AULA 07 – HIBERNATE

Ao término desse capítulo você terá aprendido:

- ✓ *Fundamentos do MVC*
- ✓ *Estrutura dos pacotes no NetBeans*

O JDBC é uma das maneiras mais diretas para desenvolver um aplicativo Java que interaja com um banco de dados. Essa tecnologia atende a um percentual considerável das necessidades para operações das mais diversas naturezas. Contudo, conforme nossas aplicações evoluem, sentimos necessidade de padronizar sua arquitetura, o padrão de codificação e as próprias operações SQL realizadas por ela.

O Hibernate é um framework open source de mapeamento objeto/relacional desenvolvido em Java, ou seja, ele transforma objetos definidos pelo desenvolvedor em dados tabulares de uma base de dados, portanto com ele o programador se livra de escrever uma grande quantidade de código de acesso ao banco de dados e de SQL.

Se comparado com a codificação manual e SQL, o Hibernate é capaz de diminuir 95% das tarefas relacionadas a persistência.

7.1 Introdução

A adoção do Hibernate em um sistema tende a ser uma tendência natural quando visamos algum tipo de padronização. Contudo, o Hibernate não faz a "mágica" sozinho. Para conseguir extrair o máximo que a tecnologia fornece, bem como entender o conceito do funcionamento da especificação JPA (Java Persistence API), entender o modelo relacional e da linguagem SQL é muito importante.

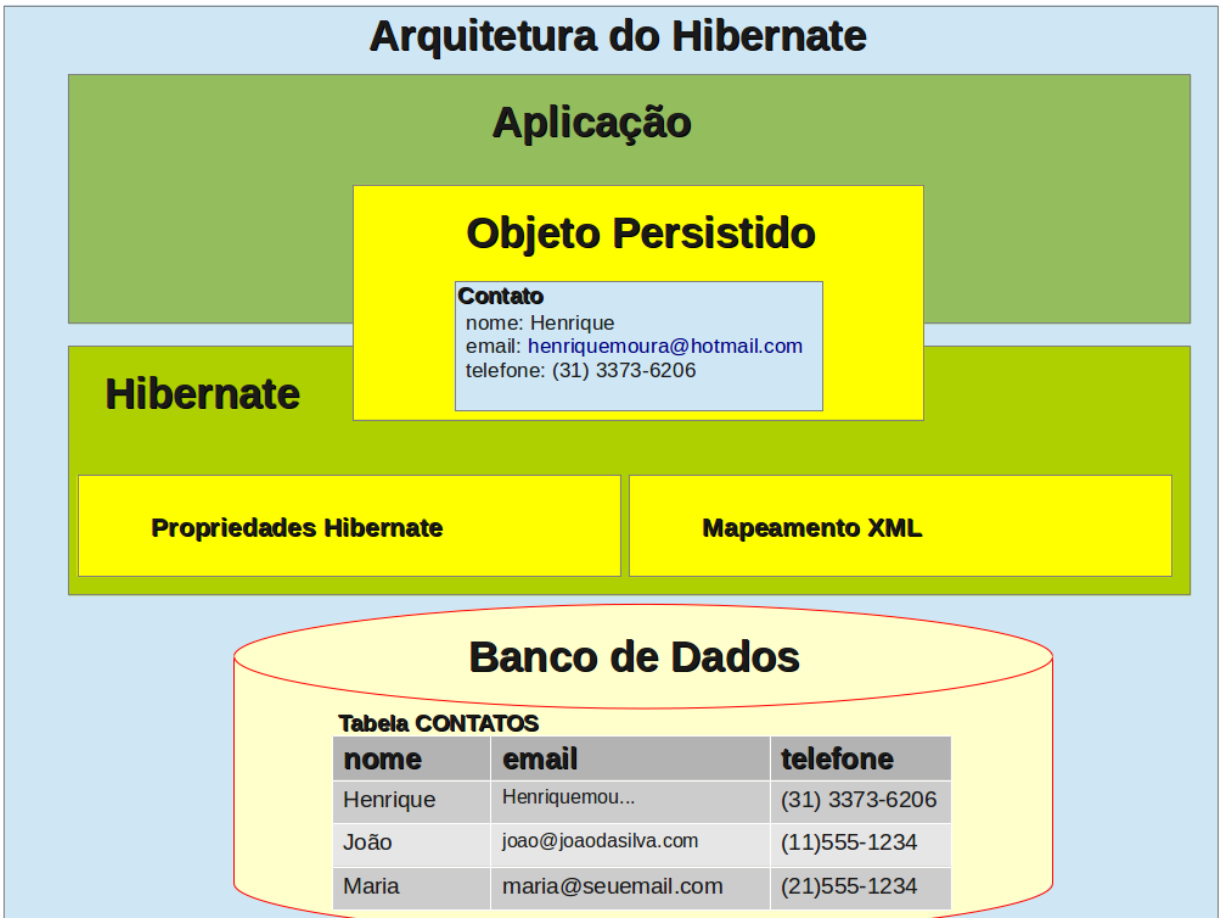
O Hibernate é apenas uma das soluções ORM (*Object Relational Mapping*, ou *Mapeamento de Objeto-Relacional*) encontradas hoje no mercado para a linguagem Java, embora seja a mais utilizada. Existem outras, como o TopLink da Oracle e OpenJPA da Apache. Independentemente disso e seja qual for a solução que venha a ser adotada pelo desenvolvedor em seus projetos, todas tendem a seguir a especificação JPA, que integra uma especificação maior, a EJB 3.0, responsável por padronizar o modelo de programação EJB. Vale destacar que o Hibernate tem três fontes de software para lidar com essa especificação:

- ✓ ***Hibernate Core***: também conhecido como Hibernate3 ou simplesmente Hibernate. É a base de todo o conjunto de soluções para persistência que essa tecnologia

oferece, contendo um API nativa e metadados de mapeamento guardados em arquivos XML. Tem uma linguagem própria estilo SQL, denominada HQL, e conta também com interfaces para realização de consultas, como a Criteria.

- ✓ ***Hibernate Annotations***: Uma nova maneira de fazer o mapeamento de objetos relacional utilizando annotations, um tipo de tags especiais estilo JavaDoc. O Hibernate segue o padrão JPA quanto a essas tags e adiciona algumas próprias. A vantagem de usar essa abordagem para mapeamento é que ela reduz o número de linhas para realizar esse processo, em comparação ao tradicional uso do XML.
- ✓ ***Hibernate EntityManager***: É uma camada que atende aos conceitos de programação de interfaces e funcionalidades de consultas, entre outros aspectos constantes na JPA. Essa interface lê o metadado ORM de uma entidade e realiza operações de persistência.

Apesar de ter tantas ferramentas e classes compondo seu núcleo, a arquitetura do Hibernate pode ser simplificada, conforme mostra a figura abaixo. A figura lista duas técnicas de mapeamento (XML e annotation). Apesar de ser possível misturá-las, isso não é recomendado em um projeto.



7.2 Vantagens

A utilização de código SQL dentro de uma aplicação agrava o problema da independência de plataforma de banco de dados e complica, em muito, o trabalho de mapeamento entre classes e banco de dados relacional.

O Hibernate abstrai o código SQL da nossa aplicação e permite escolher o tipo de banco de dados enquanto o programa está rodando, permitindo mudar sua base sem alterar nada no seu código Java.

Além disso, ele permite criar suas tabelas do banco de dados de um jeito bem simples, não se fazendo necessário todo um design de tabelas antes de desenvolver seu projeto que pode ser muito bem utilizado em projetos pequenos.

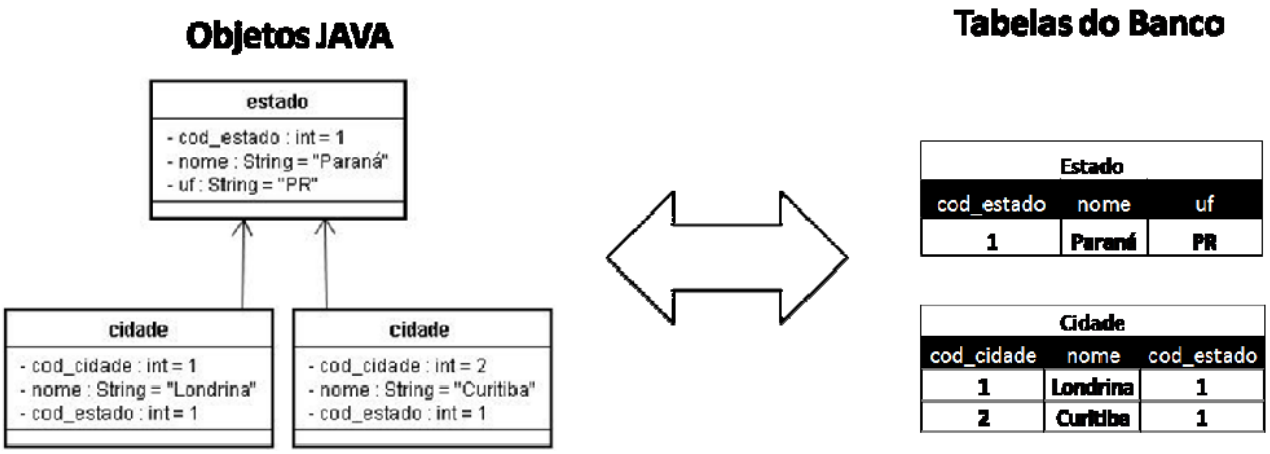
O Hibernate não apresenta apenas a função de realizar o mapeamento objeto relacional. Também disponibiliza um poderoso mecanismo de consulta de dados, permitindo uma redução considerável no tempo de desenvolvimento da aplicação.

7.3 Conceito de ORM

A maneira mais comuns de se armazenar dados é em base de dados relacionais, porém as linguagens orientadas a objeto vêm se desenvolvendo muito e torna-se necessário que a interação entre os bancos de dados relacionais ocorra da maneira mais funcional e simples possível.

Para que essa comunicação ocorra é necessário converter objetos em tabelas e tabelas em objetos, e muitas vezes os dados não são compatíveis (os tipos de dados de uma linguagem não são compatíveis com os do banco de dados).

O ORM (*Object Relational Mapping - Mapeamento Objeto/Relacional*) faz a transformação entre objetos e linhas de tabelas, como a ilustra a figura abaixo, com um exemplo de armazenamento da cidade que contem um objeto estado, nas tabelas estado e cidade, sendo que a tabela cidade possui uma chave estrangeira de estado.



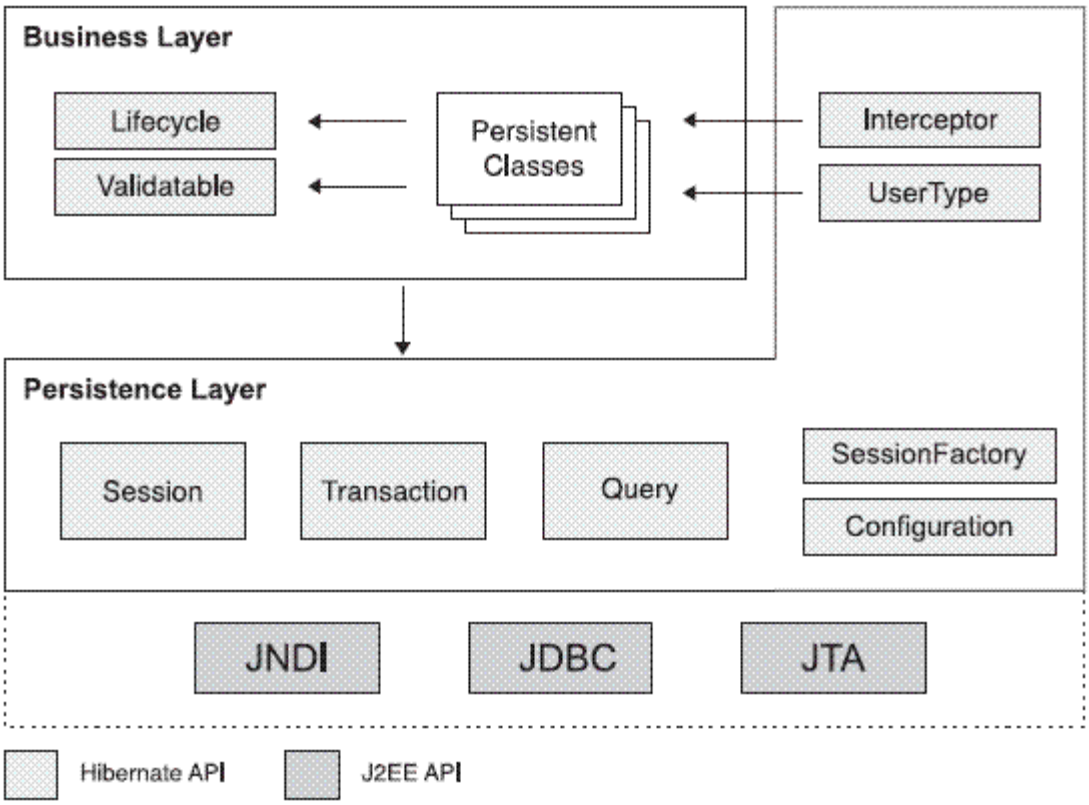
7.4 Arquitetura

A arquitetura do Hibernate é formada basicamente por um conjunto de interfaces. A próxima figura ilustra as interfaces mais importantes nas camadas de negócio e persistência.

A camada de negócio aparece acima da camada de persistência por atuar como uma cliente da camada de persistência. As interfaces do Hibernate podem ser classificadas como:

- ✓ Interfaces chamadas pela aplicação para executar operações básicas do CRUD (Create, Retreive, Update, Delete). Essas são as principais interfaces de dependência entre a lógica de negócios/controle da aplicação e o Hibernate. Estão incluídas *Session*, *Transactione Query*.
- ✓ Interfaces chamadas pela infra-estrutura da aplicação para configurar o Hibernate, mais especificamente *Configuration*;
- ✓ Interfaces responsáveis por realizar a interação entre os eventos do Hibernate e a aplicação: *Interceptor*, *Lifecycle e Validatable*.
- ✓ Interfaces que permitem a extensão das funcionalidades de mapeamento do Hibernate: *UserType*, *CompositeUserType*, *IdentifierGenerator*.

O Hibernate também interage com APIs já existentes do Java: JTA, JNDI e JDBC.



De todas as interfaces apresentadas na figura acima, as principais são: ***Session***, ***SessionFactory***, ***Transaction***, ***Query***, ***Configuration***. Os sub-tópicos seguintes apresentam uma descrição mais detalhada sobre elas.

7.4.1 Session (org.hibernate.Session)

O objeto ***Session*** é aquele que possibilita a comunicação entre a aplicação e a persistência, através de uma conexão JDBC. É um objeto leve de ser criado, não deve ter tempo de vida por toda a aplicação e não é threadsafe. Um objeto ***Session*** possui um cache local de objetos recuperados na sessão. Com ele é possível criar, remover, atualizar e recuperar objetos persistentes.

7.4.2 SessionFactory (org.hibernate.SessionFactory)

O objeto ***SessionFactory*** é aquele que mantém o mapeamento objeto relacional em memória. Permite a criação de objetos ***Session***, a partir dos quais os dados são acessados, também denominado como fábrica de objetos ***Sessions***.

Um objeto SessionFactory é threadsafe, porém deve existir apenas uma instância dele na aplicação, pois é um objeto muito pesado para ser criado várias vezes.

7.4.3 Configuration (org.hibernate.Configuration)

Um objeto ***Configuration*** é utilizado para realizar as configurações de inicialização do Hibernate. Com ele, definem-se diversas configurações do Hibernate, como por exemplo: o driver do banco de dados a ser utilizado, o dialeto, o usuário e senha do banco, entre outras. É a partir de uma instância desse objeto que se indica como os mapeamentos entre classes e tabelas de banco de dados devem ser feitos.

7.4.4 Transaction (org.hibernate.Transaction)

A interface ***Transaction*** é utilizada para representar uma unidade indivisível de uma operação de manipulação de dados. O uso dessa interface em aplicações que usam Hibernate é opcional. Essa interface abstrai a aplicação dos detalhes das transações JDBC, JTA ou CORBA.

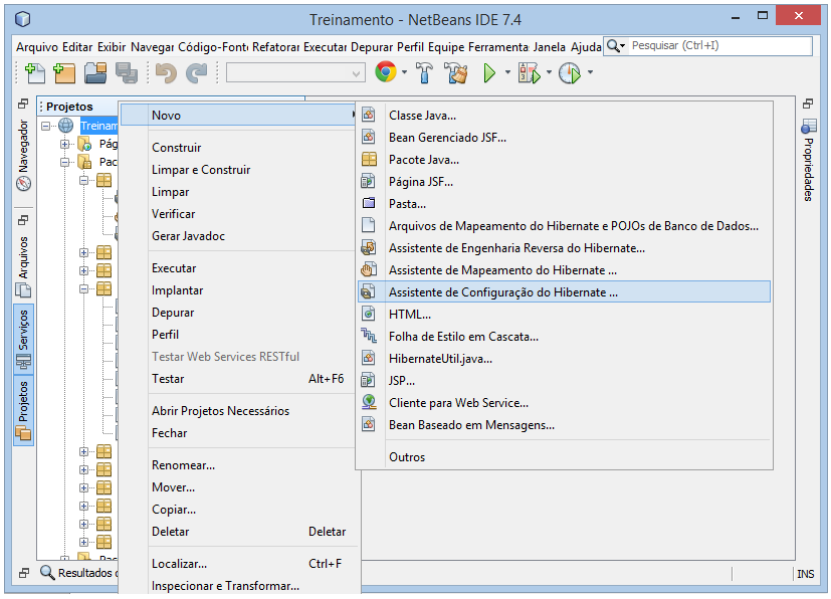
7.4.5 Interfaces Criteria e Query

As interfaces ***Criteria*** e ***Query*** são utilizadas para realizar consultas ao banco de dados.

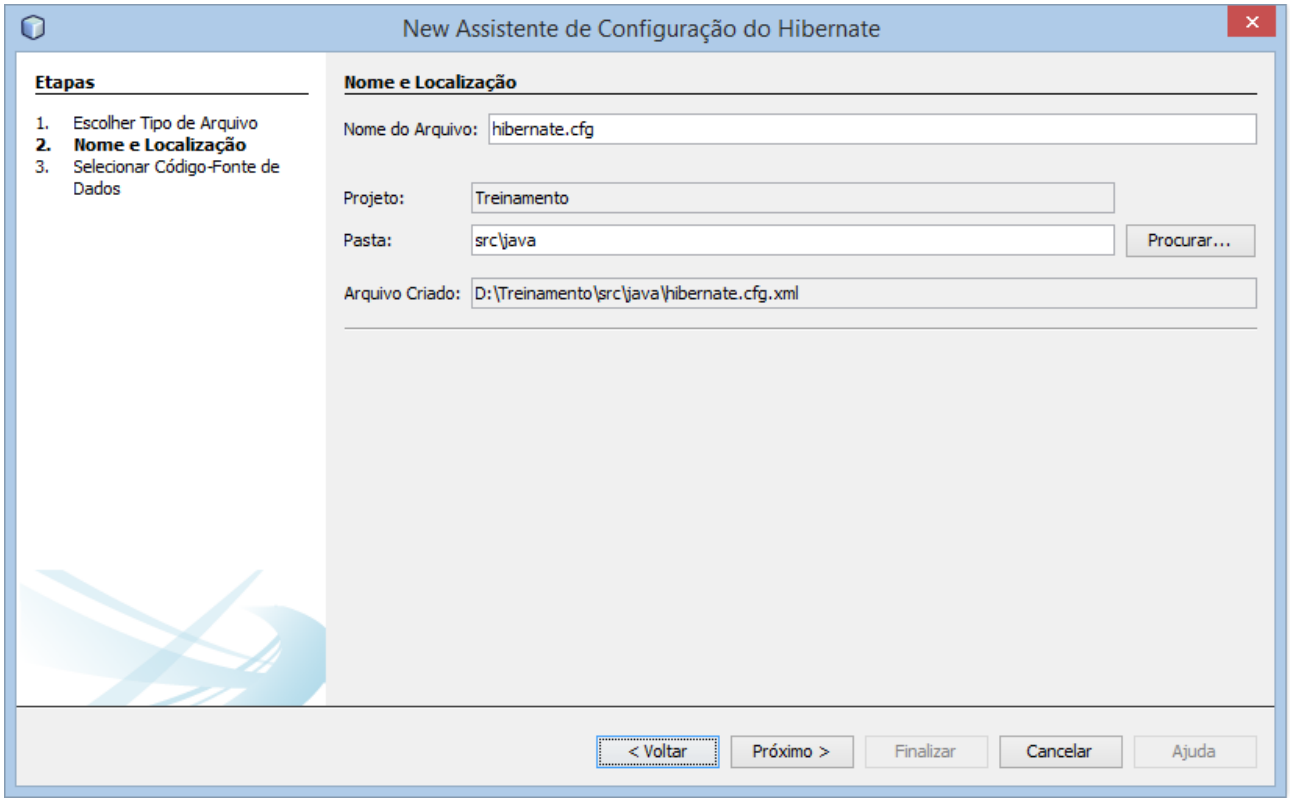
7.5 Configuração do Hibernate

Vamos utilizar o assistente de configuração do Hibernate no Netbeans para gerar o arquivo ***hibernate.cfg.xml***. Ele é responsável por armazenar as informações da conexão como driver, url, usuário, senha e etc.

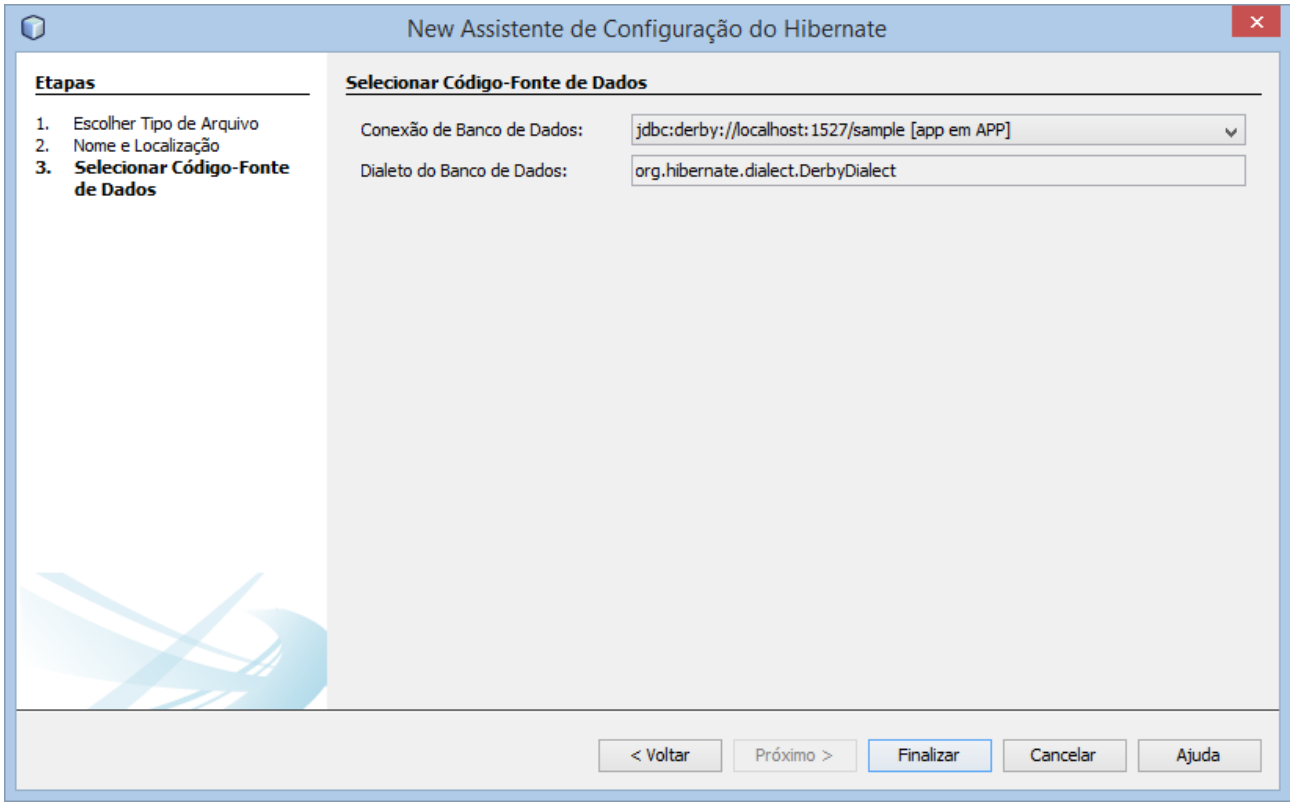
Para isso você deve clicar com o botão direito no projeto “Treinamento” → “Novo” → “Assistente de Configuração do Hibernate”.



Na próxima tela, pode manter as configurações padrões e clique em “Próximo”.

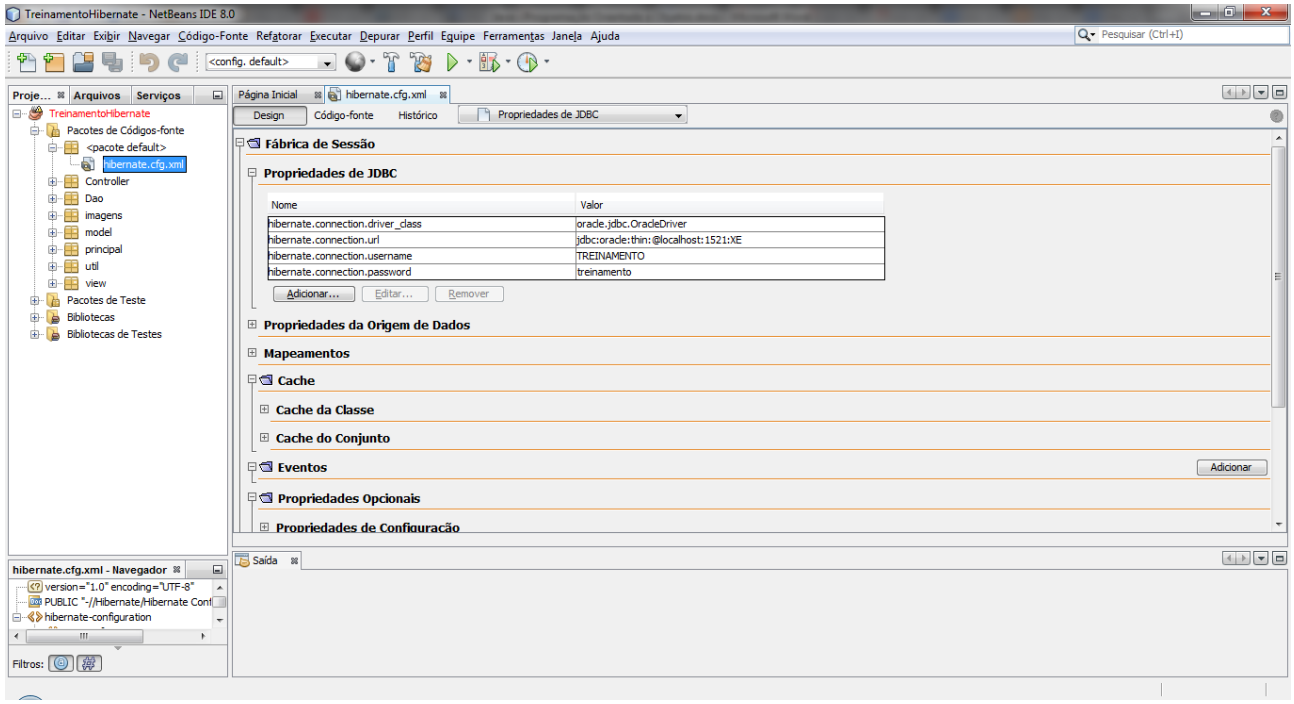


Na terceira etapa é possível escolher uma conexão com o banco de dados ou criar uma nova conexão, através de um driver específico.



Após esse processo, o arquivo ***hibernate.cfg.xml*** será criado em <pacote default>, dentro de “Pacotes de Códigos-fonte”, podendo ser alterado em modo ***design*** ou pelo ***código-fonte*** gerado (XML).

A figura abaixo mostra o arquivo de configuração do hibernate no modo design, com suas propriedades e respectivos valores.



7.6 Mapeamento das classes

Como os bancos e dados não entendem dados orientados a objetos, a solução utilizada pelo Hibernate é utilizar um identificador não natural, assim o banco de dados é capaz de compreender os objetos e montar seus relacionamentos.

Assim o mapeamento consiste em relacionar cada campo de uma tabela da base de dados a uma variável de uma classe, e também montar os identificadores não naturais. No Hibernate, há duas maneiras de fazer o mapeamento, via XML, ou utilizando anotações.

7.6.1 Mapeamento via XML

No mapeamento via XML são criados arquivos XML que devem ter a extensão ***.hbm.xml***, e também devem ser referenciados no arquivo de configuração. A desvantagem desse tipo de mapeamento é que para cada tabela da base de dados deve ser criado um arquivo de mapeamento e uma classe POJO. Esse processo é mais

trabalhoso que o mapeamento utilizando anotações, porém, há ferramentas, como o XDoclet, que são utilizadas para gerar os mapeamentos.

7.6.2 Mapeamento via Anotações

Com o mapeamento via anotações, não é necessário criar nenhum arquivo XML (ou em qualquer outro formato) para fazer o mapeamento, basta somente colocar as anotações (annotations) na classe POJO relacionada à tabela.

Abaixo segue um exemplo do mapeamento da tabela usuário utilizando anotações do Hibernate.


```
 1 package model;
 2
 3 import javax.persistence.Column;
 4 import javax.persistence.Entity;
 5 import javax.persistence.Id;
 6 import javax.persistence.Table;
 7
 8 @Entity
 9 @Table(name = "USUARIO", schema = "TREINAMENTO")
10 public class Usuario implements java.io.Serializable {
11     private long usuId;
12     private String usuNome;
13     private String usuLogin;
14     private String usuSenha;
15     private Boolean usuAtivo;
16
17     @Id
18     @Column(name = "USU_ID", unique = true, nullable = false, precision = 10,
scale = 0)
19     public long getUsuId() {
20         return this.usuId;
21     }
22
23     public void setUsuId(long usuId) {
24         this.usuId = usuId;
25     }
26
27     @Column(name = "USU_NOME", nullable = false, length = 30)
28     public String getUsuNome() {
29         return this.usuNome;
30     }
31
32     public void setUsuNome(String usuNome) {
33         this.usuNome = usuNome;
34     }
35 ...
36 }
```

Para que o código acima funcione corretamente, devem ser adicionados a ele os construtores e métodos ***get*** e ***set*** para cada atributo.

Também podemos observar algumas anotações no código anterior, que possuem as seguintes funcionalidades:

@Entity: declara a classe como uma entidade, ou seja, uma classe persistente.

@Table: define qual tabela da base de dados será mapeada.

@Id: define qual campo será usado como identificador.

@Column: define qual coluna da tabela será mapeada.

Existem ainda muitas anotações Hibernate que podem ser encontradas na documentação oficial no site.

AULA 07 – DESENVOLVIMENTO DE CADASTRO COM HIBERNATE

Ao término desse capítulo você terá aprendido:

- ✓ *Criação de um cadastro utilizando Hibernate e MVC*

Este capítulo contém a criação de um projeto que será composto de pacotes e classes no padrão MVC, efetuando conexão e manipulação a uma tabela do banco de dados através do Hibernate.

A seguir, são apresentados os arquivos para controle do Hibernate, e cada classe necessária ao cadastro da tabela USUARIO, dentro do seu respectivo pacote no padrão MVC.

7.1 Arquivo de configuração do Hibernate

```
1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-
configuration-3.0.dtd">
 3 <hibernate-configuration>
 4     <session-factory>
 5         <property
name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
 6         <property
name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>
 7         <property
name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</property>
 8         <property name="hibernate.connection.username">TREINAMENTO</property>
 9         <property name="hibernate.connection.password">treinamento</property>
10         <mapping class="model.Usuario"/>
11         <mapping class="model.Paciente"/>
12         <mapping class="model.Convenio"/>
13         <mapping class="model.Especialidade"/>
14         <mapping class="model.Cidade"/>
15         <mapping class="model.Agendamento"/>
16     </session-factory>
17 </hibernate-configuration>
```

7.2 Arquivo HibernateUtil

```
 1 package util;
 2
 3 import org.hibernate.HibernateException;
 4 import org.hibernate.cfg.AnnotationConfiguration;
 5 import org.hibernate.SessionFactory;
 6
13 public class HibernateUtil {
14
15     private static final SessionFactory sessionFactory;
16
17     static {
18         try {
19             // Create the SessionFactory from standard (hibernate.cfg.xml)
20             // config file.
21             sessionFactory = new
AnnotationConfiguration().configure().buildSessionFactory();
22         } catch (HibernateException ex) {
23             // Log the exception.
24             System.err.println("Initial SessionFactory creation failed." +
ex);
25             throw new ExceptionInInitializerError(ex);
26         }
27     }
28
29     public static SessionFactory getSessionFactory() {
30         return sessionFactory;
31     }
32 }
```

7.3 Classe da Camada Model do MVC

```
 1 package model;
 2
 3 import javax.persistence.Column;
 4 import javax.persistence.Entity;
 5 import javax.persistence.Id;
 6 import javax.persistence.Table;
 7
 8 @Entity
 9 @Table(name = "USUARIO", schema = "TREINAMENTO")
10 public class Usuario implements java.io.Serializable {
11     private long usuId;
12     private String usuNome;
13     private String usuLogin;
14     private String usuSenha;
15     private Boolean usuAtivo;
16
17     @Override
18     public String toString() {
19         return this.usuNome;
20     }
21
22     public Usuario() {
23     }
24
25     public Usuario(long usuId, String usuNome, String usuLogin, String
usuSenha) {
26         this.usuId = usuId;
```

```
27          this.usuNome = usuNome;
28          this.usuLogin = usuLogin;
29          this.usuSenha = usuSenha;
30      }
31
32      public Usuario(long usuId, String usuNome, String usuLogin, String
usuSenha, Boolean usuAtivo) {
33          this.usuId = usuId;
34          this.usuNome = usuNome;
35          this.usuLogin = usuLogin;
36          this.usuSenha = usuSenha;
37          this.usuAtivo = usuAtivo;
38      }
39
40      @Id
41      @Column(name = "USU_ID", unique = true, nullable = false, precision = 10,
scale = 0)
42      public long getUsuId() {
43          return this.usuId;
44      }
45
46      public void setUsuId(long usuId) {
47          this.usuId = usuId;
48      }
49
50      @Column(name = "USU_NOME", nullable = false, length = 30)
51      public String getUsuNome() {
52          return this.usuNome;
53      }
54
55      public void setUsuNome(String usuNome) {
56          this.usuNome = usuNome;
57      }
58
59      @Column(name = "USU_LOGIN", nullable = false, length = 10)
60      public String getUsuLogin() {
61          return this.usuLogin;
62      }
63
64      public void setUsuLogin(String usuLogin) {
65          this.usuLogin = usuLogin;
66      }
67
68      @Column(name = "USU_SENHA", nullable = false, length = 10)
69      public String getUsuSenha() {
70          return this.usuSenha;
71      }
72
73      public void setUsuSenha(String usuSenha) {
74          this.usuSenha = usuSenha;
75      }
76
77      @Column(name = "USU_ATIVO", precision = 1, scale = 0)
78      public Boolean getUsuAtivo() {
79          return this.usuAtivo;
80      }
81
82      public void setUsuAtivo(Boolean usuAtivo) {
83          this.usuAtivo = usuAtivo;
84      }
85  }
86
```

7.4 Classe da Camada DAO do MVC

```java
package Dao;

import java.util.ArrayList;
import model.Usuario;
import util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;

public class UsuarioDao {
    public void save(Usuario usuario) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction t = session.beginTransaction();
        session.save(usuario);
        t.commit();
        session.close();
    }

    public Usuario getUsuario(long id) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        return (Usuario) session.load(Usuario.class, id);
    }

    public ArrayList<Usuario> list(String filtro) {
        String sql = "from Usuario "+filtro;
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction t = session.beginTransaction();
        ArrayList lista = (ArrayList) session.createQuery(sql).list();
        t.commit();
        session.close();
        return lista;
    }

    public void remove(Usuario usuario) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction t = session.beginTransaction();
        session.delete(usuario);
        t.commit();
        session.close();
    }

    public void update(Usuario usuario) {
        Session session = HibernateUtil.getSessionFactory().openSession();
        Transaction t = session.beginTransaction();
        session.update(usuario);
        t.commit();
        session.close();
    }
}
```

7.5 Classe da Camada Controller do MVC

```
 1 package Controller;
 2
 3 import Dao.UsuarioDao;
 4 import java.util.ArrayList;
 5 import java.util.List;
 6 import model.Usuario;
 7
 8 public class UsuarioController {
 9
10     private Usuario usuario;
11     private List<Usuario> listausuarios;
12
13     public ArrayList<Usuario> getListarUsuarios(String filtro) {
14         listausuarios = new UsuarioDao().list(filtro);
15         return (ArrayList<Usuario>) listausuarios;
16     }
17
18     public Usuario getUsuario() {
19         return usuario;
20     }
21
22     public void setUsuario(Usuario usuario) {
23         this.usuario = usuario;
24     }
25
26     public boolean excluirUsuario(Usuario usuario) {
27         try {
28             UsuarioDao dao = new UsuarioDao();
29             dao.remove(usuario);
30             return true;
31         } catch (Exception e) {
32             return false;
33         }
34     }
35
36     public boolean adicionarUsuario(Usuario usuario) {
37         try {
38             UsuarioDao dao = new UsuarioDao();
39             dao.save(usuario);
40             return true;
41         } catch (Exception e) {
42             return false;
43         }
44     }
45
46     public boolean alterarUsuario(Usuario usuario) {
47         try {
48             UsuarioDao dao = new UsuarioDao();
49             dao.update(usuario);
50             return true;
51         } catch (Exception e) {
52             return false;
53         }
54     }
55
56     public boolean gravar(String operacao, Usuario usuario) {
57         boolean retorno = true;
58         if (operacao.equals("incluir")) {
59             retorno = adicionarUsuario(usuario);
60         } else if (operacao.equals("alterar")) {
```

```
61              retorno = alterarUsuario(usuario);
62          }
63          return retorno;
64      }
65  }
```

7.6 Classe da Camada View do MVC

- ✓ Abrir os arquivos:

UsuarioView.java

UsuarioTableModel.java