesmo arquivo do exemplo.
    - Alt+Shift+F → importar pacotes

Criando um Entity Bean – Container-Managed Persistence.

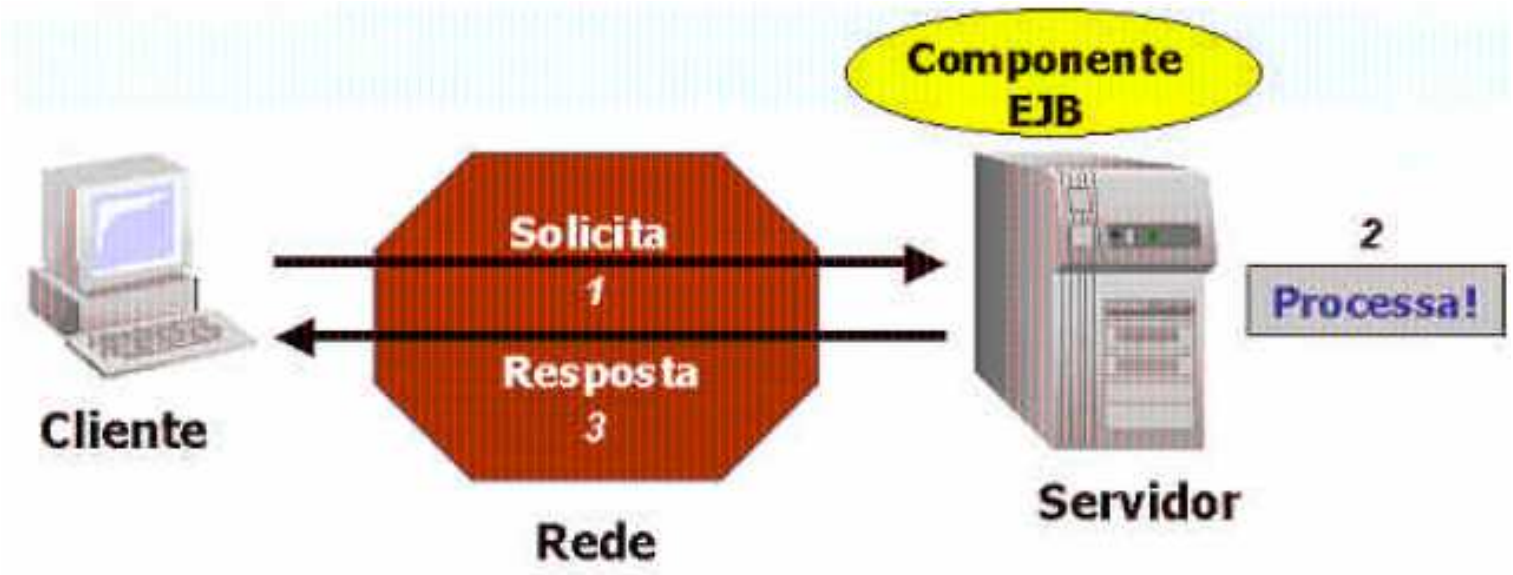
- Criar os Business Methods:
  - Copiar os métodos do arquivo RosterBean.java da pasta do professor;
  - Iniciar a cópia no método testFinder e terminar no método copyPlayersToDetails.

Implementação de um Mensage-Driven Bean

- Os componentes EJB vistos até aqui (Session Beans) usam um protocolo de invocação remota (RMI ou RMI-IIOP) para receber solicitações de aplicações clientes.
- Esses protocolos exigem uma comunicação síncrona e bloqueante, ou seja, o cliente deve ficar em espera enquanto o servidor realiza o processamento e envia a resposta ao cliente.

Implementação de um Mensage-Driven Bean

- A figura abaixo mostra a seqüência de uma comunicação síncrona:



Implementação de um Mensage-Driven Bean

- A sequência ilustrada na figura pode ser descrita da seguinte forma:

1. O cliente localiza o componente EJB no servidor;
2. O cliente solicita a execução de determinado método sobre o componente;
3. O componente processa, enquanto o cliente aguarda sua resposta;
4. O cliente recebe a resposta e só então está liberado para seguir seu processamento.

Implementação de um Mensage-Driven Bean

- Esse simples modelo traz duas limitações (ou desvantagens para os mais negativos): A primeira delas é o fato do cliente ter que esperar por uma resposta do componente para prosseguir sua execução.
- É verdade que em alguns casos esse espera é impossível de ser evitada, por exemplo, os casos onde o cliente depende da resposta do componente para prosseguir.
- No entanto, existem casos em que o cliente não precisa de nenhuma resposta do componente para continuar seu serviço.

Implementação de um Mensage-Driven Bean

- A segunda limitação está relacionada ao fato de o cliente deve conhecer a interface remota do componente para conseguir invocar algum método no EJB. Esse fato cria uma dependência entre esses dois elementos.
- Uma mudança da interface remota do componente deve ser refletida também na interface que o cliente conhece. O ponto negativo aqui é o acoplamento entre os clientes e os componentes.

Implementação de um Mensage-Driven Bean

- Os componentes Message-Driven Beans surgem com uma proposta para resolver esses problemas.
- A idéia é fazer a comunicação entre o cliente e o componente de forma assíncrona através da troca de mensagens entre os envolvidos.

Implementação de um Mensage-Driven Bean

- O modelo de troca de mensagens funciona, simplificadamente, da seguinte forma:
- O cliente, envia mensagens para uma estrutura que as armazena.
- Os componentes Message-Driven Beans processam conteúdos oriundos de uma fila de mensagens.
- Quando uma mensagem é recebida o Container se encarrega de entregá-la ao componente, que executa o código de negócio.

Implementação de um Mensage-Driven Bean

- A figura abaixo ilustra o funcionamento:

Implementação de um Mensage-Driven Bean

- A sequência ilustrada na figura pode ser descrita da seguinte forma:

1. A aplicação que deseja invocar um componente Message-Driven Bean envia uma mensagem para uma fila;
2. O servidor de aplicações recebe a mensagem e repassa para o componente; O componente processa a mensagem e nada retorna para o cliente.

Implementação de um Mensage-Driven Bean

- Veja que esse modelo soluciona os problemas relacionados anteriormente. Agora o cliente envia a mensagem para o servidor de aplicações sem ficar bloqueado esperando alguma resposta do componente.

Implementação de um Mensage-Driven Bean

- Além disso, os componentes Message-Driven Beans não possuem interface Home e Remote, pois nenhum protocolo de invocação remota é utilizado.
- Como eles processam mensagens oriundas de qualquer cliente que seja capaz de produzir uma mensagem JMS válida, existe o mínimo de dependência possível.
- Ex. Arquivo MD.java

Importar Pacotes

- j2ee.jar
- appserv-rt.jar
- appserv-admin.jar
- imqjmsra.jar