Introdução a JPA e EJB 3.0

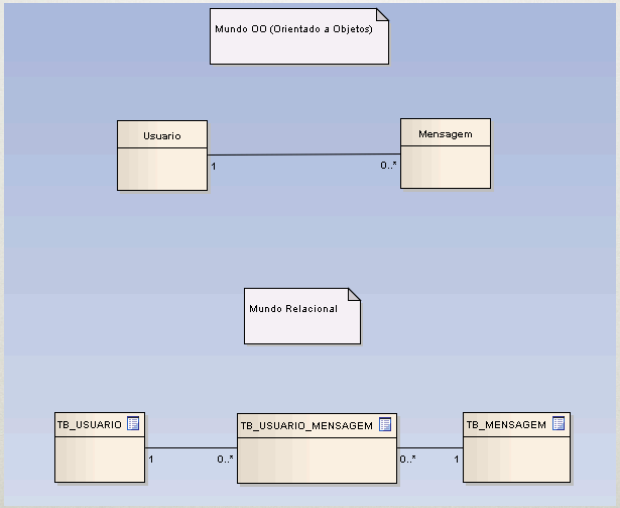
GUSTAVO WAGNER - GUGAWAG@GMAIL.COM

Histórico

- JDBC (veja exemplo no próximo slide)
- ORM (Mapeamento Objeto Relacional. Ver exemplo em dois slides)
- Hibernate, TopLink
- JPA V1

```
        Connection con = null;

        try
        {
            // Este é um dos meios para registrar um driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").getInstance();

            // Registrado o driver, vamos estabelecer uma conexão
            con = DriverManager.getConnection("jdbc:odbc:meusCdsDb","conta","senha");

            // Após estabelecermos a conexão com o banco de dados
            // Utilizamos o método createStatement de con para criar o Statement
            Statement stm = con.createStatement();

            // Vamos executar o seguinte comando SQL :
            String SQL = "Select titulo, autor, total_faixas from MeusCDs";

            // Definido o Statement, executamos a query no banco de dados
            ResultSet rs = stm.executeQuery(SQL);

            // O método next() informa se houve resultados e posiciona o cursor do banco
            // na próxima linha disponível para recuperação
            // Como esperamos várias linhas utilizamos um laço para recuperar os dados
            while(rs.next())
            {

               // Os métodos getXXX recuperam os dados de acordo com o tipo SQL do dado:
               String tit = rs.getString("titulo");
               String aut = rs.getString("autor");
               int totalFaixas = rs.getInt("total_faixas");

               // As variáveis tit, aut e totalFaixas contém os valores retornados
               // pela query. Vamos imprimí-los

               System.out.println(48:"Titulo: "+tit+" Autor: "+aut+"49:                Tot. Faixas: "+totalFaixas);
            }

        }
        catch(SQLException e)
        {
            // se houve algum erro, uma exceção é gerada para informar o erro
            e.printStackTrace(); //vejamos que erro foi gerado e quem o gerou
        }
        finally
        {
           try
           {
              con.close();
           }
           catch(SQLException onConClose)
           {
               System.out.println("Houve erro no fechamento da conexão");
               onConClose.printStackTrace();
           }
        } // fim do bloco try-catch-finally
```

Exemplo ORM


Mundo OO (Orientado a Objetos)
Usuario
Mensagem
1
0..*
Mundo Relacional
TB_USUARIO
TB_USUARIO_MENSAGEM
TB_MENSAGEM
1
0..*
0..*
1

Escolhendo modelo de programação

Modelos

- Para programar em Hibernate é necessário escolher um modelo de programação pretendido;
- A imagem abaixo mostra diferentes ferramentas a serem usadas dependendo do modelo escolhido;

Modelos Mais Usados

- **Top down:** A partir de um modelo e de classes desenvolvidas em Java, gera-se o esquema do banco: usa-se a ferramenta *hbm2ddl*
- **Bottom up:** A partir de um modelo de dados e esquema de banco de dados pré-existente, deve-se usar engenharia reversa para se ter o esquema em hibernate xml ou classes Java anotadas. Usa-se, para esse último caso, hbm2java
- etc

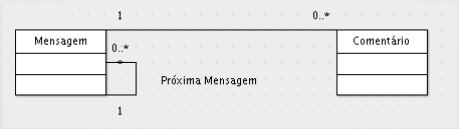
Hello World com JPA

- Para desenvolver um sistem em JPA é necessário:
  - Mapear classes de negócio em entidades de um banco (iremos usar o modelo top down);
  - Definir uma unidade de persistência;
  - Definir um *managed datasource* (conexão com um banco de dados);

Mapeando Classes de Negócio

- Há várias formas de mapear classes de negócio a entidades de um banco de dados;
- É possível fazer com xml;
- Porém, é mais prático usar anotações;

Exemplo em JPA



Mapeando Classes de Negócio

```
package hello.modelo;
...
public class Mensagem implements Serializable{
    private Long id;
    private String texto;
    private Mensagem proximaMensagem;
    private List<Comentario> comentarios;
    public Mensagem() {
      this(null);
    }
    public Mensagem(String texto) {
        this.texto = texto;
        comentarios = new ArrayList<Comentario>();
    }
    public void acrescentaComentario(Comentario comentario){
      this.comentarios.add(comentario);
    }
      public Long getId() {
        return id;
    }
    private void setId(Long id) {
        this.id = id;
    }
    public String getTexto() {
        return texto;
    }
    public void setTexto(String texto) {
        this.texto = texto;
    }
     public Mensagem getProximaMensagem() {
         return proximaMensagem;
     }
     public void setProximaMensagem(Mensagem proximaMensagem) {
         this.proximaMensagem = proximaMensagem;
     }
      public List<Comentario> getComentarios() {
            return comentarios;
      }
      ...
```

Toda classe do modelo que será armazenada no banco deve ser um Pojo: JavaBean, com métodos gets e sets para atributos e construtor sem argumentos. Isso porque Hibernate faz reflexão de código.

A classe Message ao lado tem um tributo id, que representa a chave primária no banco. Se houver dois ou mais objetos com o mesmo id, esses objetos representam a mesma tupla (linha) no banco de dados.

Mapeando Classes de Negócio

```java
package hello.modelo;
...
@Entity
public class Mensagem implements Serializable{
    @Id @GeneratedValue
    private Long id;
    private String texto;
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "NEXT_MESSAGE_ID")
    private Mensagem proximaMensagem;

    @OneToMany(cascade= CascadeType.ALL, fetch = FetchType.EAGER)
    private List<Comentario> comentarios;
    public Mensagem() {
     this(null);
    }
    public Mensagem(String texto) {
        this.texto = texto;
        comentarios = new ArrayList<Comentario>();
    }
    ...
}
```

Classe com anotações JPA.

@Entity: indica que essa classe será persistida (armazenada) no banco de dados.

@Id: usada para indicar o atributo que será a chave primária

@GeneratedValue: usada para gerar automaticamente um novo valor de chave primária

Definindo Unidade de Persistência

- Uma unidade de persistência é uma referência a uma unidade de trabalho com um banco;
- Ela é definida no arquivo src/META-INF/persistence.xml
- Lá é definido se será usado hibernate, toplink ou outra implementação JPA, conexão com banco de dados, etc

Exemplo de persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="1.0" 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_1_0.xsd">

<persistence-unit name="HelloWorldJPA">

<jta-data-source>java:/DefaultDS</jta-data-source>

  <properties>
      <property name="hibernate.archive.autodetection"
          value="class, hbm"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.connection.driver_class"
                value="org.hsqldb.jdbcDriver"/>
      <property name="hibernate.connection.url"
                value="jdbc:hsqldb:hsql://localhost"/>
      <property name="hibernate.connection.username"
                value="sa"/>
      <property name="hibernate.dialect"
                value="org.hibernate.dialect.HSQLDialect"/>
      <property name="hibernate.hbm2ddl.auto" value="create"/>
  </properties>

</persistence-unit>


</persistence>
```

- Define o nome da unidade de persistência.
- Define qual é o datasource (configurações sobre o banco de dados) da aplicação. Daremos mais detalhes sobre datasource mais à frente.
- Define a forma como o hibernate vai mapear as classes em entidades no banco. **CLASS** diz ao hibernate para procurar anotações nas classes. **HBM** diz ao hibernate que você definirá num xml quais são as classes que serão entidades no banco.
- Define qual será o dialeto que o Hibernate usará. Nesse caso, HSQLDialect, significando que hibernate gerará código SQL para HSQL. Se o banco for MySQL, por exemplo, altera-se esse dialeto.
- Gera automaticamente o esquema no banco de dados.

Definindo um Managed Datasource

- Finalmente precisamos definir um datasource
- Um datasource representa uma configuração de banco de dados
- Iremos usar o Banco de Dados HSQL e o datasource que já vem previamente configurado no JBoss
- Para ver o datasource do HSQL veja o arquivo <JBoss>/server/default/deploy/hsqldb-ds.xml
- Perceba que uma das configurações desse arquivo é o jndi-name:
- `<jndi-name>DefaultDS</jndi-name>`
- Outros arquivos de configuração de datasources (para diferentes banco de daos) podem ser vistos em <jboss>/docs/examples/jca

Usando o Hibernate

- Uma vez anotadas as classes, definir uma unidade de persitência e um datasource, resta apenas usar o hibernate
- Veja um exemplo de uso no próximo slide

Exemplo de uso de hibernate

```
package hello;

import java.util.*;

import org.hibernate.*;
import persistence.*;

public class HelloWorld {

    public static void main(String[] args) {

        // First unit of work
        Session session =
            HibernateUtil.getSessionFactory().openSession();
        Transaction tx = session.beginTransaction();

        Message message = new Message("Hello World");
        Long msgId = (Long) session.save(message);

        tx.commit();
        session.close();

        // Second unit of work
        Session newSession =
            HibernateUtil.getSessionFactory().openSession();
        Transaction newTransaction = newSession.beginTransaction();

        List messages =
            newSession.createQuery("from Message m order by
        ➥ m.text asc").list();

        System.out.println( messages.size() +
            " message(s) found:" );
```

Session: representa uma unidade de trabalho com o Banco. Em JPA, EntityManager faz o mesmo papel

Transaction: delimita área de transação do Banco. Com EJB, o próprio container pode gerenciar transações.

Query (consulta): pode ser feita em HQL (do Hibernate) ou SQL padrão.

Exemplo de Uso de Hibernate

- Fácil escrever o código anterior, não?! :)
- Também não gostei! :)
- Que tal deixar alguém cuidar das sessões, transações, entre outras coisas, do banco de dados?
- O Container faz isso! Isso se chama CMP: Container Managed Persistence

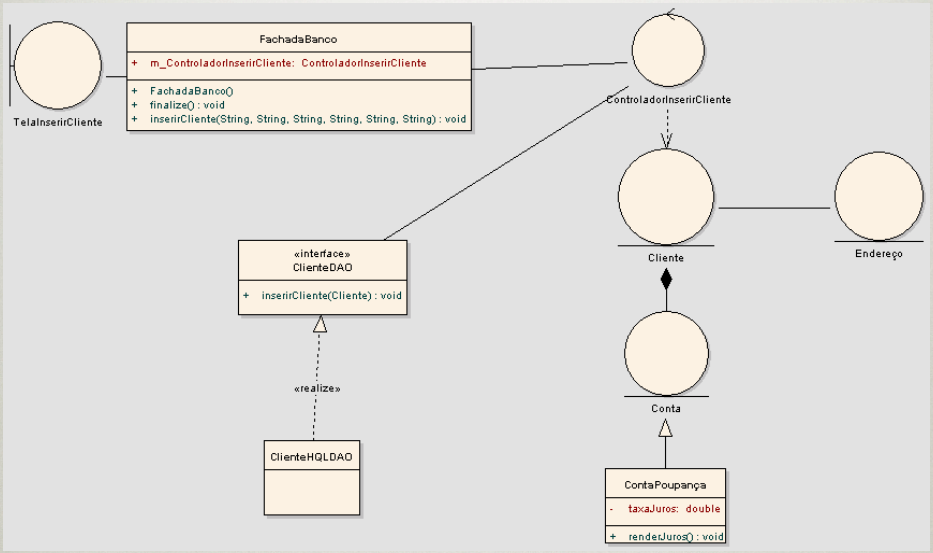
EJB 3.0 - Session Beans

EJB 3.0

- A especificação EJB 3.0 é completamente diferente do EJB 2.1
- Atualmente JPA faz parte da especificação EJB 3.0. Além de Entity, EJB 3.0 tem SessionBean (Stateless e Stateful) e MessageDriven Bean
- Para acessar o EntityManager, gestor das entidades de uma unidade de persistência, é muito mais fácil via EJB 3.0 com Session Bean

SessionBean Stateless

- Um SessionBean é um objeto em Java que pode ser acessado remotamente
- Muito usado para ambiente onde os objetos são distribuídos
- Veja a imagem da próxima imagem para explicarmos onde usaremos SessionBean


TelaInserirCliente
FachadaBanco
+ m_ControladorInserirCliente: ControladorInserirCliente
+ FachadaBanco()
+ finalize() : void
+ inserirCliente(String, String, String, String, String, String) : void
ControladorInserirCliente
«interface»
ClienteDAO
+ inserirCliente(Cliente) : void
«realize»
ClienteHQLDAO
Cliente
Endereço
Conta
ContaPoupança
- taxaJuros: double
+ renderJuros() : void

SessionBean Stateless

- Veja que no exemplo de arquitetura do último slide, a GUI (TelaInserirCliente) se comunica com a FachadaBanco
- Num ambiente distribuído, podemos ter a GUI (em JSF) num servidor e a parte de lógica de negócio (Fachadas, Controladores, Repositórios e Objetos de Negócio) em outro servidor
- Programar isso na mão (usando RMI, etc) seria muito complicado

SessionBean Stateless

- Dessa forma, iremos usar os serviços do Container (JBoss) para que a GUI se comunique com a lógica de negócio
- Para que uma classe seja considerada um SessionBean Stateless, é necessário usar algumas anotações e implementar uma interface Remota ou Local (se os objetos estiverem na mesma JVM)

SessionBean Stateless

```
package hello.dados;

import hello.modelo.Mensagem;

import java.util.List;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless(name="RepositorioMensagens")
public class RepositorioMensagens implements RepositorioMensagensIF{

     @PersistenceContext(unitName="HelloWorldJPA")
     private EntityManager em;

     public List<Mensagem> getMensagens() {
          return (List<Mensagem>) em.createQuery("select m from Mensagem m")
                .getResultList();
     }

     public void insereMensagem(Mensagem mensagem) {
          em.persist(mensagem);
     }

     public Mensagem getMensagemInicio(String inicio) {

          return (Mensagem)em.createQuery("from Mensagem where texto like ':inicio%'").setParameter("inicio",
inicio).getResultList().get(0);
     }

}
```

Ao lado está um Repositório de Mensagens. Perceba que essa classe está anotada com Stateless, indicando que essa classe é um SessionBean.

Além disso, ela implementa RepositorioMensagensIF, uma interface remota.

É aqui que começam as facilidades de EJB3.0. Para solicitar ao container a unidade de persistência HelloWorldJPA definida no arquivo persistence.xml, usa-se a anotação PersistenceContext. Essa anotação injeta na classe uma instância da classe EntityManager de JPA (similar a Session do Hibernate)

Com o EM (EntityManager), faz-se todas as ações possíveis com o banco de dados. O método createQuery possibilita definir uma consulta na linguagem JPA QL, similar a SQL. Mais informações sobre JPA QL: http://www.hibernate.org/hib_docs/entitymanager/reference/en/html/queryhql.html

SessionBean Stateless

- `package hello.dados;`

```
import hello.modelo.Mensagem;

import java.util.List;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless(name="RepositorioMensagens")
public class RepositorioMensagens implements RepositorioMensagensIF{

     @PersistenceContext(unitName="HelloWorldJPA")
     private EntityManager em;

     public List<Mensagem> getMensagens() {
          return (List<Mensagem>) em.createQuery("select m from Mensagem m")
                .getResultList();
     }

     public void insereMensagem(Mensagem mensagem) {
          em.persist(mensagem);
     }

     public Mensagem getMensagemInicio(String inicio) {

          return (Mensagem)em.createQuery("from Mensagem where texto like ':inicio%'").setParameter("inicio",
inicio).getResultList().get(0);
     }

}
```

O método persist de EM salva o objeto no banco. Se o objeto já existir (id existente no banco), o hibernate atualiza a tupla no banco.

Interface Remota de um SessionBean

- `package hello.dados;`

```
import hello.modelo.Mensagem;

import java.util.List;

import javax.ejb.Remote;

@Remote
public interface RepositorioMensagensIF {

    public void insereMensagem(Mensagem mensagem);
    public List<Mensagem> getMensagens();
    public Mensagem getMensagemInicio(String inicio);

}
```

Define uma interface como sendo remota. Uma classe que implementa uma interface remota pode ser acessada por uma instância de JVM diferente da que ela se encontra. Perceba que essa interface é uma interface padrão de Java.

Acessando o Repositório

- Por fim, precisamos acessar o repositório e inserir mensagens no banco
- Para isso, vamos fazer um programa que se comunica com um objeto remoto de RepositorioMensagens

Programa acessando o RepositorioMensagens

```
package hello.gui;

import hello.dados.RepositorioMensagensIF;
import hello.modelo.Comentario;
import hello.modelo.Mensagem;

import java.util.List;

import javax.naming.InitialContext;
import javax.naming.NamingException;

public class AppMensagem {

     public static void main(String[] args) {
          RepositorioMensagensIF repMensagens = null;
         try {
              InitialContext context = new InitialContext();
              repMensagens = (RepositorioMensagensIF) context.lookup( "RepositorioMensagens/remote" );
         } catch (NamingException e) {
             e.printStackTrace();
         }

         Mensagem mensagem1 = new Mensagem("Olá mundo 1");
         Mensagem mensagem2 = new Mensagem("Olá mundo 2");
         mensagem1.setProximaMensagem(mensagem2);
         mensagem1.acrescentaComentario(new Comentario("comentario 1"));
         mensagem1.acrescentaComentario(new Comentario("comentario 2"));
         repMensagens.insereMensagem(mensagem1);
         List<Mensagem> mensagens = repMensagens.getMensagens();
         for (Mensagem mensagemAtual : mensagens) {
              System.out.println(mensagemAtual);
          }
     }
}
```

Para se comunicar com um objeto remoto (o RepMensagens que é SessionBean Stateless), se a classe não tiver no contexto EJB, é necessário solicitar ao contexto o objeto.

InitialContext vai buscar na url especificada no arquivo src/jndi.properties (que pode ser visto no próximo slide) uma referência de RepositorioMensagensIF. Perceba que o nome pedido é RepositorioMensagens, o mesmo definido na classe RepositorioMensagens.java.

Para criar uma Mensagem basta chamar o construtor da mesma. Não existe mágica nem conceitos novos.

Finalmente, basta chamar os métodos do Repositório.

Arquivo jndi.properties

Nesse arquivo é definido como se comunicar remotamente com o serviço de nomes do JBoss. Para alterar a url, basta mudar o ip abaixo

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=127.0.0.1:1099
```

JEE

Benefícios JPA

Benefícios JPA

- Teste de código independente do container
  - Tudo são POJOs
- Mapeamentos via annotations:
  - Não precisa mexer em xml! ☺
- Independente de tecnologias específicas
  - Hibernate
  - Top-link
- Classes do modelo podem ser reusadas fora do contexto de persistência, já que elas são apenas JavaBeans;

Mais Anotações JPA

Anotações Básicas

- Uma lista extensiva de anotações JPA e exemplos pode ser encontrada em:
  - http://www.oracle.com/technology/products/ias/toplink/jpa/resources/toplink-jpa-annotations.html

Anotações

- @Entity
  - Anota a classe como um Entity Bean. É preciso ter um construtor sem argumentos
- @Table(name = "forum")
  - Nome da tabela da entidade
- @PersistenceContext(name="nome_no_persistence.xml") EntityManager em
  - Insere um EntityManager
  - Normalmente usado dentro de um SessionBean
- @Id @GeneratedValue
  - Diz qual atributo é a chave primária.
  - Apenas @Id em cima de um método getxxx indica que a chave primária é xxx

Anotações

- @ManyToOne(cascade=PERSIST)
  ou @OneToMany ou @OneToOne
  @JoinColumn(name="MANAGER_ID",
  referencedColumnName="EMP_ID")
  - Faz ligações entre tabelas

EntityManager

- Um objeto do tipo EntityManager controla o acesso ao banco de dados;
- Por exemplo, EntityManager.persist(object) irá persistir um POJO object no banco;
- Com um EntityManager pode-se também recuperar ou remover um objeto no banco
  - Cliente cliente = em.find(Cliente.class, cpf);
- Pode-se fazer consultas específicas:
  - em.createQuery("SELECT reg.login FROM Servidor reg ORDER BY reg.login").getResultList();

Atividades do Projeto

Projeto Twitter

- Sua missão é mapear as classes do modelo com anotações JPA
- O projeto está no site com a classe Usuário anotada e já pode ser salva em banco
- A Fachada e o ControladorUsuario são EJBs SessionBean Stateless. Você deve alterar o ControladorMensagem para que ele também seja EJB
- Siga o exemplo do que foi feito com Usuário para lhe ajudar com Mensagem