



Open-source Education →

Hibernate: passado, presente e futuro

Agenda

1. Histórico sobre persistência
2. Tecnologias populares
3. Hibernate clássico
4. Java Persistence API
5. Hibernate & JPA
6. Demo: Desenvolvimento Swing com Hibernate / JPA
7. Demo: Desenvolvimento Web com Hibernate e Faces
8. Conclusões

Persistência

- Capacidade de gravarmos dados em memórias não voláteis: hard-disk, tape-backup, CD, DVD, etc.
- Necessidade básica, 99% dos softwares corporativos necessitam;
- “*Quanto mais culpados queremos encontrar, mais dados armazenamos*”;
- Linguagens como Dbase, Clipper, Visual Object, Paradox, FoxPro eram orientadas a dados persistentes;

Agenda

1. Histórico sobre persistência
2. Tecnologias populares
3. Hibernate clássico
4. Java Persistence API
5. Hibernate & JPA
6. Demo: Desenvolvimento Swing com Hibernate / JPA
7. Demo: Desenvolvimento Web com Hibernate e Faces
8. Conclusões

Tecnologias Populares

- “Homens quando eram homens, escreviam seu próprio código SQL” – *especialista em SQL embriagado, lembrando quando ganhava R\$ 80,00 por hora para escrever procedures*
- JDBC com código SQL e Data Access Object Pattern;
- RDO, ADO e DAO na plataforma Microsoft;
- PL/SQL com Oracle Forms;
- “Computadores quando não eram computadores, rodavam código SQL escrito por homens” – *autor deste slide embriagado, tentando fazer uma gracinha para o público*

JDBC

- API do Java para acesso a banco de dados;
- Trabalha com esquema de Driver;
- 4 Tipos de Driver:
 - *1 JDBC / ODBC – Bridge para comunicação com Drivers Windows*
 - *2 Nativo – Código de acesso ao DB é escrito em C / C++;*
 - *3 Middleware Driver – DB é acessado via middleware especializado*
 - *4 Puro Java – Acesso de baixo nível ao DB é totalmente escrito em Java. Driver desejado e utilizado na maior parte.*
- Com o Driver obtemos conexões;
- Com conexões enviamos comandos;
- Comandos podem gerar resultados (select)

DEMO JDBC

Mundo novo

- No mundo novo...
 - *Entity Beans 1.x (padrão Java EE)*
 - *Entity Beans 2.0 e 2.1 (padrão Java EE)*
 - *Hibernate*
 - *Oracle TopLink*
 - *Java Data Objects*
 - *Prevailer*
- Grande parte das abordagens trocam SQL por XML meta-dado;

Alternativas..

- Em algumas implementações, o XML ficou excessivo (Entity 2.x);
- Alternativas para o padrão Entity Bean:
 - Spring;
 - Pico Container;
 - OJB;
 - Hibernate stand-alone;
 - Hiberante com EJBs Session;
 - Outro framework com Web ou EJB;

Agenda

1. Histórico sobre persistência
2. Tecnologias populares
3. Hibernate clássico
4. Java Persistence API
5. Hibernate & JPA
6. Demo: Desenvolvimento Swing com Hibernate / JPA
7. Demo: Desenvolvimento Web com Hibernate e Faces
8. Conclusões

Porque utilizar?

- **Custo:** é opensource LGPL;
- **Benefício:** é uma solução poderosa, madura e portátil compatível com diversos bancos de dados relacionais e servidores de aplicação JEE;
- **Curva de aprendizado:** é rápida comparada com as outras soluções;
- **Documentação:** livros publicados e diversos tutoriais e artigos disponíveis na internet;

Porque utilizar?

- **Suporte:** pode ser contratado comercialmente ou pode se recorrer a uma comunidade extremamente ativa nos fóruns de discussão;
- **Padrão “De Facto”** : amplamente adotado pelo mercado superando as especificações EJB 2.x e JDO;
- Os conceitos do projeto Hibernate foram adotados para os entity beans segundo a especificação EJB 3;
- **Número de profissionais;**

Exemplo

- Um **JavaBean Cliente**, contendo os atributos:
codigo, nome, endereco, telefones
- Uma tabela no RDBMS:
codigo (auto-increment) int
nome (varchar 255)
endereco (varchar 255)
telefones (varchar 255)

XML's

- Um hibernate-config.xml configurando o JDBC e dialeto:

```
<hibernate-configuration>
  <session-factory>
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="show_sql">true</property>
    <property name="connection.driver_class">org.gjt.mm.mysql.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost/teleentrega</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <mapping resource="br/org/soujava/teleentrega/persistence/Cliente.hbm.xml" />
    <mapping resource="br/org/soujava/teleentrega/persistence/Pedido.hbm.xml" />
    <mapping resource="br/org/soujava/teleentrega/persistence/Item.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

- Devemos criar um XML por entidade para mapeamento;

XML's

- Um xml para cada definição de entidade, neste caso chamamos o arquivo abaixo de Cliente.hbm.xml:

```
<hibernate-mapping>
  <class table="clientes" name="br.org.soujava.teleentrega.model.Cliente">
    <id column="codigo" name="codigo">
      <generator class="native"/>
    </id>
    <property name="nome" type="string" column="nome"/>
    <property name="endereco" type="string" column="endereco"/>
    <property name="telefones" type="string" column="telefones"/>
  </class>
</hibernate-mapping>
```

Código Java – Ler Clientes

```
public void exemploListar() throws Exception{
    URL url = Manager.class.getResource("hibernate-config.xml");
    Configuration cfg = new Configuration();
    cfg.configure(url);
    SessionFactory sf = cfg.buildSessionFactory();
    Session session = sf.openSession();
    String hql1 = "from Cliente";
    Query q1 = session.createQuery(hql1);
    Collection<Cliente> clientes = (Collection<Cliente>) q1.list();
    for(Cliente cliente : clientes) {
        System.out.println("Cliente: " + cliente.getNome());
    }
}
```

Salvar e Remover

```
public void save(Object objeto) {  
    Transaction t = session.beginTransaction();  
    session.saveOrUpdate(objeto);  
    t.commit();  
  
}  
public void delete(Object objeto) {  
    Transaction t = session.beginTransaction();  
    session.delete(objeto);  
    t.commit();  
  
}
```

Relacionamentos

- Podemos definir as associações / composições dos nossos objetos. Vejamos um exemplo para a classe

```
<hibernate-mapping>
  <class table="pedidos" name="br.org.soujava.teleentrega.model.Pedido">
    <id column="codigo" name="codigo">
      <generator class="native"/>
    </id>
    <property name="dataPedido" type="date" column="data_pedido"/>
    <property name="observacoes" type="string" column="observacoes"/>
    <list cascade="all" name="itens">
      <key column="codigo_pedido"/>
      <index column="index"/>
      <one-to-many class="br.org.soujava.teleentrega.model.Item"/>
    </list>
    <many-to-one not-null="false" column="codigo_cliente" name="cliente"/>
  </class>
</hibernate-mapping>
```

Relacionamentos

- Neste caso estamos definindo que um Pedido contém uma coleção de itens (um-para-muitos):

```
<list cascade="all" name="itens">  
  <key column="codigo_pedido"/>  
  <index column="index"/>  
  <one-to-many class="br.org.soujava.teleentrega.model.Item"/>  
</list>  
<many-to-one not-null="false" column="codigo_cliente" name="cliente"/>
```

- Um Pedido tem um Cliente (muitos-para-um)

A classe Pedido:

```
public class Pedido {  
    private int codigo;  
    private Cliente cliente;  
    private java.util.Date dataPedido;  
    private String observacoes;  
    private Collection<Item> itens = new ArrayList();  
}
```

- Totalmente simples, com atributos encapsulados por getters e setters!

Resumo Hibernate Clássico

- Conquistou uma grande comunidade devido a sua simplicidade aliada a competência;
- Tem capacidades para gerenciamento de estratégia de recuperação de objetos compostos;
- Aumenta a produtividade;
- Em muitos casos o código SQL gerado pelo Hibernate é superior ao código humano;

O fenômeno Xdoclet

- Diversas tecnologias passaram a utilizar documentos XML como parte da implementação e configuração de um framework
- Exemplos: Struts, Hibernate, EJBs, Java Web Components, Log4J, Tiles, etc.
- Resultado: um emaranhado de XMLs para gerenciar;
- Solução = usar comentários formato JavaDoc para inserir meta-dados e configurações no código Java.

O fenômeno Xdoclet

O Xdoclet estende o JavaDoc, criando anotações no estilo @XPTO para gerar arquivos XML para os frameworks.

```
/**  
 * @hibernate.class table="clientes"  
 */
```

```
public class Cliente {  
    private int codigo;  
    private String nome="";  
    private String endereco="";  
    private String telefones="";
```

```
/**...*/
```

```
public Cliente() {...}
```

```
/**  
 * @hibernate.id column="codigo"  
 *                 generator-class="identity"  
 */
```

```
public int getCodigo() {...}
```

O fenômeno Xdoclet

- Xdoclet foi amplamente utilizado para EJBs Entity e Session, uma vez que seu primeiro objetivo foi simplificar tais tecnologias;
- Xdoclet influenciou muito no desenvolvimento Java Enterprise;
- Trabalha integrado ao Ant;

O fenômeno Xdoclet

- No Java 1.5 as anotações são nativas, ou seja, não precisamos colocar em comentários. Ex.:

```
public class Clientes implements Serializable {  
  
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)  
    @Column(name = "codigo", nullable = false)  
    private Integer codigo;  
  
    @Column(name = "nome")  
    private String nome;  
  
    @Column(name = "telefones")  
    private String telefones;  
  
    @Column(name = "endereco")  
    private String endereco;  
}
```

Agenda

1. Histórico sobre persistência
2. Tecnologias populares
3. Hibernate clássico
4. Java Persistence API
5. Hibernate & JPA
6. Demo: Desenvolvimento Swing com Hibernate / JPA
7. Demo: Desenvolvimento Web com Hibernate e Faces
8. Conclusões

Persistence API

- Parte resultante da especificação de EJB 3.0, alguns dos objetivos do EJB 3.0, relevantes para persistência:
 - Definição de meta-dados (annotations do Java 5)
 - Definição de valores defaults programáticos e de meta-dados a fim de reduzir a necessidade de o desenvolvedor declarar comportamentos comuns e esperados.
 - Simplificação para persistência via entity beans. Suporte para modelos de domínio 'leves', com herança e polimorfismo.
 - Eliminação completa das interfaces para entidades persistentes
 - Especificação de meta-dados e elementos de deployment descriptor para mapeamento objeto relacional

Persistence API

- Padronização do mecanismo de persistência mais adotado no mercado;
- Hibernate é uma implementação JPA;
- TopLink é uma implementação JPA;
- Podemos utilizar o JPA sem a necessidade de um container de EJBs (lighweight container);
- Entidades podem ser injetadas em EJBs ao invés de lookups JNDI;
- As capacidades podem ser ampliadas através de anotações específicas;

Agenda

1. Histórico sobre persistência
2. Tecnologias populares
3. Hibernate clássico
4. Java Persistence API
- 5. Hibernate & JPA**
6. Demo: Desenvolvimento Swing com Hibernate / JPA
7. Demo: Desenvolvimento Web com Hibernate e Faces
8. Conclusões

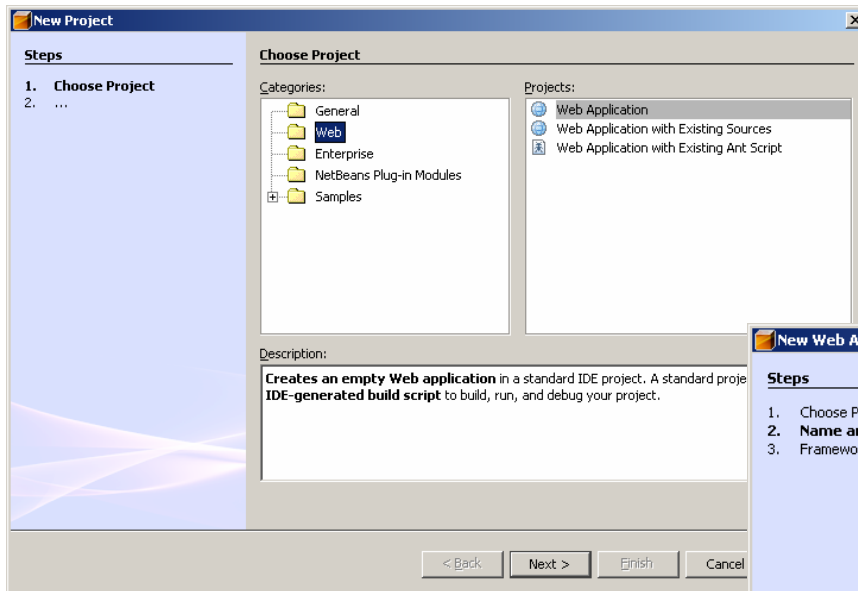
Hibernate e JPA

- Downloads:
 - Hibernate 3.2 Core
 - Hibernate 3.2 Persistence Manager
- Configurar uma biblioteca com os Jars do lib de ambos diretórios;
- O NetBeans 5.5 oferece suporte para JPA e vem pré-configurado com Oracle TopLink;
- Pode-se utilizar com Swing, Web Applications e EJBs;

Agenda

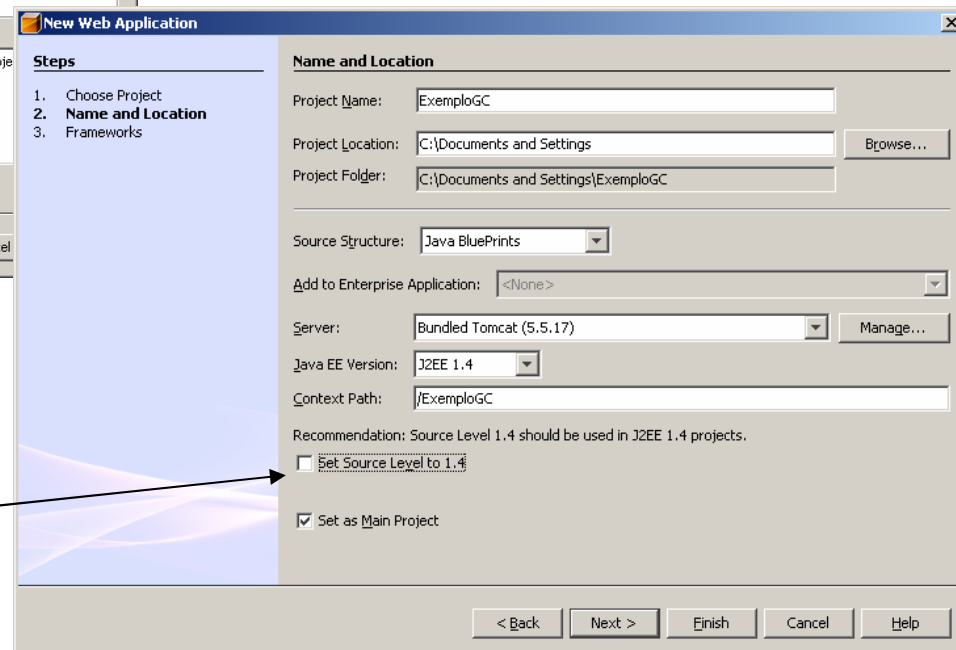
1. Histórico sobre persistência
2. Tecnologias populares
3. Hibernate clássico
4. Java Persistence API
5. Hibernate & JPA
6. Demo: Desenvolvimento Swing com Hibernate / JPA
7. Demo: Desenvolvimento Web com Hibernate e Faces
8. Conclusões

Demo: Netbeans, JPA com Hibernate e Faces

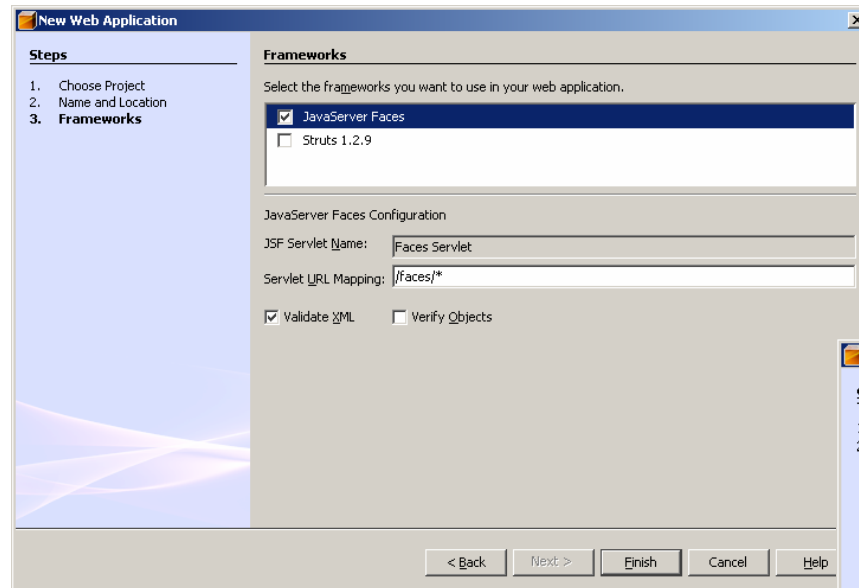


1. Crie um novo projeto Web

2. Digite o nome do projeto e desabilite o código 1.4

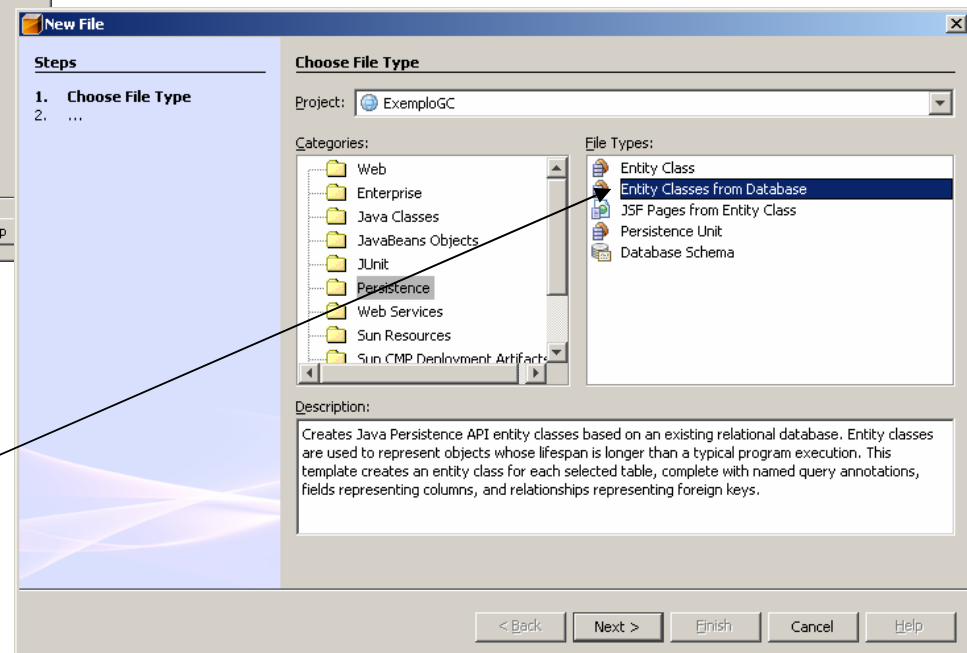


Demo: Netbeans, JPA com Hibernate e Faces

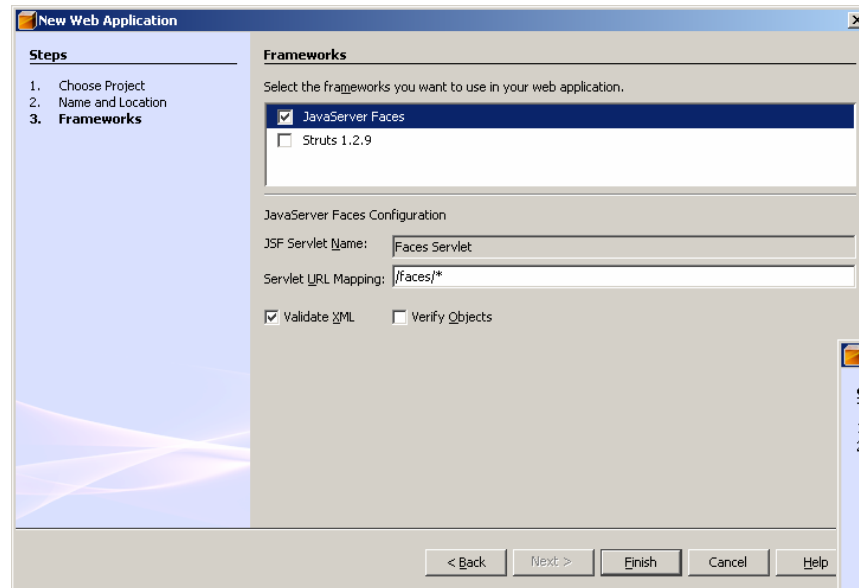


3. Escolha o framework JSF

4. Em File -> New File, escolha Persistence -> Entity from DB

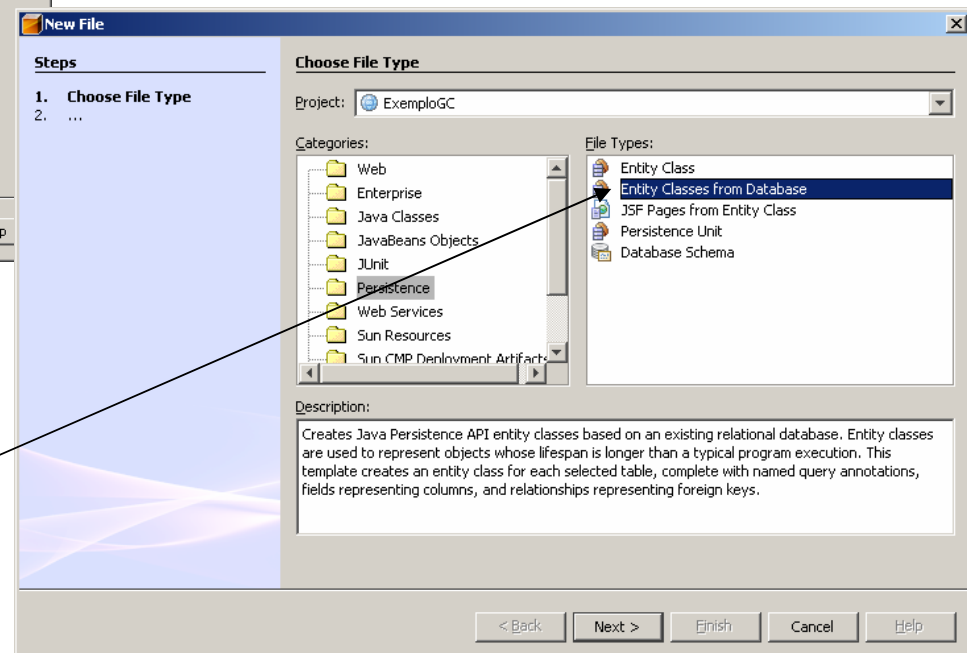


Demo: Netbeans, JPA com Hibernate e Faces

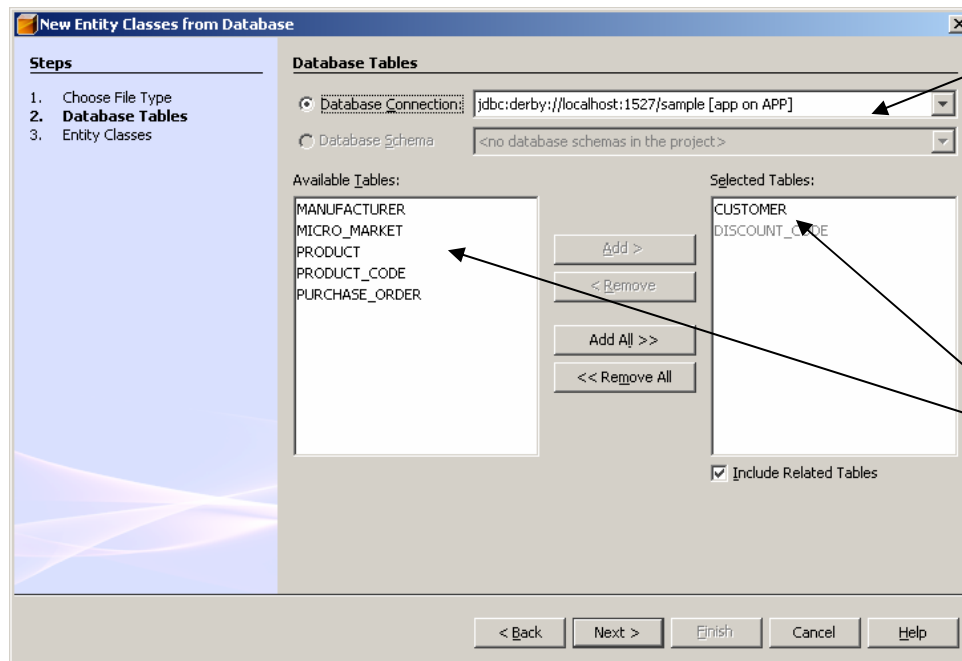


3. Escolha o framework JSF

4. Em File -> New File, escolha Persistence -> Entity from DB



Demo: Netbeans, JPA com Hibernate e Faces

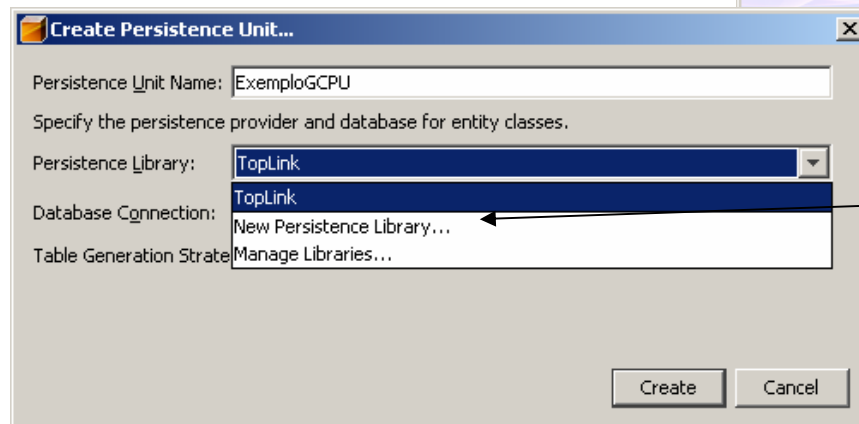
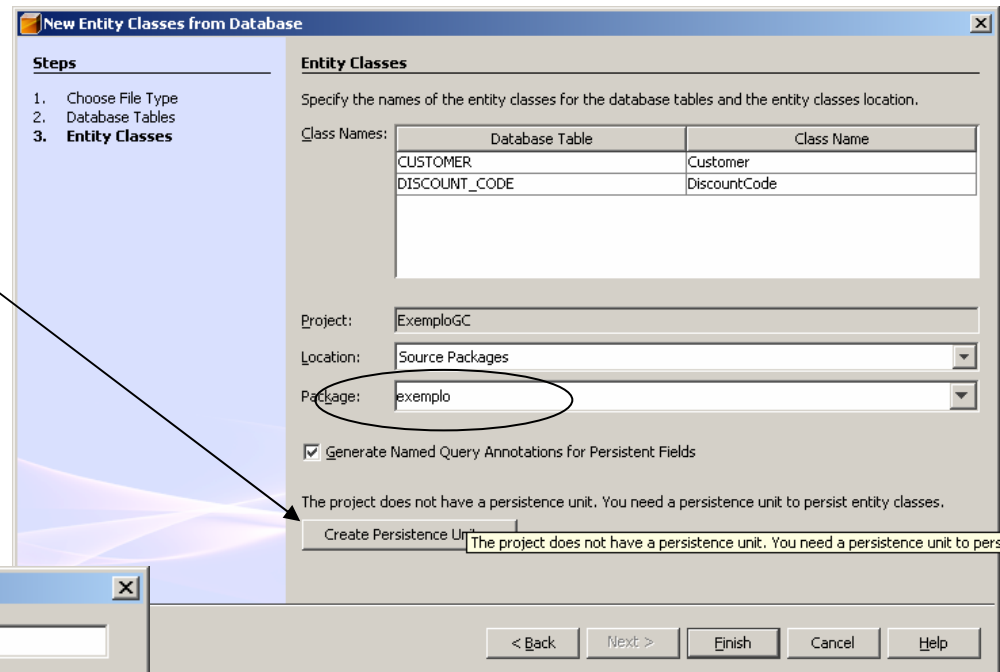


5. Escolha uma conexão JDBC que você tenha configurado na janela Runtime, ou escolha a conexão pré-configurada para o Java DB do NetBeans.

6. Selecione a(s) tabela(s) que deseja reverter em classes Entity.

Demo: Netbeans, JPA com Hibernate e Faces

7. Coloque o nome do pacote das classes e em seguida clique em “Create Persistence Unit”



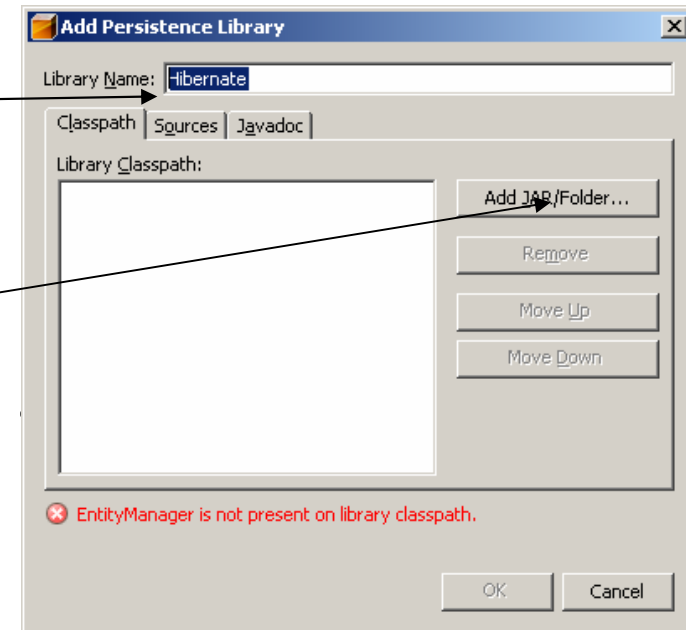
8. Para trabalhar com Hibernate clique em New Persistence Library.

Demo: Netbeans, JPA com Hibernate e Faces

9. Digite Hibernate no nome da biblioteca e adicione todos os seguintes Jars:

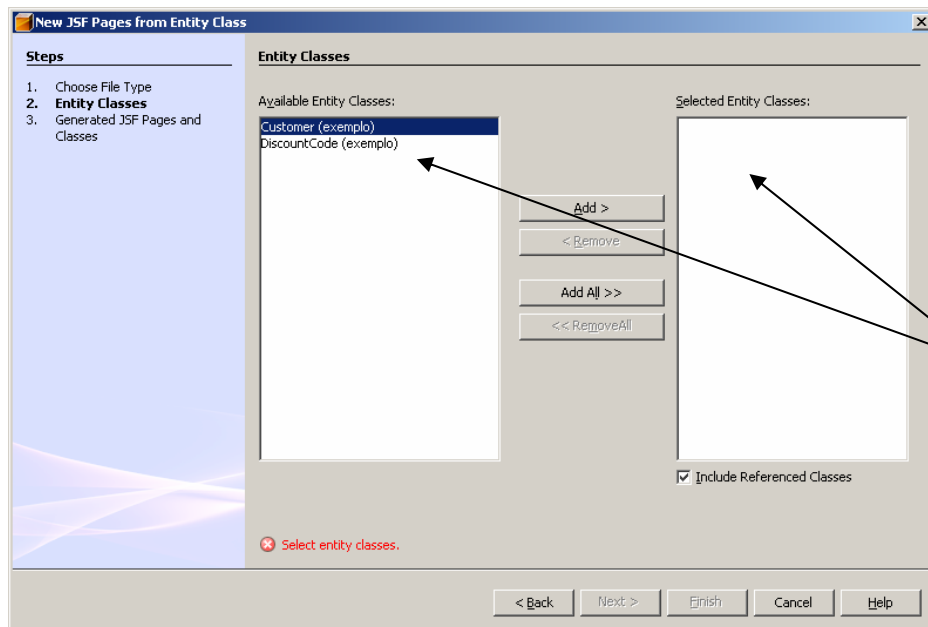
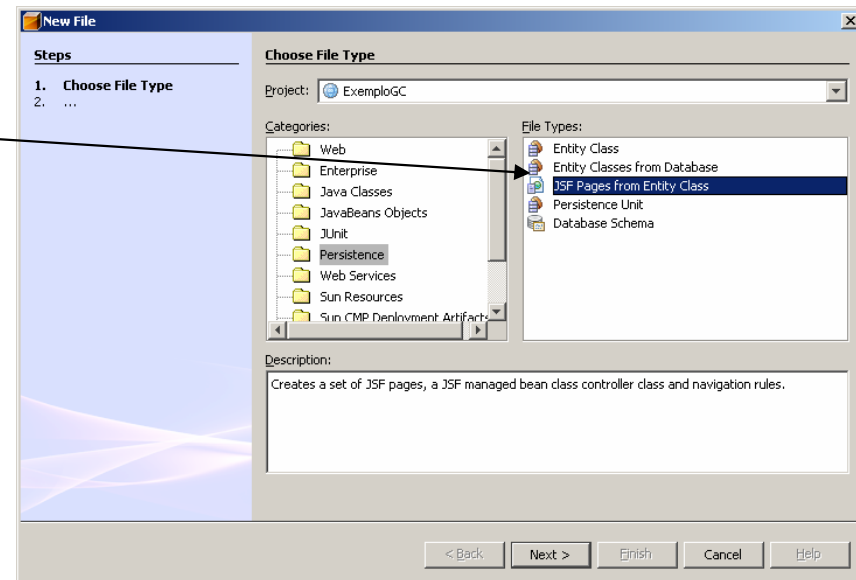
- hibernate3.jar
- jars do hibernate/lib
- hibernate-entitymanager.jar
- jars do hibernate-entitymanager/lib

Clicando em OK e finish o NetBeans vai gerar as entidades / entity classes. Basta agora nos próximos passos, gerar as páginas JSF.



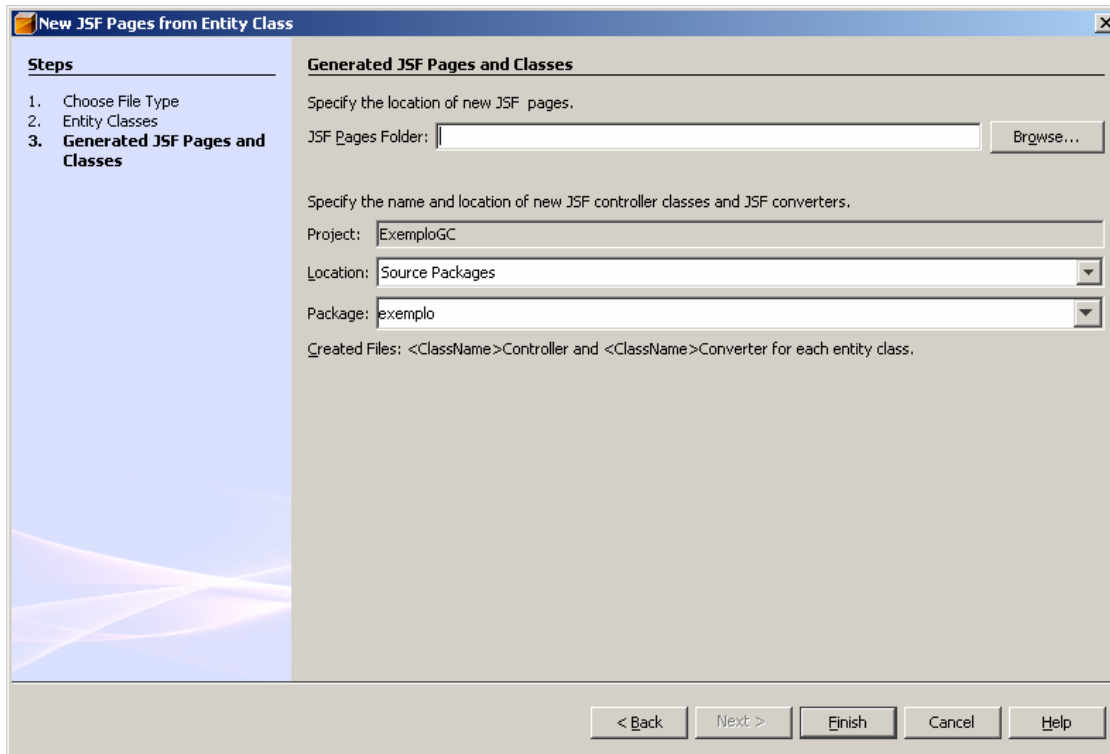
Demo: Netbeans, JPA com Hibernate e Faces

10. Clique em File -> New File e escolha “JSF Pages from Entity”



11. Escolha as entidades que deseja gerar as páginas de CRUD.

Demo: Netbeans, JPA com Hibernate e Faces

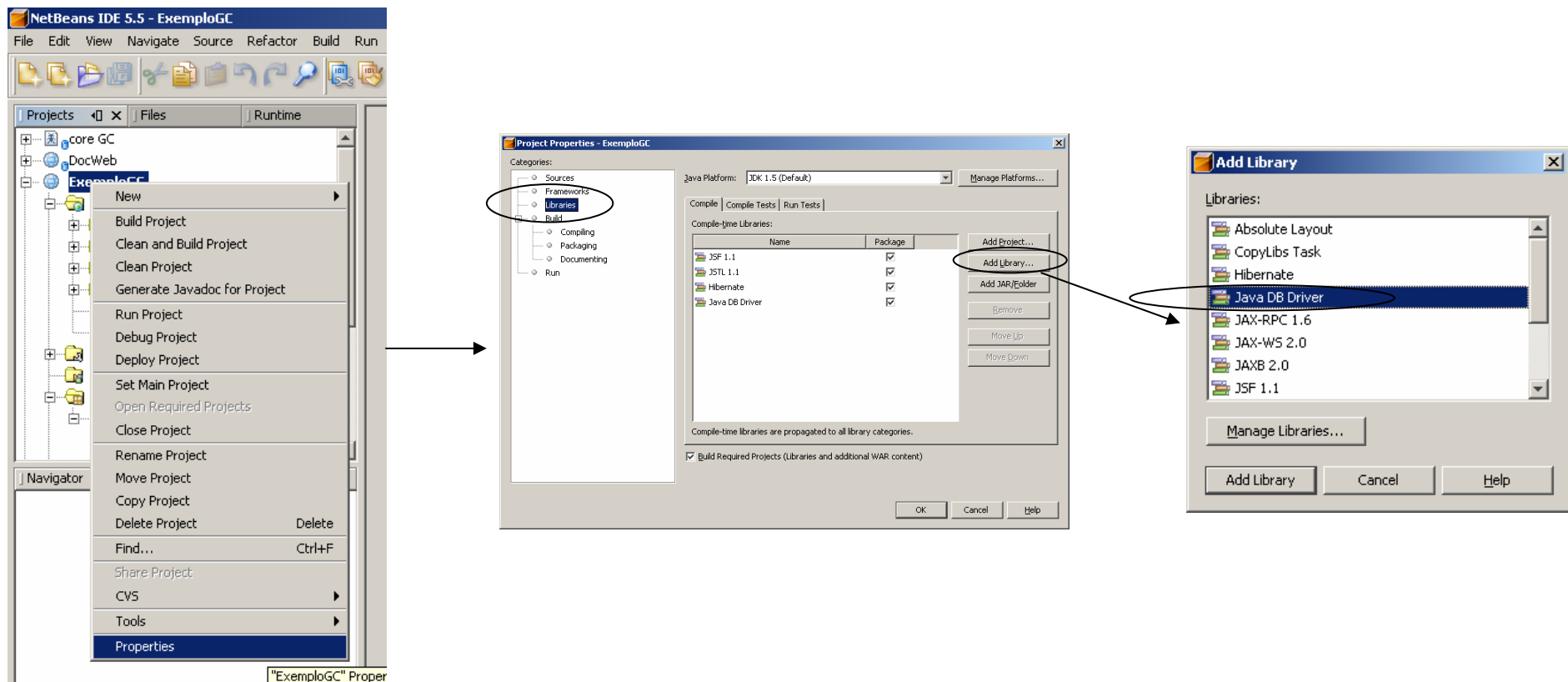


12. Agora clique em Finish para gerar automaticamente páginas JSF e controladores de páginas.

Opcionalmente você poderá escolher um sub-diretório Web para o NetBeans gravar as páginas e o package dos controladores / managed beans.

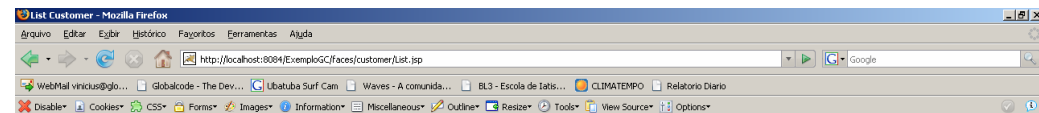
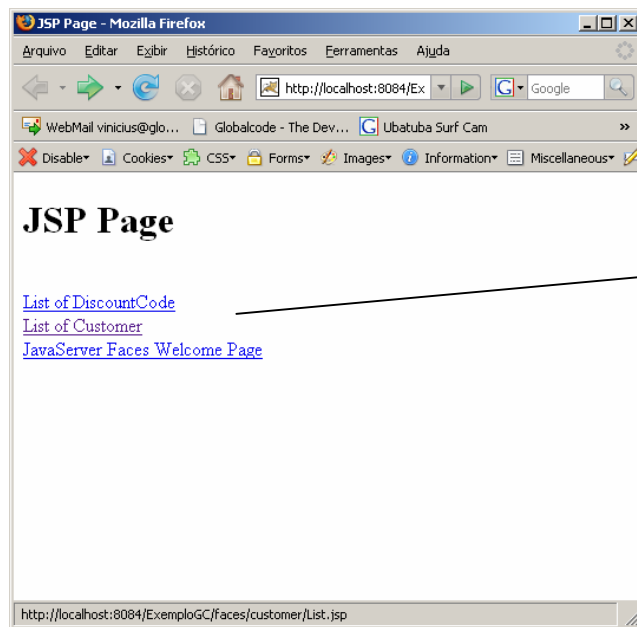
Demo: Netbeans, JPA com Hibernate e Faces

13. O penúltimo passo é adicionar a biblioteca “Java DB Driver” ou o driver JDBC do seu banco de dados.



Demo: Netbeans, JPA com Hibernate e Faces

14. Clique em Run e você verá o resultado, um menu com a entidades geradas que permitem o acesso a listagem de dados para edição, adição ou exclusão dos dados.



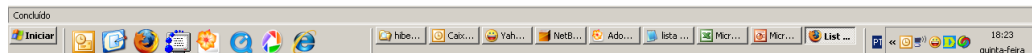
Listing Customers

[New Customer](#)

[Back to index](#)

Item 1, 13 of 13

CustomerId	Zip	Name	Addressline1	Addressline2	City	State	Phone	Fax	Email	CreditLimit	DiscountCode	
1	33015	JumboCom	111 E Las Olas Blvd	Suite 51	Fort Lauderdale	FL	305-777-4632	305-777-4635	jumbocom@gmail.com	100000	N	Destroy Edit
2	33055	Livermore Enterprises	9754 Main Street	P.O. Box 567	Miami	FL	305-456-8888	305-456-8889	www.tsoft.com	50000	M	Destroy Edit
25	75200	Oak Computers	8989 Qume Drive	Suite 9897	Houston	TX	214-999-1234	214-999-5432	www.oakc.com	25000	M	Destroy Edit
3	12347	Nano Apple	8585 Murray Drive	P.O. Box 456	Alanta	GA	555-275-9900	555-275-9911	www.nanoapple.net	90000	L	Destroy Edit
36	94401	HostProCom	62655 El Camano	Suite 2523	San Mateo	CA	650-426-8876	650-426-1120	www.hostprocom.net	65000	H	Destroy Edit
106	95035	CentralComp	829 Flex Drive	Suite 853	San Jose	CA	408-987-1256	408-987-1277	www.centralcomp.com	26500	L	Destroy Edit
149	95117	Golden Valley Computers	4381 Kelly Ave	Suite 77	Santa Clara	CA	408-432-6868	408-432-6899	www.gvc.net	70000	L	Destroy Edit
863	94401	Top Network Systems	456 4th Street	Suite 45	Redwood City	CA	650-345-5656	650-345-4433	www.hpsys.net	25000	N	Destroy Edit
777	48128	West Valley Inc.	88 North Drive	Building C	Dearborn	MI	313-563-9900	313-563-9911	www.westv.com	100000	L	Destroy Edit
753	48128	Ford Motor Co	2267 Michigan Ave	Building 21	Dearborn	MI	313-787-2100	313-787-3100	www.parts@ford.com	5000000	H	Destroy Edit
722	48124	Big Car Parts	52963 Outer Dr	Suite 35	Detroit	MI	313-788-7682	313-788-7600	www.sparts.com	50000	N	Destroy Edit
409	10092	New Media Productions	4400 22nd Street	Suite 562	New York	NY	212-222-5656	212-222-5600	www.nymedia.com	10000	L	Destroy Edit
410	10096	Yankee Computer Repair	9653 33rd Ave	Floor 4	New York	NY	212-535-7000	212-535-7100	www.nycmp@repair.com	25000	M	Destroy Edit



Conclusões

- O que já era bom (Hibernate 3), ficou ainda melhor! (3.2 com Persistence API);
- O foco das especificações na facilidade de uso (ex. EJB, Faces, Java 5) começa a fazer diferença no mercado.
- Você ainda escreve código SQL???