

# Tutorial J2EE

Aprendendo EJB de uma maneira fácil!

Paulo Silveira

# Aprendendo J2EE

RemoteException

ApplicationServer

SessionBean

JNDI

EJBLocalHome

CMP

ejb-jar.xml

RequiresNew

CMR

EJBContext

# Aprendendo J2EE

- Quem já tentou aprender EJBs?
- Maiores dificuldades?

# Objetivos desse tutorial

Público: quem não conhece EJB, ou quem já conhece, mas não sabe o que está fazendo (**extremamente** comum).

## **Superar as enormes dificuldades iniciais!**

- O que realmente é EJB?
- Quando preciso usar?
- Quando eu estiver codificando, quero saber realmente o que estou fazendo.
- Porque esse monte de interfaces e XMLs?

# Tutoriais comuns

Como **não** aprender J2EE

# Home Interface

```
public interface HelloHome extends  
    EJBHome {  
  
    Hello create() throws CreateException,  
        RemoteException;  
  
}
```

# Object Interface

```
public interface Hello extends EJBObject
{
    String sayHello() throws
        RemoteException;
}
```

# Component

```
public class HelloBean implements  
    SessionBean {  
    public void ejbCreate() throws  
        CreateException {}  
    public String sayHello() {  
        return "Olá Mundo";  
    }  
}
```



# ejb-jar.xml

...

```
<session>
```

```
  <ejb-name>Hello</ejb-name>
```

```
  <home>HelloHome</home>
```

```
  <remote>Hello</remote>
```

```
  <ejb-class>HelloBean</ejb-class>
```

```
</session>
```

...

Ou então usa uma ferramenta estranha para gerá-lo, e a gente não sabe o que acontece!

# Cliente

```
Context initialContext = new  
    InitialContext();  
HelloHome home = (HelloHome)  
    initialContext.get("HelloHome");  
Hello hello = home.create();  
System.out.println(Hello.sayHello());
```

Para que tudo isso?

????????????????

# Incrível resultado

Olá Mundo.

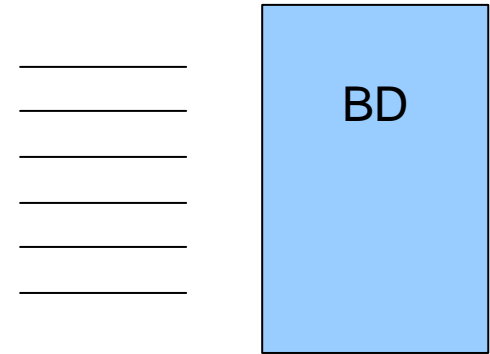
# Porque J2EE?

Motivação

Exemplo: abrindo conexões com BD

# Abrindo uma conexão para cada

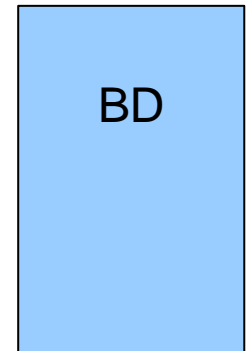
```
void save() {  
    Connection con =  
        DriverManager.getConnection("jdbc://my  
        sql...");  
    Statement s = con.createStatement();  
  
    ...  
}
```



The diagram illustrates a database connection. A light blue rectangle labeled 'BD' (Database) is connected to a series of horizontal lines, representing a data stream or query execution.

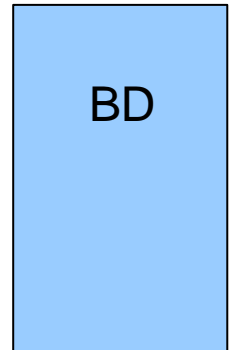
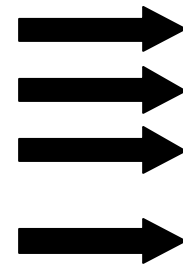
# Uma conexão para todos

```
static Connection con;  
void save() {  
    Statement s = con.createStatement();  
  
    ...  
}
```



# Pool de conexões

```
List list = new ArrayList();  
public synchronized Connection get() {  
    return (Connection) list.remove(0);  
}  
public synchronized void free(Connection  
    c) {  
    list.add(c);  
}
```





Qual é a melhor solução?

# Nenhuma das anteriores!

- Qual o número de conexões que devem ser abertos pelo Pool?
- Esses números podem variar com o dia!

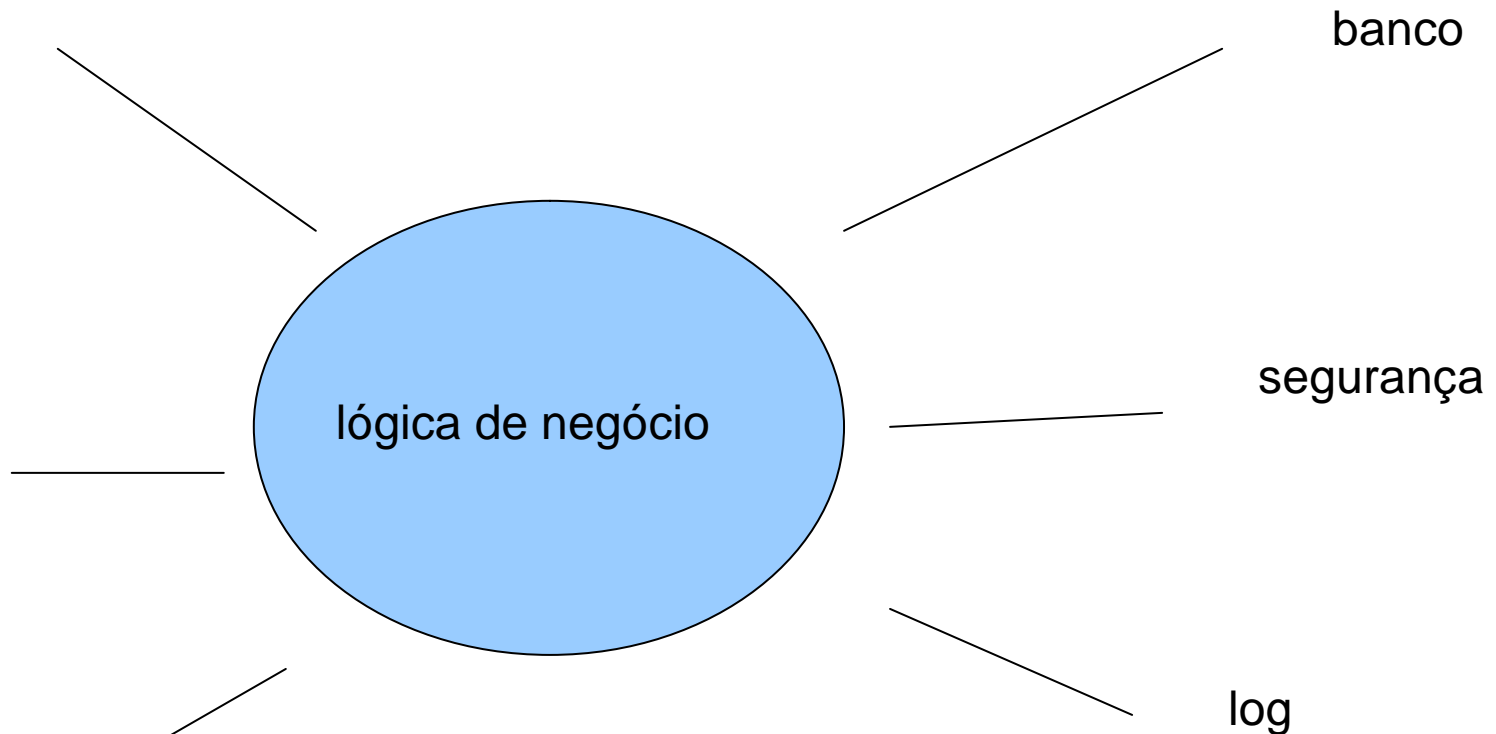
**Você realmente precisa se preocupar com isso?**

# Lógica de negócio

- Parte crucial da aplicação
- Não envolve **aspectos** gerais da aplicação, como segurança, logging, etc.

```
if(usuario.isAdmin()) {  
    logger.log("apagando usuario");  
    userDao.delete(usuario);  
}
```

# Você tem de se importar com:



Sua lógica de negócio ou aplicação está fazendo muita coisa!

## *Development AntiPattern:*

### **The Blob**

- **Symptoms**

- Single class with many attributes & operations
- Controller class with simple, data-object classes.
- Lack of OO design.
- A migrated legacy design

- **Consequences**

- Lost OO advantage
- Too complex to reuse or test.
- Expensive to load



**MITRE**

## *Development AntiPattern:* **Poltergeists**

- Proliferation of classes [Riel 96]
- Spurious classes and associations
  - Stateless, short-lifecycle classes
  - Classes with few responsibilities
  - Transient associations
- Excessive complexity
- Unstable analysis and design models
- Analysis paralysis
- Divergent design and implementation
- Poor system performance
- Lack of system extensibility



# Separation of Concerns

# Preocupações da aplicação

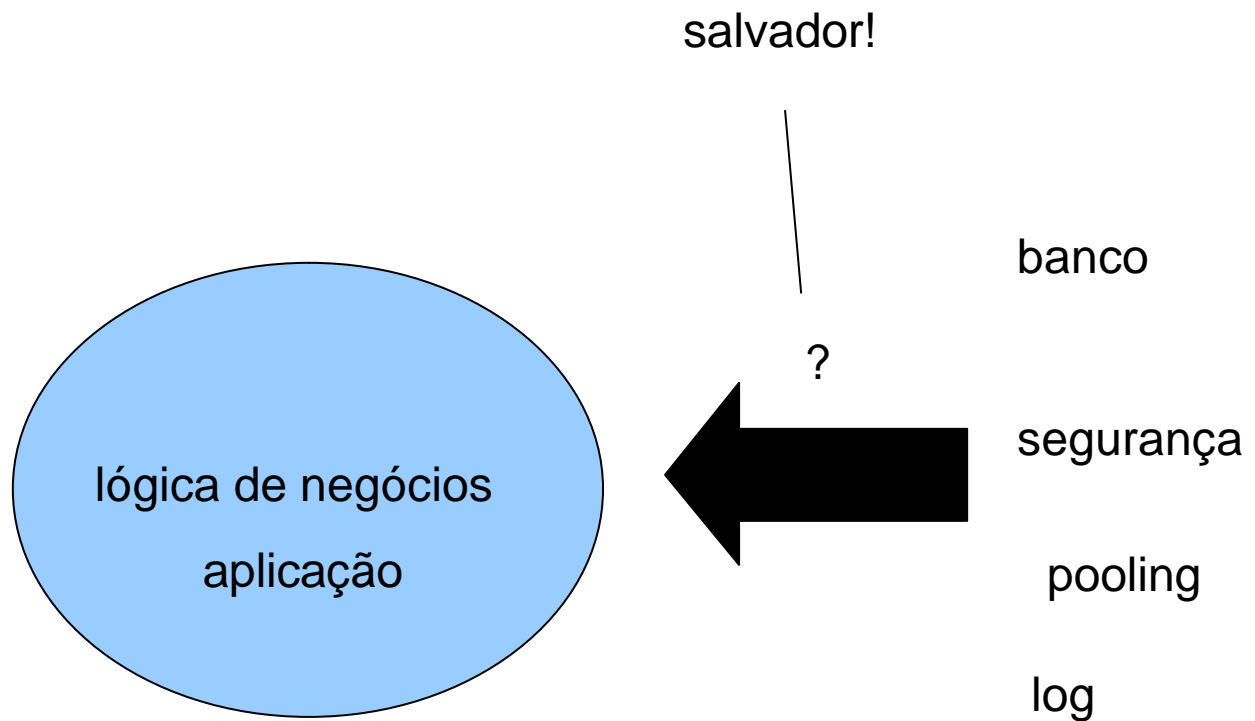
- Você não tem tempo de escrever um pool super eficiente
- Você não quer perder tempo fazendo milhares de IFs de segurança repetidos
- Você não quer ter tempo de se preocupar com concorrência ou transações!
- Socorro! Não quero mais SQL!



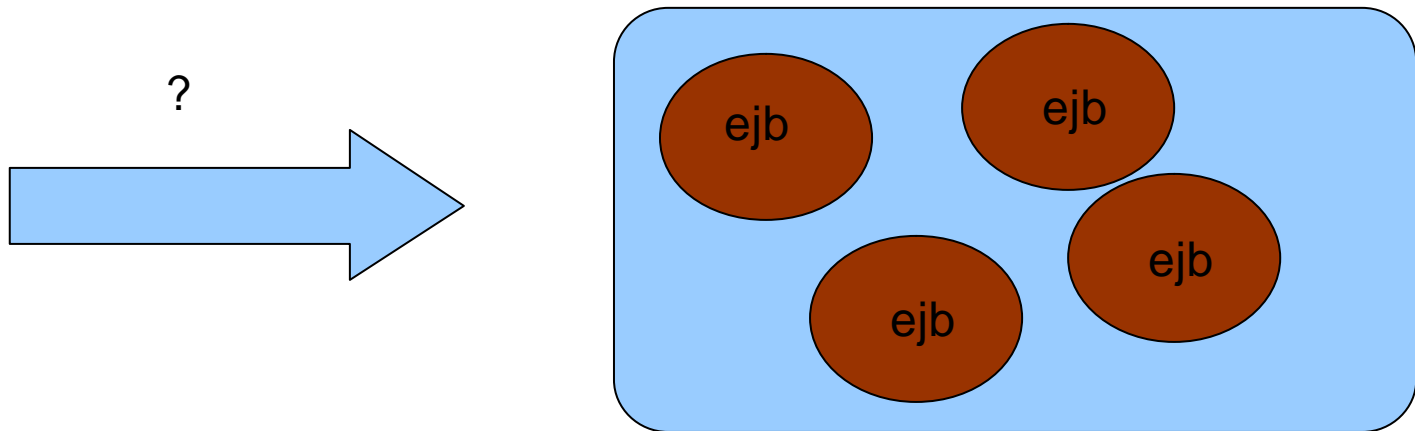
# Carrinho de compras

- A Amazon precisa de um sistema que aguarde 1 milhão de carrinhos simultaneamente instanciados.
- Você precisa implementar o site inteiro, e ainda se preocupar com o volume de acesso que ele vai ter!

# Inversão de Controle



- Enterprise Java Beans  
fazem principalmente o papel de  
lógica de negócios e entidades



# Servidor de Aplicação

Uma especificação para a moradia dos EJBs.

Ele que **serve** as necessidades dos EJBs.

Quem quiser, pode implementar um servidor de aplicação (application server)

- JBoss
- WebLogic
- WebSphere
- Jonas

# Carrinho de Compras

Precisamos de um sistema que manipule milhões de carrinhos de compras simultaneamente (amazon.com).

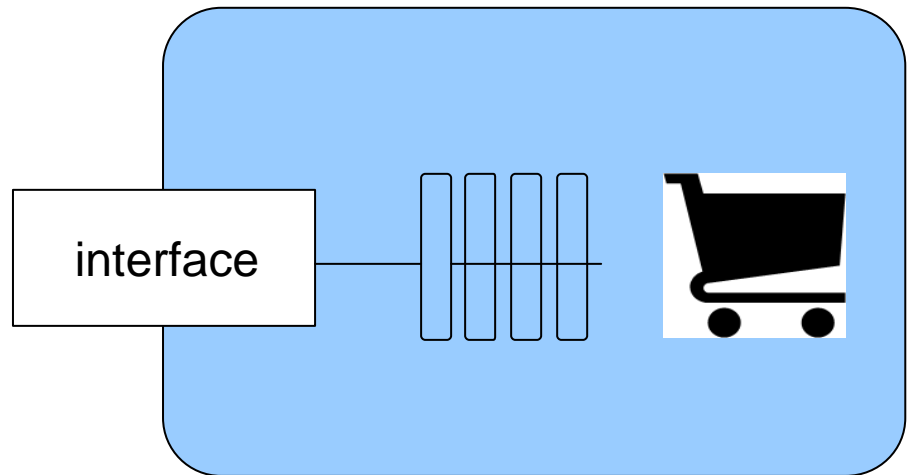
Necessidades:

- Transação
- Acesso multi threaded
- Persistência
- Segurança

# Idéia!

- Conversamos com um objeto de “mentira”.

`carrinho.add(livro);`



O servidor trata a segurança, pooling, log, transação, thread, etc...

# Por enquanto, o que precisamos?

- O Carrinho de compras de mentira
- O Carrinho de compras de verdade (EJB).

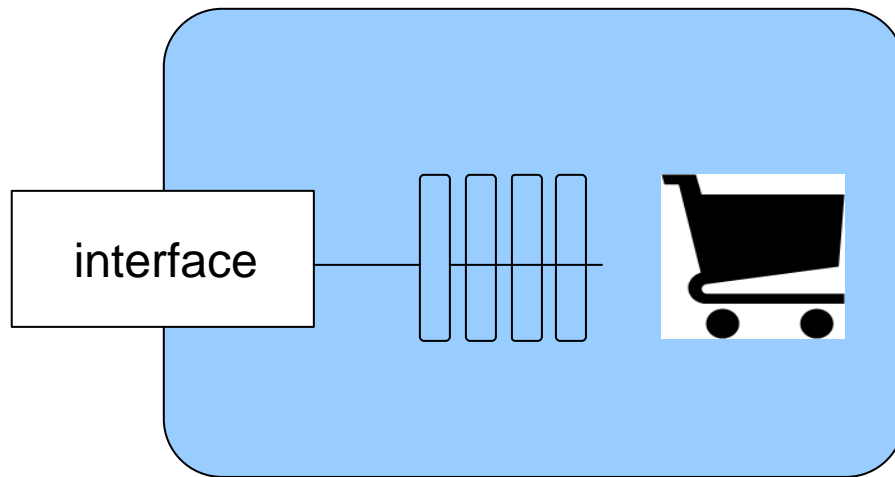
interface Carrinho (o servidor implementa)  
class CarrinhoEJB (com a lógica)

# O EJB (o que está faltando?)

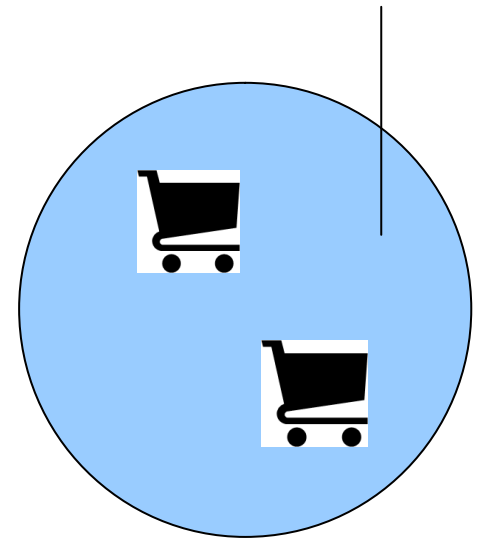
```
class CarrinhoEJB implements SessionBean {  
    List compras;  
    double total;  
  
    public void add(Livro livro) {  
        compras.add(livro);  
        total += livro.getPreco();  
    }  
  
    public double getTotal() {  
        return total;  
    }  
  
    // + um monte de metodos estranhos  
}
```



# É \$caro\$ construir um EJB!



Carrinhos usados!



# O “reciclador” de ejbs!

```
class CarrinhoEJB implements SessionBean {  
    List compras;  
  
    public void ejbCreate() throws CreateException {  
        compras = new ArrayList();  
    }  
  
    public void add(Livro livro) {  
        compras.add(livro);  
    }  
    // + um monte de metodos estranhos e o getTotal()  
}
```

# A interface

```
interface Carrinho extends
    EJBLocalObject {
    public void add (Livro livro);
    public double getTotal();
    // outros metodos que seria interessantes
}
```

# O que mais falta?

Já conseguimos brincar com nosso carrinho.

```
carrinho.addLivro(livro);  
carrinho.getTotal();
```

Como acessar um carrinho, ou criar um novo?

# Idéia: A Casa dos EJBs

Existe um objeto que é responsável por criar, localizar, remover EJBs. É a casa de cada EJB.

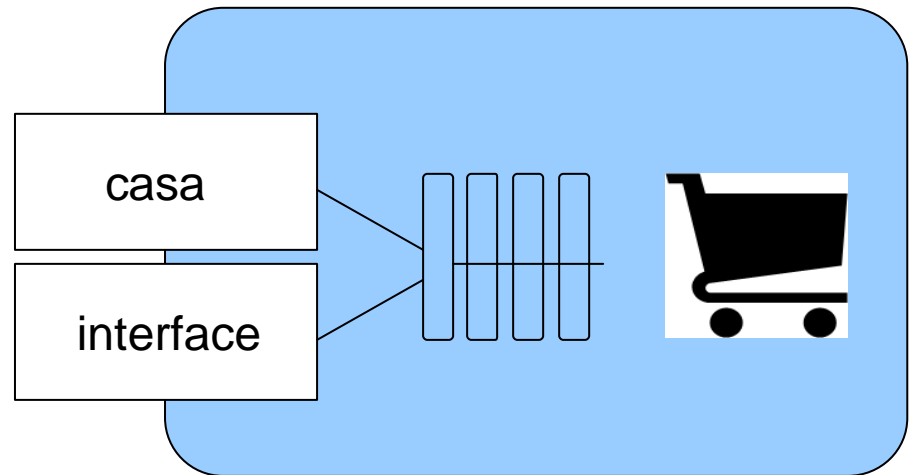
```
CasaDosCarrinhos casa = ....
```

```
Carrinho carrinho = casa.create();
```

```
carrinho.add(livro); ...
```

# A Casa

Acessando a casa dos carrinhos, você não acessa um em específico.  
É o ponto de entrada.



# Código da casa

```
interface CasaDosCarrinhos extends  
    EJBLocalHome {  
    Carrinho create() throws  
        CreateException;  
}
```

A casa é a Home! O nome “correto” dela seria CarrinhoHome

# Outras vantagens

Pra que tudo isso? Porque não instanciar diretamente os CarrinhoEJBs?

- Um milhão de carrinhos de compra
- Todos estão ativos ao mesmo tempo?
- E se alguém deixar o browser aberto?

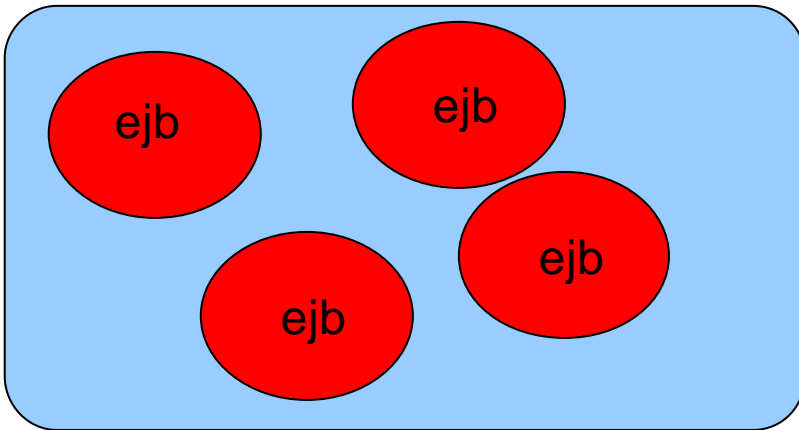
1 milhão de objetos são **muitos** megas!



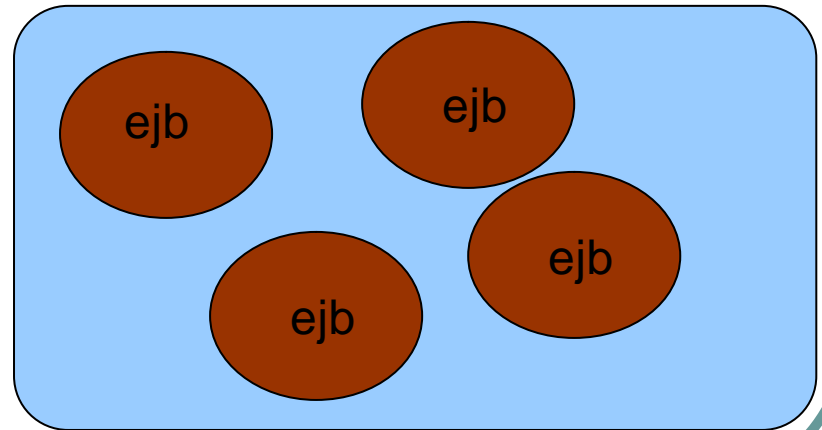
# Passivação

- Apesar de termos 1 milhão de clientes acessando os carrinhos, alguns carrinhos não são tão acessados.

10 mil carrinhos acessados no último minuto

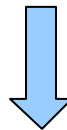
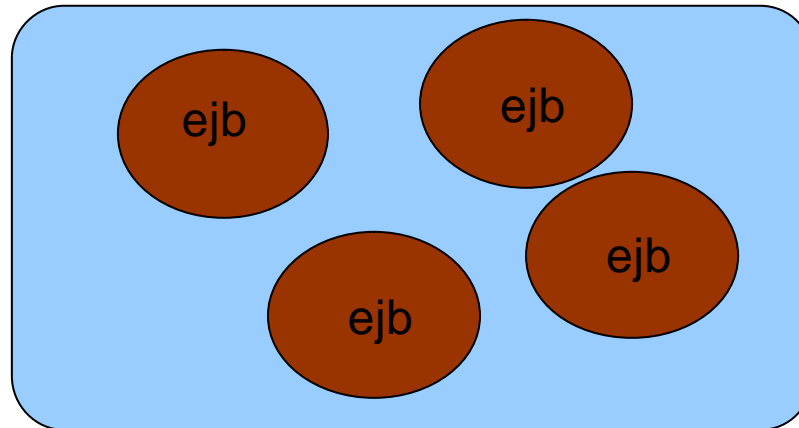


990 mil carrinhos não acessados no último minuto



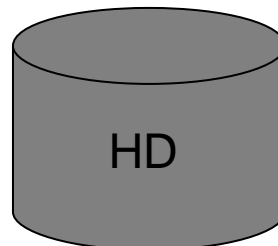
# Passivando!

990 mil carrinhos não acessados  
no último minuto



**Você precisa se  
preocupar com isso?**

ciclo de vida...



# E se...

- precisar que o acesso a um EJB seja transacional?
- precisar logar todo acesso a EJBs?
- precisar definir regras de segurança?
- precisar de acesso remoto?

**Você precisa se  
preocupar com isso?**

**Alguém que me sirva!**

# E o XML?

Para você rodar uma aplicação no seu servidor de aplicação, você precisa de um XML, que fale quem são seus EJBs:

“Caro servidor, meu EJB chamado CarrinhoDeCompras é um EJB, a casa dele se chama CasaDosCarrinhos, e a interface dele para os clientes se chama Carrinho. E quem tem a lógica em si, é o CarrinhoEJB.

Atenciosamente, programador”

# Tipos de EJBs

- SessionBeans

Serviços caros ou muito usados! (exemplos?)

Dois tipos! Stateless e Stateful

- EntityBeans

Representam entidades (veremos).

- MessageDrivenBeans

Consomem mensagens de determinados assuntos.

# Entidades: uma introdução

# Entidades

Implementamos um SessionBean

Gostaríamos agora de não nos preocupar mais com SQLs e persistência.

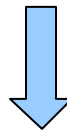
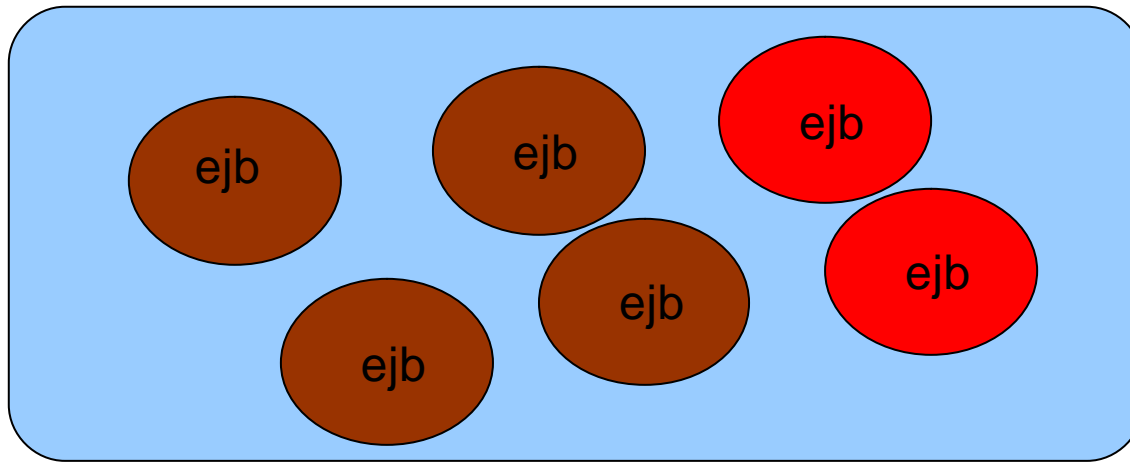
```
compra.add(carrinho);
```

```
compra.save();
```

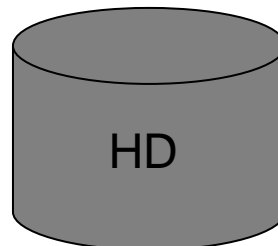
Aqui dentro vai um SQL gigante!

Será que foi um bom momento para gravar a compra?

# Quando persistir?



**Você precisa se  
preocupar com isso?**





# Persistência transparente!

```
CasaDosUsuarios casa = ...
```

```
Usuario user = casa.create("paulo");
```

```
user.setSenha("xpto");
```

```
user.setEndereco("lins de vasconcelos");
```

// nao existe "save"!!! Algum dia, será persistido!  
Quem tem de se preocupar com isso **não** é  
você!

Container Managed Persistence (CMP)

# Container Managed Relationship

```
compra.add(livro);
```

Livro também é uma entidade!

Quem deve se preocupar com a geração de primary keys e foreign keys?

# Aspect Oriented Programming

# Programação orientada a Aspecto

- Hype do momento!
- Servidores de aplicação usam programação orientada a aspecto. Porque?

# Um aspecto: segurança

Ao adicionar um usuário:

```
if(!usuario.isAdmin()) {  
    throw new SecurityException();  
}
```

Ao remover um usuário:

```
if(!usuario.isAdmin()) {  
    throw new SecurityException();  
}
```

- Ao visualizar encomendas

```
if(!usuario.isAdmin()) {  
    throw new SecurityException();  
}
```
- Visualizar encomendas e editar usuários tem alguma relação em comum?

Apenas o **ASPECTO** de segurança

# Idéia!

De alguma maneira, codificar:

“sempre que os métodos X e Y da classe Usuario e os métodos Z e W da classe Encomenda forem acessados, rode:”

```
if(!usuario.isAdmin()) {  
    throw new SecurityException();  
}
```

# Linguagens e Frameworks

- AspectJ: A Xerox começou seu desenvolvimento, e hoje em dia está sobre o projeto Eclipse.
- AspectWerkz: um sueco que resolveu criar um framework de “aspectagem”. O brasileiro Carlos Villela colabora com o projeto.



# Próximos passos!

Agora estão prontos para ler um péssimo tutorial de EJB!!!

[http://java.sun.com/j2ee/tutorial/1\\_3-fcs/index.html](http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html)

Ou procurem por “j2ee tutorial” no search do java.sun.com

# Próximos passos!

Baixem o jboss 3.2.1

<http://www.jboss.org>

Para CMP e CMR, procurem também nos tutoriais da Sun.

# Dificuldades que irão encontrar

- Escrevendo XML do ejb-jar  
Não usem ferramentas na 1a vez!
- Empacotando um jar de ejb
- Fazendo o deploy
- Rodando o cliente

# Obrigado! Perguntas e Respostas

Paulo Silveira [www.guj.com.br](http://www.guj.com.br)

