Hibernate

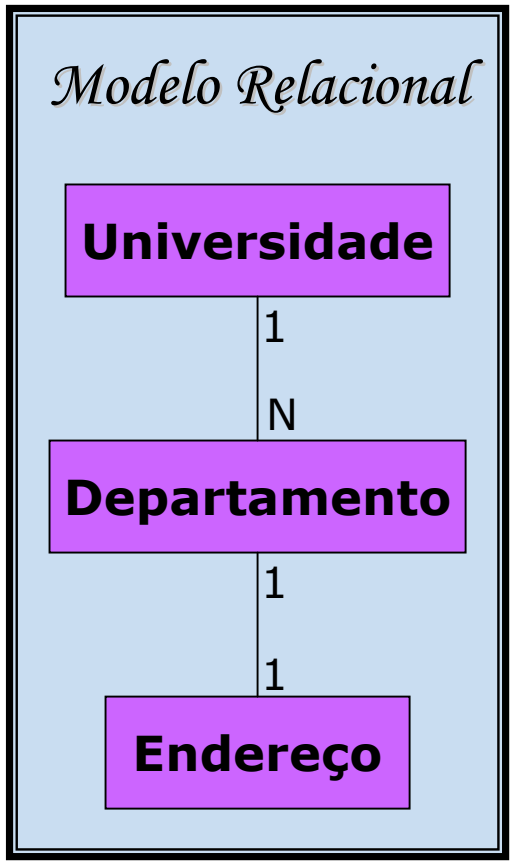
Raquel Defelippo Rodrigues

Hibernate é um framework usado para persistir objetos Java em tabelas de banco de dados relacionais.

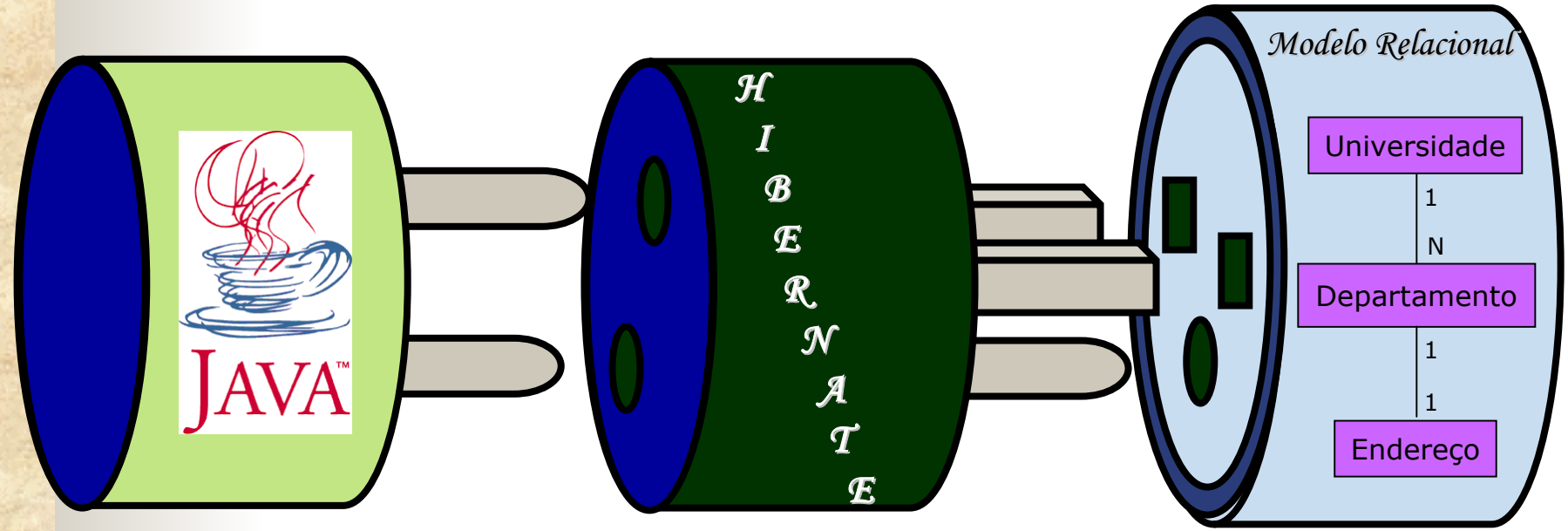
Por quê precisamos do Hibernate?

Trabalhamos com

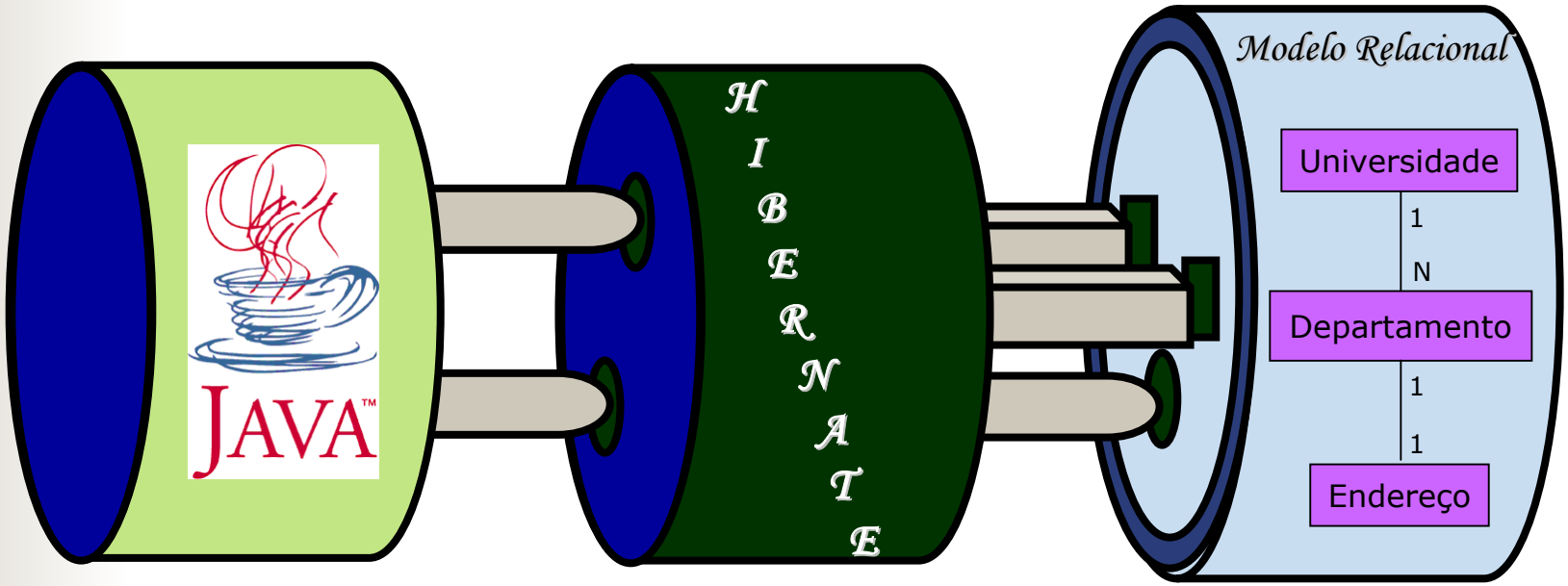
+



Java + Modelo Relacional



Java + Modelo Relacional



O **Hibernate** é um mapeamento objeto/relacional .

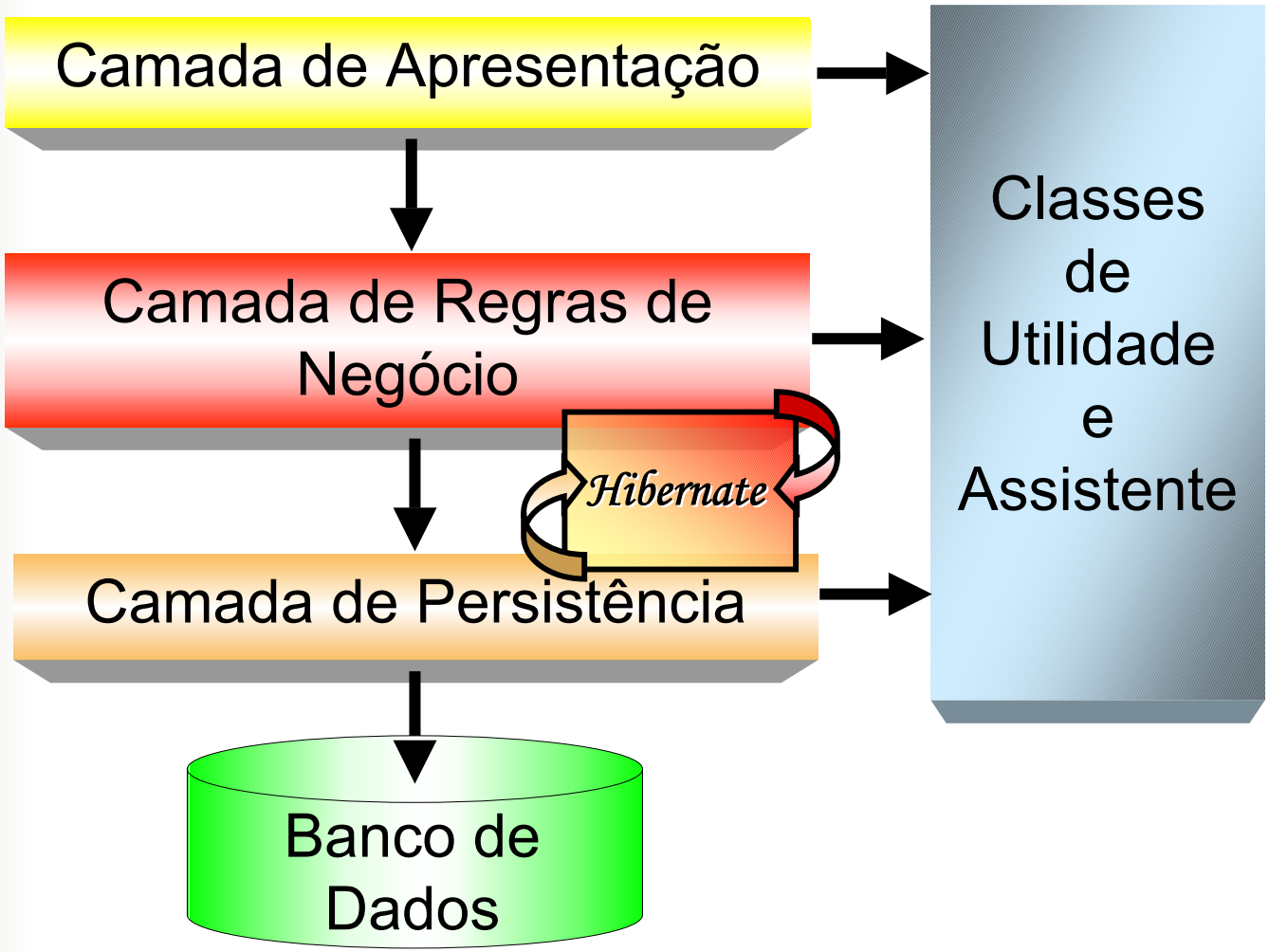
Mas o que é ORM ?
(Mapeamento Objeto/Relacional)

ORM

- *ORM é um meio de persistir objetos de um aplicativo Java em tabelas de bancos de dados relacionais, de maneira transparente.*
- *Hibernate é uma solução open source de ORM.*

Onde se localiza o Hibernate?

Arquitetura em camadas de projetos OO

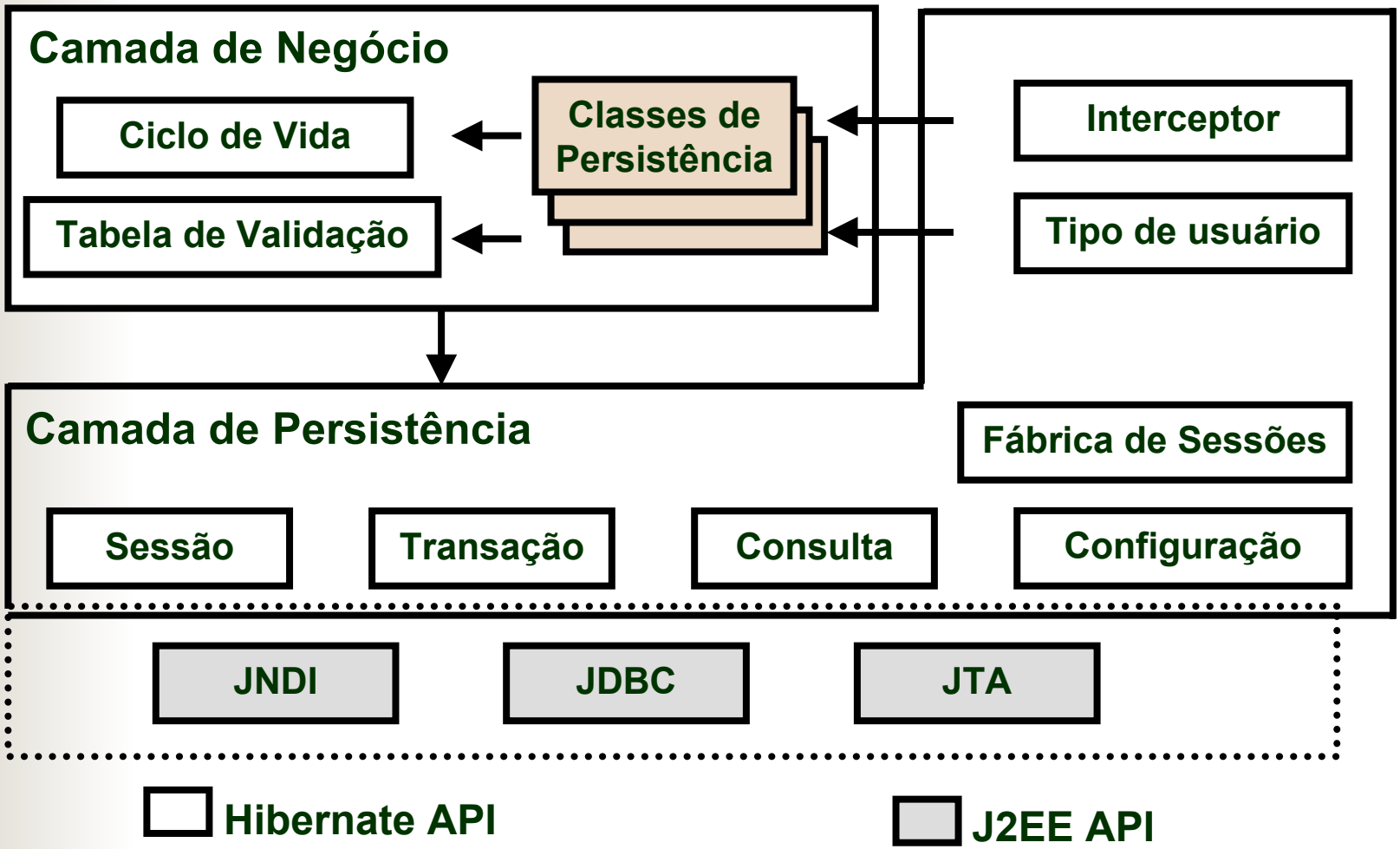


Abstração do SGBD

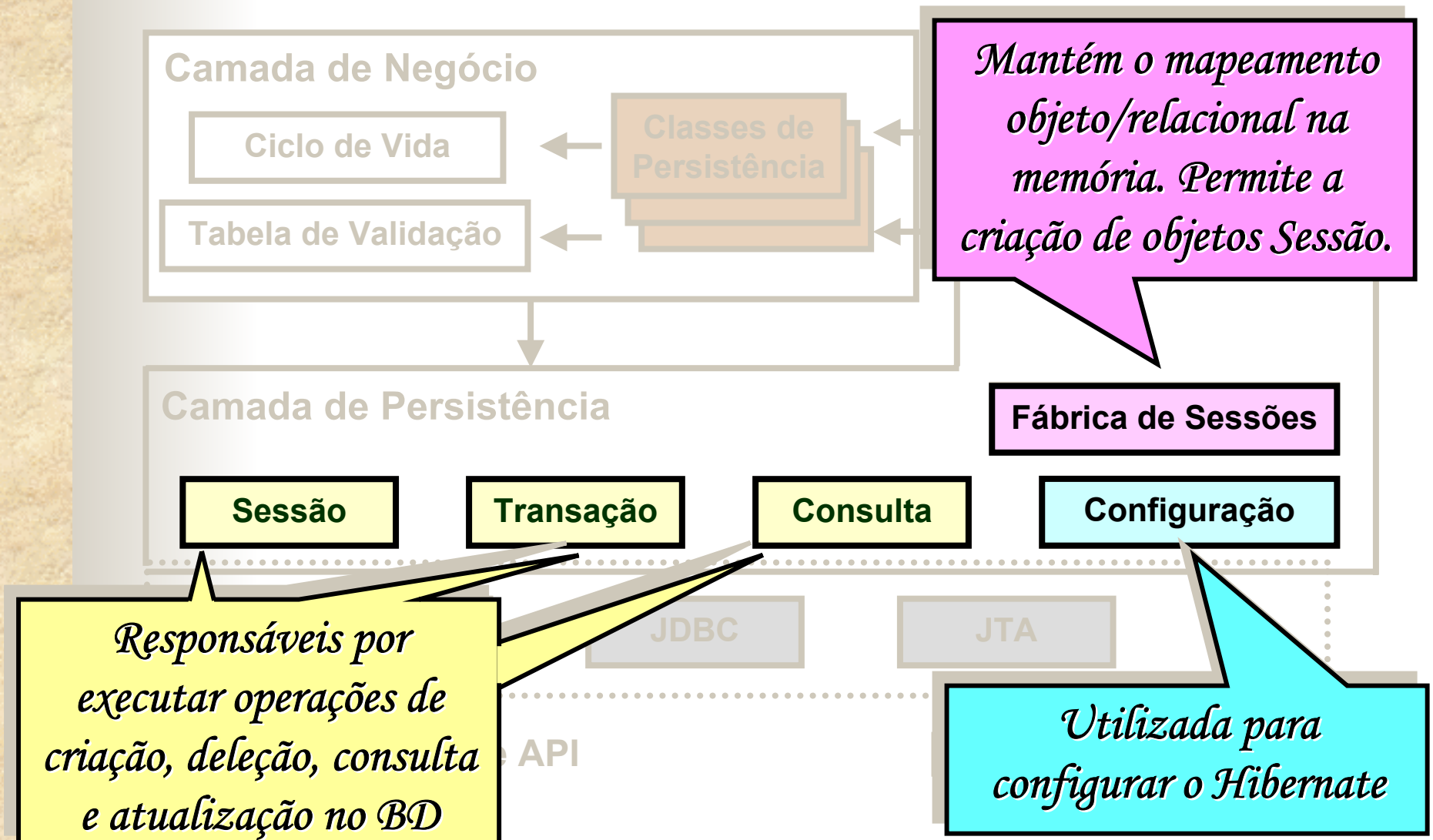
- O Hibernate está localizado entre o JDBC e a aplicação.
- Caso seja preciso mudar o SGBD de sua aplicação basta apenas alterar o driver usado e sua referência no arquivo hibernate.cfg.xml.

Como é a Arquitetura do Hibernate?

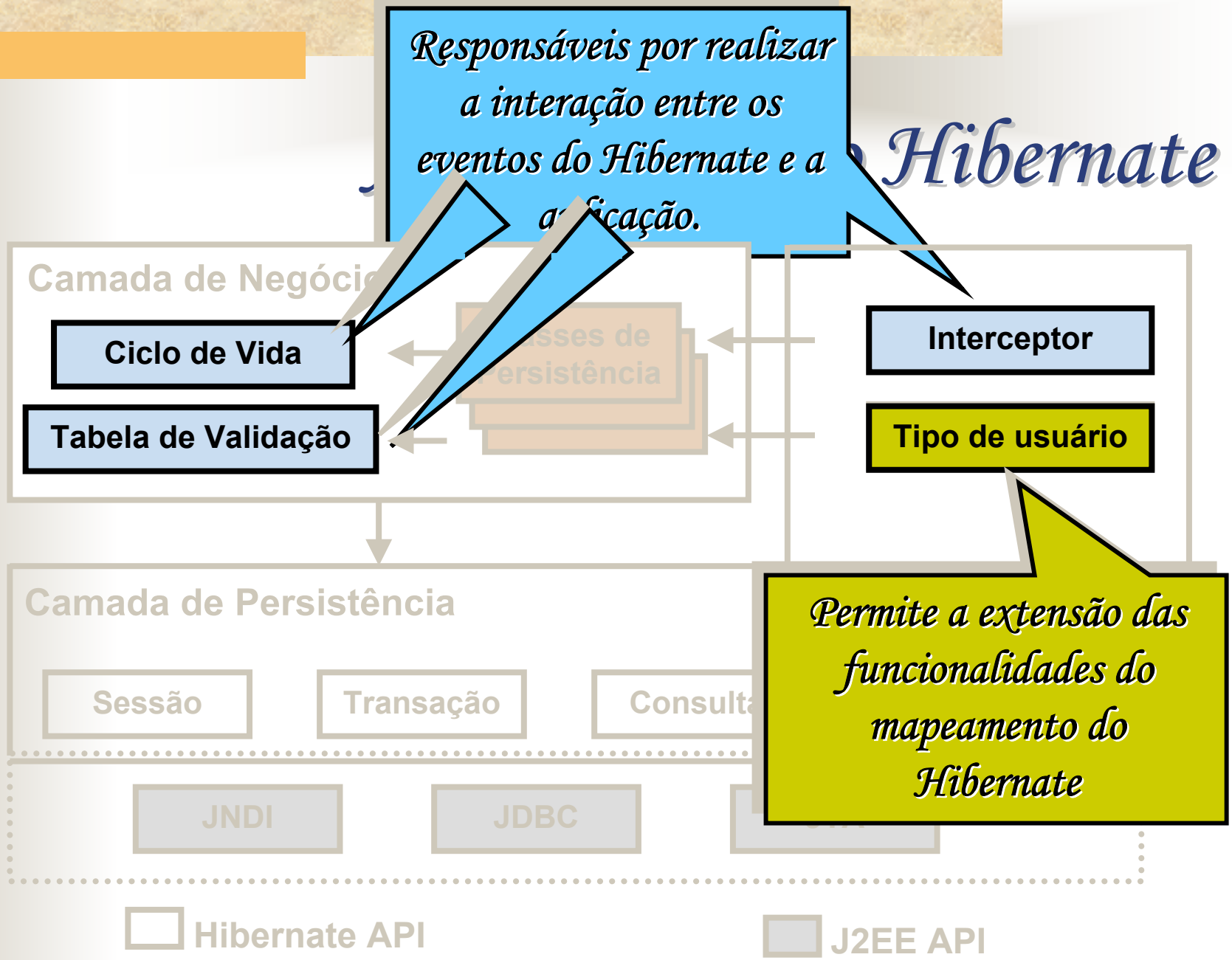
Arquitetura do Hibernate



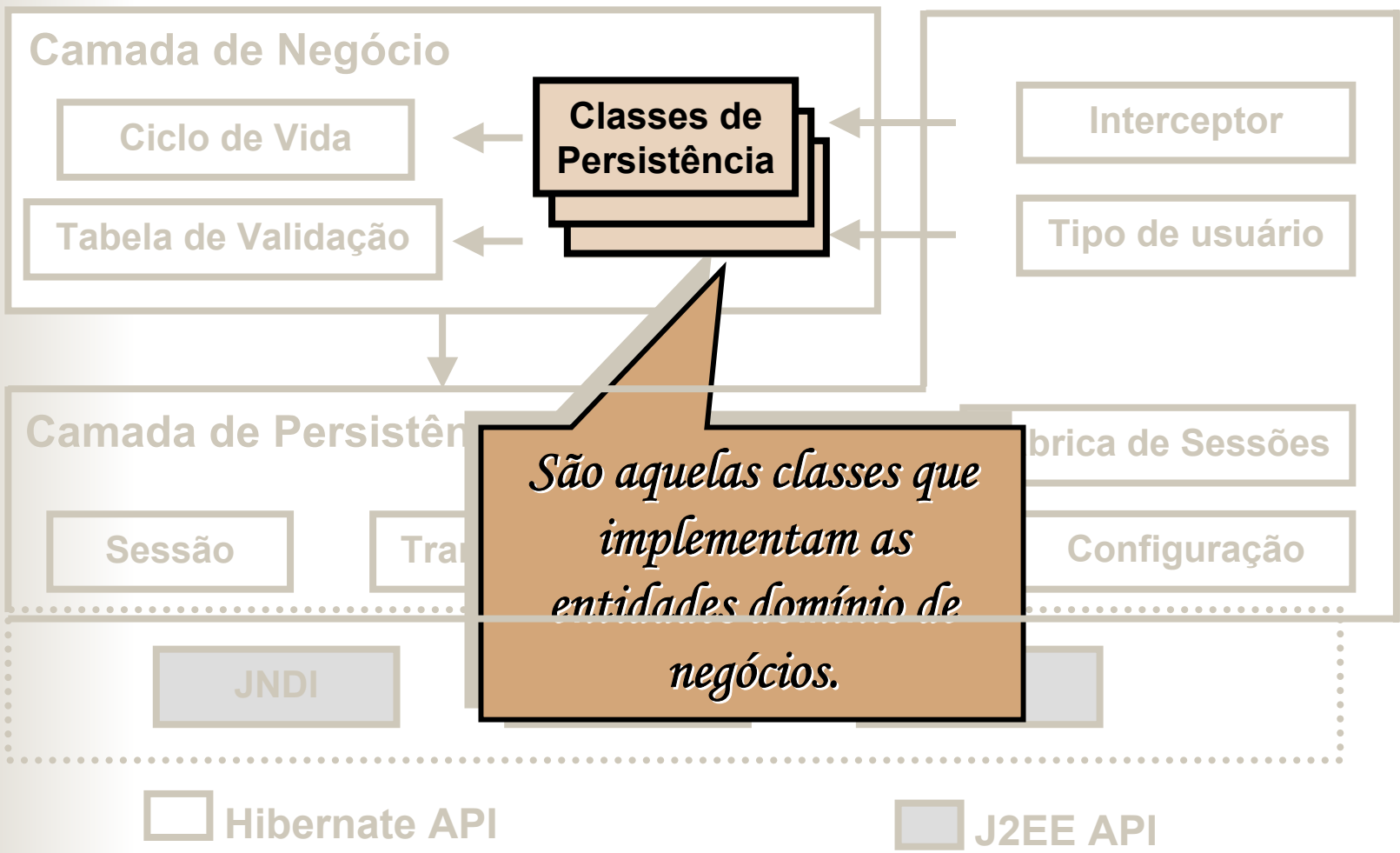
Arquitetura do Hibernate



Hibernate


Responsáveis por realizar a interação entre os eventos do Hibernate e a aplicação.
Camada de Negócio
Ciclo de Vida
Tabela de Validação
Interceptor
Tipo de usuário
Permite a extensão das funcionalidades do mapeamento do Hibernate
Camada de Persistência
Sessão
Transação
JNDI
JDBC
Hibernate API
J2EE API

Arquitetura do Hibernate



Classes de Persistência

- O Hibernate trabalha associando cada tabela do banco de dados a um POJO.
- O Hibernate persiste as propriedades no estilo JavaBeans;
- É importante que, ao utilizar o Hibernate, todos os objetos persistentes possuam um identificador e que estes sejam independentes da lógica de negócio da aplicação.

Escolhas das chaves primárias

- Chaves primárias naturais, como por exemplo **`CPF`** para a tabela **`Pessoa`**, devem ser evitadas.
- Recomenda-se fortemente que as chaves primárias das tabelas sejam sintéticas, ou seja, não possuam significados de negócios .
- O Hibernate apresenta vários mecanismos internos para a geração de chaves primárias.
  - Por exemplo, **`sequence`** que mapeia uma seqüência no PostegreSQL.

Como é a instalação do Hibernate?

Preparando o ambiente

- *Passo 1:*

  Instalação do SGBD PostegreSQL;

- *Passo 2:*

  Instalação da IDE Eclipse;

- *Passo 3:*

  Instalação do Hibernate;

- *Passo 4:*

  Configuração do Hibernate;

Passo 1:

Instalação do PostegreSQL

1. Entre no site www.postegresql.org
2. Clique em download → FTP mirros → Brazil → win32 → postegresql-8.1.4-1.zip
3. Descompacte o arquivo baixado
4. Clique com o botão direito em postegre-8.1.msi e escolha → instalar
5. Siga o passo a passo
6. Escolha a opção de Idioma = Português
7. Escolha: superusuário = admin e senha = admin

Passo 2:

Instalação da IDE Eclipse

1. Considerando que já exista um ambiente para programação em Java devidamente instalado;
2. Entre no site http://www.eclipse.org/
3. Clique em downloads → download now: Eclipse SDK 3.2
4. Escolha → [Brazil] PUCPR (ftp)
5. Salvar o arquivo eclipse-SDK-3.2-win32.zip
6. Descompactar o arquivo eclipse-SDK-3.2-win32.zip

Passo 3:

Instalação do Hibernate

1. Entre no site http://www.hibernate.org/
2. Clique em downloads → Hibernate Core → Download → hibernate-3.1.3.zip → Curitiba, Brazil → Download
3. Descompacte o arquivo baixado (hibernate-3.1.3.zip)
4. Observe que a documentação está localizada em

hibernate-3.1.3\hibernate-3.1\doc\reference\em\html\index.html

Passo 4:

Configuração do Hibernate

A configuração do Hibernate será apresentada através de um exemplo prático.

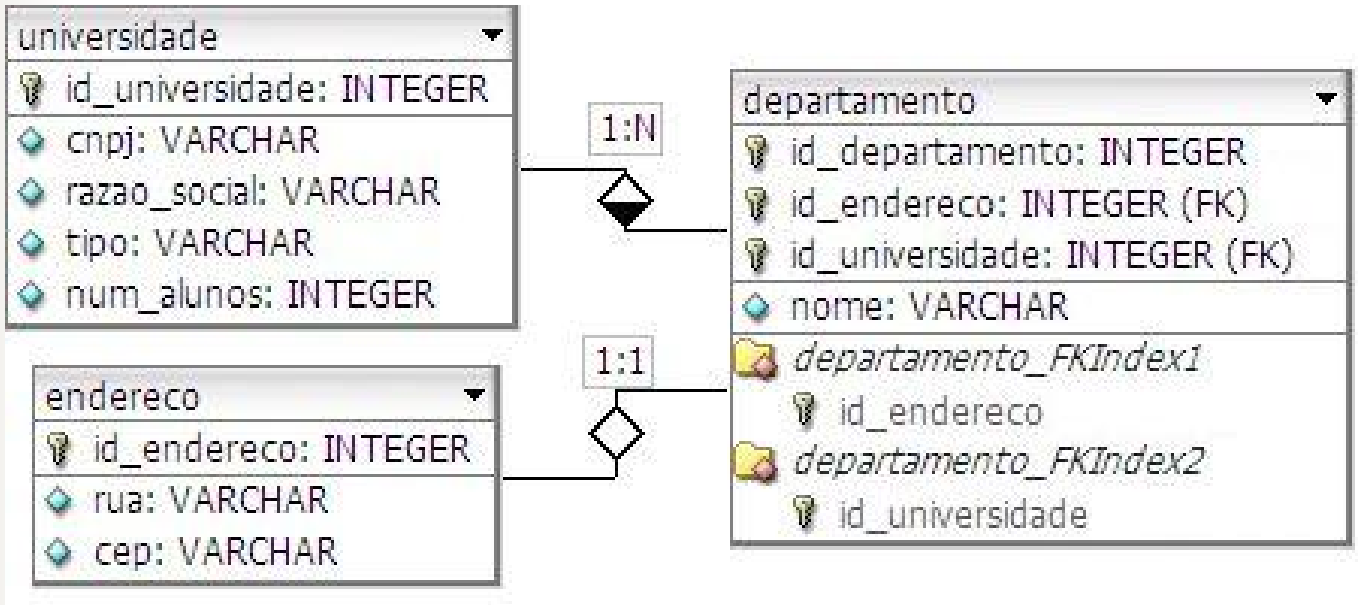
Exemplo

1. Criar o banco de dados Universidades com as 3 tabelas: Universidade, Departamento e Endereço.
2. Configurar o Hibernate.
3. Criar os arquivos ORM que relacionam as propriedades do objeto aos campos da tabela.
4. Criar os arquivos de configuração .xml contendo as propriedades para que o Hibernate se conecte ao banco de dados.
5. Testar o que foi feito.

1. Criação do Banco de Dados

Exemplo

Banco de Dados - Universidades



Criando as seqüências usadas pelas chaves primárias

```
-- Sequence: universidade_id_universidade_seq
-- DROP SEQUENCE universidade_id_universidade_seq;
CREATE SEQUENCE universidade_id_universidade_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 2
  CACHE 1;
ALTER TABLE universidade_id_universidade_seq OWNER TO
"admin";
```

Criando as seqüências usadas pelas chaves primárias

```
-- Sequence: universidade_id_universidade_seq
-- DROP SEQUENCE universidade_id_universidade_seq;
CREATE SEQUENCE universidade_id_universidade_seq
  INCREMENT 1
  MIN
  MAX
  STA
  CAC
ALTER
"admi
```

```
-- Sequence: departamento_id_departamento_seq
-- DROP SEQUENCE departamento_id_departamento_seq;
CREATE SEQUENCE departamento_id_departamento_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 2
  CACHE 1;
ALTER TABLE departamento_id_departamento_seq OWNER TO
"admin";
```

Criando as seqüências usadas pelas chaves primárias

```
-- Sequence: universidade_id_universidade_seq
-- DROP SEQUENCE universidade_id_universidade_seq;
CREATE SEQUENCE universidade_id_universidade_seq
  INCREMENT 1
  MIN
  MAX
  STA
  CAC
ALTER
"admi
```

```
-- Sequence: departamento_id_departamento_seq
-- DROP SEQUENCE departamento_id_departamento_seq;
CREATE SEQUENCE departamento_id_departamento_seq
  INCREMENT 1
  MIN
  MAX
  STA
  CAC
ALTER
"admi
```

```
-- Sequence: endereco_id_endereco_seq
-- DROP SEQUENCE endereco_id_endereco_seq;
CREATE SEQUENCE endereco_id_endereco_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 2
  CACHE 1;
ALTER TABLE endereco_id_endereco_seq OWNER TO
"admin";
```

Criando as tabelas

```
-- Table: universidade
-- DROP TABLE universidade;
CREATE TABLE universidade
(
  id_universidade int4 NOT NULL DEFAULT
nextval('universidade_id_universidade_seq'::regclass),
  cnpj varchar NOT NULL,
  razao_social varchar,
  tipo varchar,
  num_alunos int4,
  CONSTRAINT universidade_pkey PRIMARY KEY (id_universidade),
  CONSTRAINT universidade_cnpj_key UNIQUE(cnpj)
)
WITH OIDS;
ALTER TABLE universidade OWNER TO "admin";
```

Criando as tabelas

```
-- Table: universidade
-- DROP TABLE universidade;
CREATE TABLE universidade
(
  id_universidade int4 NOT NULL DEFAULT
nextval('universidade_id_universidade_seq'::regclass),
  cnpj varchar NOT NULL,
  razao_social varchar,
  tipo varchar,
  num_alunos int4,
  CONSTRAINT universi
  CONSTRAINT universi
)
WITH OIDS;
ALTER TABLE universid
```

```
-- Table: endereco
-- DROP TABLE endereco;
CREATE TABLE endereco
(
  id_endereco int4 NOT NULL DEFAULT
nextval('endereco_id_endereco_seq'::regclass),
  rua varchar,
  cep varchar,
  CONSTRAINT endereco_pkey PRIMARY KEY (id_endereco)
)
WITH OIDS;
ALTER TABLE endereco OWNER TO "admin";
```

Criando as tabelas

```
-- Table: universidade
-- DRO
CREATE
(
  id_u
nextva
  cnpj
  raza
  tipo
  num_
  CONS
  CONS
)
WITH O
ALTER
```

```
-- Table: departamento
-- DROP TABLE departamento;
CREATE TABLE departamento
(
  id_departamento int4 NOT NULL DEFAULT
nextval('departamento_id_departamento_seq'::regclass),
  nome varchar,
  id_universidade int4 NOT NULL,
  id_endereco int4,
  CONSTRAINT departamento_pkey PRIMARY KEY (id_departamento),
  CONSTRAINT departamento_fk FOREIGN KEY (id_universidade)
      REFERENCES universidade (id_universidade) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT departamento_fk1 FOREIGN KEY (id_endereco)
      REFERENCES endereco (id_endereco) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT departamento_id_endereco_key UNIQUE (id_endereco)
)
WITH OIDS;
ALTER TABLE departamento OWNER TO "admin";
```

```
ALTER TABLE endereco OWNER TO "admin";
```

2. Configurar o Hibernate

Configuração do Ambiente

- Crie um diretório de trabalho chamado workspace;
- Crie um pacote chamado universidades. Escolha:
  - Add external JAR: importar os arquivos .jar de hibernate-3.1.3\hibernate-3.1\lib
  - Add external JAR: importar hibernate3.jar de hibernate-3.1.3\hibernate-3.1

Configuração do Ambiente

- Crie um pacote br.model.bean para os arquivos de persistência;
- Crie um pacote br.testes.hibernate onde ficarão os arquivos de teste;
- Crie um pacote br.testes.hibernate.persistence onde ficará o arquivo HibernateFactory.java;
- Importe postgresql-8.2dev-503.jdbc3.jar, que deve também ser baixado do site www.postegresql.org;
- O arquivo Hibernate.cfg.xml deverá ficar na raiz da aplicação.

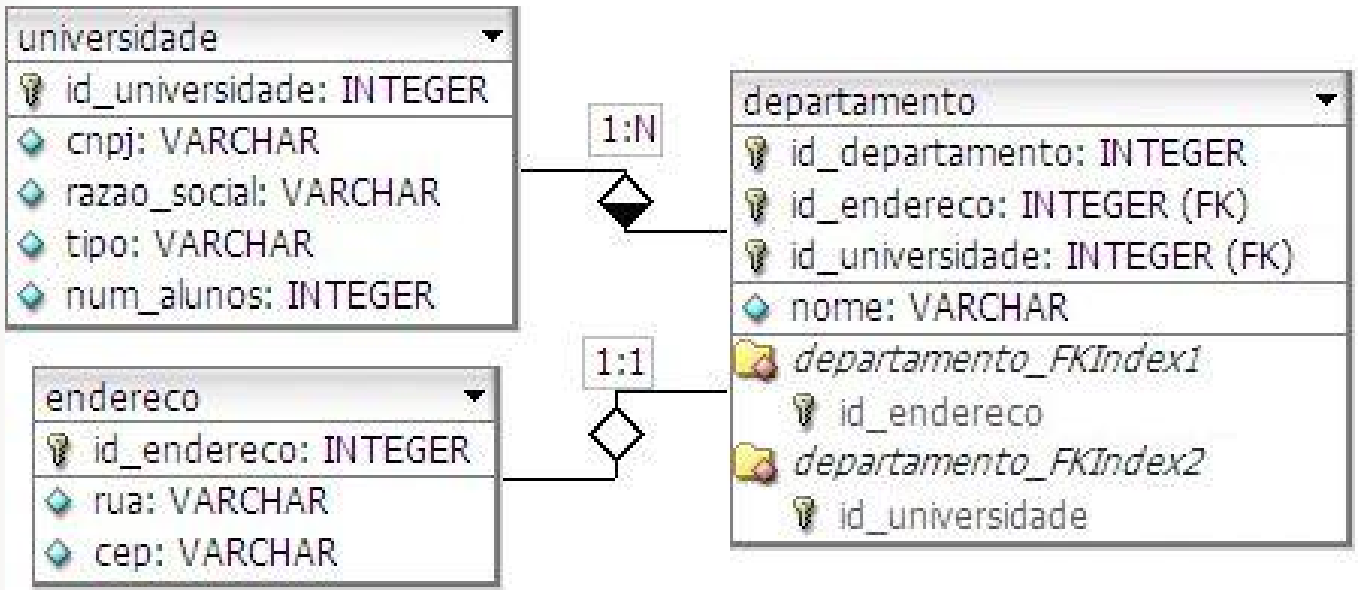
3. Criação dos Arquivos ORM

Arquivos ORM

- Para cada tabela é necessário a criação dos arquivos ORM;
  - O primeiro arquivo é uma classe de persistência JAVA que deve ter:
    - Um construtor padrão sem argumentos
    - Variáveis para cada atributo da tabela. Cada uma destas variáveis deve possuir os métodos get e set;
  - O segundo arquivo possui a extensão .hbm.xml, e faz o mapeamento entre os atributos das tabelas e as variáveis das classes de persistência JAVA;

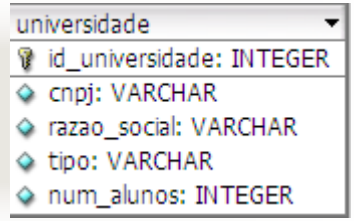
Arquivos ORM

- **Mapeamento dos atributos**
- Mapeamento dos relacionamentos



Mapeamento dos Atributos

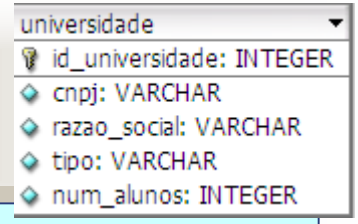
Universidade.java




```
package br.model.bean;

import java.util.HashSet;
import java.util.Set;

/**
 * Classe Universidade - Parte integrante do esquema necessario para a
 * utilizacao do Hibernate
 * Junto com universidade.hbm.xml faz o mapeamento da tabela universidade
 */
public class Universidade {

    /**
     * O mapeamento exige que para cada campo da tabela seja criada
     * uma variavel do tipo correspondente
     */
    private int id_universidade;
    private String cnpj;
    private String razao_social;
    private String tipo;
    private int num_alunos;
…
```

Mapeamento dos Atributos

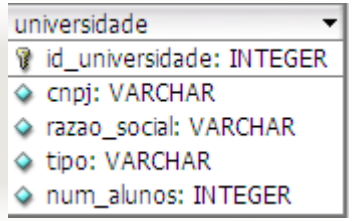
Universidade.java




```
...

    /** O mapeamento exige que para cada variavel existam os
     *  metodos get e set
     */
    public String getCnpj() {
            return cnpj;      }
    public void setCnpj(String cnpj) {
            this.cnpj = cnpj;}
    public int getId_universidade() {
            return id_universidade;   }
    public void setId_universidade(int id_universidade) {
            this.id_universidade = id_universidade;    }
    public String getRazao_social() {
            return razao_social;      }
    public void setRazao_social(String razao_social) {
            this.razao_social = razao_social;  }
    public String getTipo() {
            return tipo;      }
    public void setTipo(String tipo) {
            this.tipo = tipo;}
    public int getNum_alunos() {
            return num_alunos;        }
    public void setNum_alunos(int num_alunos) {
            this.num_alunos = num_alunos;      }
}
```

Mapeamento dos Atributos

Universidade.hbm.xml




```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="br.model.bean.Universidade" table="universidade">

        <id name="id_universidade"
            column="id_universidade"
            type="integer">
          <generator class="sequence">
              <param name="sequence">universidade_id_universidade_seq</param>
          </generator>
        </id>

        <property name="cnpj" type="string"/>
        <property name="razao_social" column="razao_social" type="string"/>
        <property name="tipo" type="string"/>
        <property name="num_alunos" column="num_alunos" type="integer"/>

    </class>
</hibernate-mapping>
```

Nome da tabela

Nome completo da classe

Seqüência criada pelo PostgreSQL

Descrição dos atributos da Tabela

Nome da variável na classe Java

Nome dos atributos na tabela

Tipo do dado

Mapeamento dos Atributos


```
package br.model.bean;

import br.model.bean.Universidade;
import br.model.bean.Endereco;

/**
 * Classe Departamento - Parte integrante do esquema necessario para a
 * utilizacao do Hibernate
 * Junto com departamento.hbm.xml faz o mapeamento da tabela departamento
 */
public class Departamento {
    /**
     * O mapeamento exige que para cada campo da tabela seja criada uma
     * variavel do tipo correspondente
     */
    private int id_departamento;
    private String nome;
    /** O mapeamento exige que para cada variavel existam os metodos get e
     *  set
     */
    public int getId_departamento() {
            return id_departamento;   }
    public void setId_departamento(int id_departamento) {
            this.id_departamento = id_departamento;    }
    public String getNome() {
            return nome;      }
    public void setNome(String nome) {
            this.nome = nome;          }
}
```

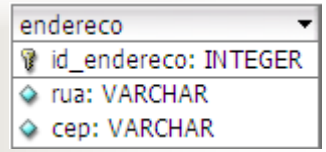
Mapeamento dos Atributos

Departamento.hbm.xml


```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

   <class name="br.model.bean.Departamento" table="departamento">

    <id name="id_departamento"
        column="id_departamento"
        type="integer">
        <generator class="sequence">
            <param name="sequence">departamento_id_departamento_seq</param>
        </generator>
    </id>

    <property name="nome" type="string"/>

    </class>
</hibernate-mapping>
```

Mapeamento dos Atributos

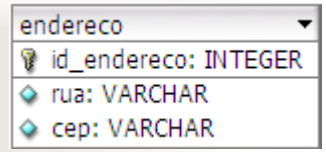
Endereco.java




```
package br.model.bean;

/**
 * Classe Endereco - Parte integrante do esquema necessario para a
 * utilizacao do Hibernate
 * Junto com endereco.hbm.xml faz o mapeamento da tabela endereco
 */
public class Endereco {
    /**
     * O mapeamento exige que para cada campo da tabela seja criada uma
     * variavel do tipo correspondente
     */
    private int id_endereco;
    private String rua;
    private String cep;

...
```

Mapeamento dos Atributos

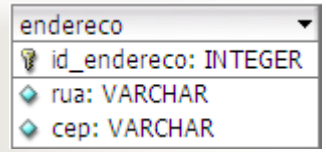
Endereco.java




```
...
    /**
     * O mapeamento exige que para cada variavel existam os metodos get
     *  e set
     */
    public int getId_endereco(){
            return id_endereco;       }
    public void setId_endereco(int id_endereco){
            this.id_endereco = id_endereco;    }
    public String getRua(){
            return rua;       }
    public void setRua(String rua){
            this.rua = rua;   }
    public String getCep(){
            return cep;       }
    public void setCep(String cep){
            this.cep = cep;   }
}
```

Mapeamento dos Atributos

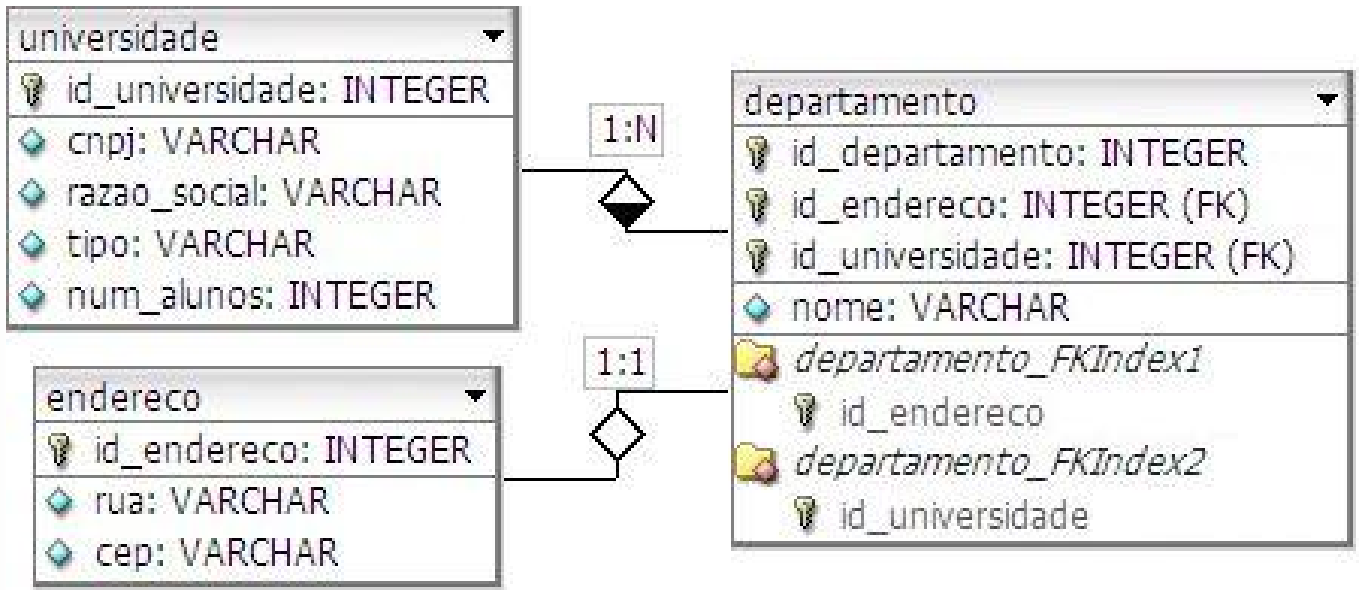
Endereco.hbm.xml




```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

        <class name="br.model.bean.Endereco" table="endereco">

           <id name="id_endereco"
                    column="id_endereco"
                    type="integer">
                 <generator class="sequence">
                          <param name="sequence">endereco_id_endereco_seq</param>
                 </generator>
           </id>

           <property name="rua" type="string"/>
           <property name="cep" type="string"/>

        </class>

</hibernate-mapping>
```

Arquivos ORM

- Mapeamento dos atributos
- **Mapeamento dos relacionamentos**



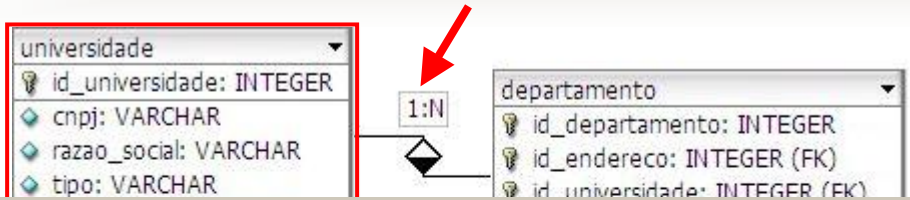
Mapeamento dos Relacionamentos

Universidade.java

```
package br.model.bean;

import java.util.HashSet;
import java.util.Set;

/**
 * Classe Universidade - Parte integrante do esquema necessario para a
 * utilizacao do Hibernate
 * Junto com universidade.hbm.xml faz o mapeamento da tabela universidade
 */
public class Universidade {

    /**
     * O mapeamento exige que para cada campo da tabela seja criada
     * uma variavel do tipo correspondente
     */
    private int id_universidade;
    private String cnpj;
    private String razao_social;
    private String tipo;
    private int num_alunos;
    private Set departamentos = new HashSet();
                //mapeia o relacionamento com a tabela departamento
    ...
```

Mapeamento dos Relacionamentos

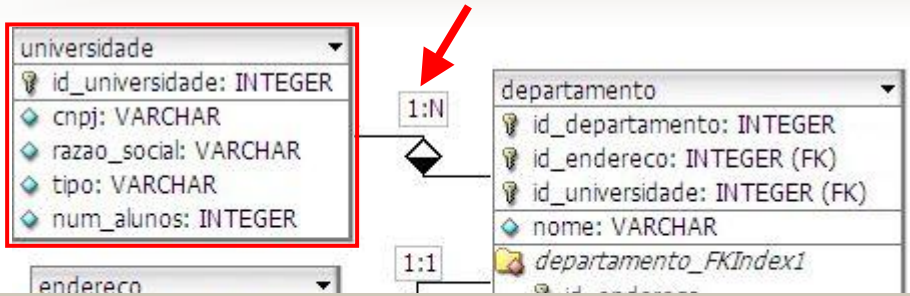
Universidade.java




```
...

    /**
     * O mapeamento exige que para cada variavel existam os metodos
     * get e set
     */
...

    /**
     * O mapeamento do relacionamento <1-para-N> com a tabela
     * departamento é feito atraves do conjunto de departamentos
     */
    public Set getDepartamentos(){
            return departamentos;     }
    public void setDepartamentos(Set departamentos){
            this.departamentos = departamentos;         }
    public void addDepartamentos(Departamento departamento){
            departamento.setId_universidade(this);
            departamentos.add(departamento);   }

}
```

Mapeamento dos Relacionamentos

Universidade.hbm.xml




```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="br.model.bean.Universidade" table="universidade">

...
        <!-- Mapeamento da Colecao de Departamentos -->
        <set name="departamentos" inverse="true" lazy="true">
            <key column="id_universidade" />
            <one-to-many class="br.model.bean.Departamento"/>
        </set>

    </class>
</hibernate-mapping>
```

O nome da coluna da chave estrangeira na classe Universidade

Informa a classe a qual pertence a coleção de objetos

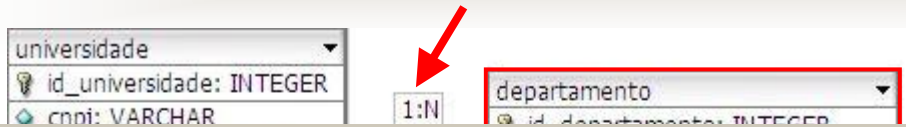
Mapeamento dos Relacionamentos

Departamento.java

```
package br.model.bean;

import br.model.bean.Universidade;
import br.model.bean.Endereco;

/**
 * Classe Departamento - Parte integrante do esquema necessario para a
 * utilizacao do Hibernate
 * Junto com departamento.hbm.xml faz o mapeamento da tabela departamento
 */
public class Departamento {
    /**
     * O mapeamento exige que para cada campo da tabela seja criada uma
     * variavel do tipo correspondente
     */
    private int id_departamento;
    private String nome;
    public Universidade id_universidade;
                // mapeia o relacionamento com a tabela universidade
    public Endereco id_endereco;
                // mapeia o relacionamento com a tabela endereco


    /* O mapeamento exige que para cada variavel existam os metodos get e
     * set
     */
    ...
```

Mapeamento dos Relacionamentos

Departamento.java




```
...
 /**
  * O mapeamento do relacionamento <N-para-1> com a tabela universidade
  * é feito atraves do objeto id_universidade da classe Universidade
  */
  public Universidade getId_universidade() {
            return id_universidade;
  }
  public void setId_universidade(Universidade id_universidade){
            this.id_universidade = id_universidade;
  }

/**
 * O mapeamento do relacionamento <1-para-1> com a tabela endereco
 * é feito atraves do objeto id_endereco da classe Endereco
 */
 public Endereco getId_endereco(){
            return id_endereco;
 }
 public void setId_endereco(Endereco id_endereco){
            this.id_endereco = id_endereco;
 }
}
```

Mapeamento dos Relacionamentos

Departamento.hbm.xml


```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

   <class name="br.model.bean.Departamento" table="departamento">
 …
     <!-- Mapeamento da Universidade -->
     <many-to-one name="id_universidade"
                  class="br.model.bean.Universidade"
                  column="id_universidade" />

     <!-- Mapeamento do Endereco -->
     <many-to-one name="id_endereco"
                  class="br.model.bean.Endereco"
                  column="id_endereco"
                  unique="true" />
   </class>

</hibernate-mapping>
```

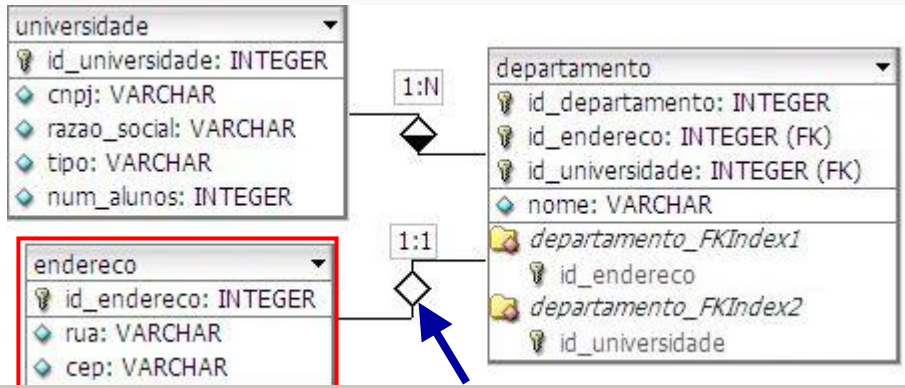
Nome do atributo na classe Java

Coluna do banco (chave estrangeira)

Não aceita valores repetidos

Mapeamento dos Relacionamentos

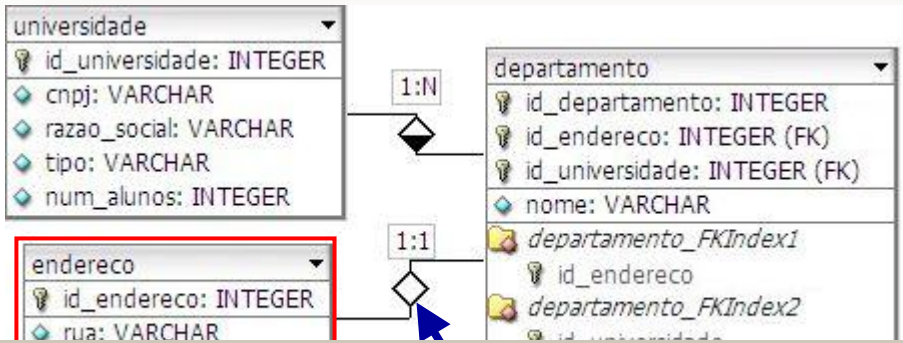
Endereco.java




```
package br.model.bean;

/**
 * Classe Endereco - Parte integrante do esquema necessario para a
 * utilizacao do Hibernate
 * Junto com endereco.hbm.xml faz o mapeamento da tabela endereco
 */
public class Endereco {
    /**
     * O mapeamento exige que para cada campo da tabela seja criada uma
     * variavel do tipo correspondente
     */
    private int id_endereco;
    private String rua;
    private String cep;

...
```

Mapeamento dos Relacionamentos

Endereco.java




```
...
    /**
     * O mapeamento exige que para cada variavel existam os metodos get
     *  e set
     */
    public int getId_endereco(){
            return id_endereco;       }
    public void setId_endereco(int id_endereco){
            this.id_endereco = id_endereco;    }
    public String getRua(){
            return rua;       }
    public void setRua(String rua){
            this.rua = rua;   }
    public String getCep(){
            return cep;       }
    public void setCep(String cep){
            this.cep = cep;   }
}
```

Mapeamento dos Relacionamentos

Endereco.hbm.xml


```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

        <class name="br.model.bean.Endereco" table="endereco">

           <id name="id_endereco"
                    column="id_endereco"
                    type="integer">
                 <generator class="sequence">
                          <param name="sequence">endereco_id_endereco_seq</param>
                 </generator>
           </id>

           <property name="rua" type="string"/>
           <property name="cep" type="string"/>

        </class>

</hibernate-mapping>
```

O mapeamento do relacionamento (1:1) com a tabela Departamento aparece apenas para a tabela que carrega a chave estrangeira.

4. Arquivos de Configuração

Configuração

Hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
    <property name="hibernate.dialect">
org.hibernate.dialect.PostgreSQLDialect</property>
    <property name="connection.driver_class">
            org.postgresql.Driver</property>
    <property name="connection.url">
jdbc:postgresql://localhost:5432/universidades</property>
       <property name="connection.username">admin</property>
    <property name="connection.password">admin</property>
    <property name="show_sql">true</property>

    <mapping resource="br/model/bean/universidade.hbm.xml"/>
    <mapping resource="br/model/bean/departamento.hbm.xml"/>
    <mapping resource="br/model/bean/endereco.hbm.xml"/>
    <mapping resource="br/model/bean/consultas.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

Dialeto SQL específico do BD utilizado

URL de conexão específica do BD

Nome da classe do driver JDBC do BD

Nome de usuário e senha com o qual o Hibernate deve se comunicar com o BD

Define se os SQLs gerados pelo Hibernate devem ou não ser exibidos

Arquivos de mapeamento

Configuração

```
package br.testes.hibernate.persistence;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateFactory {

    private static HibernateFactory factory;
    private static Exception exception;

    private SessionFactory sessionFactory;
    private Session session;

    static {
        try {
            factory = new HibernateFactory();
        } catch( Exception e ) {
            exception = e;
        }
    }

    public HibernateFactory() throws Exception {
        setSessionFactory( new
Configuration().configure().buildSessionFactory() );
        setSession( getSessionFactory().openSession() );
    }
...
```

Configuração

HibernateFactory.java

```
...
 public static HibernateFactory getInstance() throws Exception {
       if( factory != null ) {
           return factory;
       } else {
           Exception e = new Exception( exception );
            //"Error initializing hibernate.cfg.xml");
           throw e;
       }
   }

   public SessionFactory getSessionFactory() {
       return sessionFactory;
   }
   public void setSessionFactory(SessionFactory sessionFactory) {
       this.sessionFactory = sessionFactory;
   }
   public Session getSession() {
       return session;
   }
   public void setSession(Session session) {
       this.session = session;
   }
}
```

4. Testes

Inserção

```
package br.testes.hibernate;
import org.hibernate.Session;
import br.model.bean.Universidade;
import br.testes.hibernate.persistence.HibernateFactory;
import org.hibernate.Transaction;

/**
 * Classe UniversidadeTeste - Utiliza o Hibernate para persistir dados na tabela
 * universidade
 */
public class UniversidadeTeste {

     public void saveUniversidade( String umCnpj, String umaRazao_social,
                                   String umTipo, int umNum_alunos) {
         try {
             Session session = HibernateFactory.getInstance().getSession();
             Transaction tx = session.beginTransaction();
             Universidade universidade = new Universidade();
             universidade.setCnpj(umCnpj);
             universidade.setRazao_social(umaRazao_social);
             universidade.setTipo(umTipo);
             universidade.setNum_alunos(umNum_alunos);
             session.save( universidade );
             tx.commit();

         } catch (Exception e) { System.out.println( e ); }
     }
}
```

Inserção

DepartamentoTeste.java

```
package br.testes.hibernate;
import org.hibernate.Session;
import br.model.bean.Universidade;
import br.model.bean.Departamento;
import br.model.bean.Endereco;
import br.testes.hibernate.persistence.HibernateFactory;
import org.hibernate.Transaction;

/**
 * Classe DepartamentoTeste - Utiliza o Hibernate para persistir dados na tabela
 * departamento
 */
public class DepartamentoTeste {
     public void saveDepartamento( String umNome, Universidade umaUniversidade,
                                   Endereco umEndereco) {
         try {
             Session session = HibernateFactory.getInstance().getSession();
             Transaction tx = session.beginTransaction();
             Departamento departamento = new Departamento();
             departamento.setNome(umNome);
             departamento.setId_universidade(umaUniversidade);
             departamento.setId_endereco(umEndereco);
             session.save( departamento );
             tx.commit();
         } catch (Exception e) { System.out.println( e ); }
     }
}
```

EnderecoTeste.java

```
package br.testes.hibernate;
import org.hibernate.Session;
import br.model.bean.Endereco;
import br.testes.hibernate.persistence.HibernateFactory;
import org.hibernate.Transaction;

/**
 * Classe EnderecoTeste - Utiliza o Hibernate para persistir dados na tabela
 * endereco
 */
public class EnderecoTeste {

    public void saveEndereco( String umaRua, String umCep) {
         try {
            Session session = HibernateFactory.getInstance().getSession();
            Transaction tx = session.beginTransaction();
            Endereco endereco = new Endereco();
            endereco.setRua(umaRua);
            endereco.setCep(umCep);
            session.save( endereco );
            tx.commit();
        } catch (Exception e) { System.out.println( e ); }

    }
}
```

Teste de Inserção

Insert.java

```
package br.testes.hibernate;
import br.model.bean.Universidade;
import br.model.bean.Departamento;
import br.model.bean.Endereco;

/**
 * Classe Insert - Utiliza o Hibernate para persistir dados nas tabelas do
 * banco universidades
 */
 public class Insert {
    public static void main(String[] args) throws Exception {
         Universidade universidade = new Universidade();
         universidade.setId_universidade(2);

         Endereco endereco = new Endereco();
         endereco.setId_endereco(3);

         //UniversidadeTeste teste1 = new UniversidadeTeste();
         //teste1.saveUniversidade("1234", "UFRJ", "publica", 30000);

         //UniversidadeTeste teste6 = new UniversidadeTeste();
         //teste6.saveUniversidade("4321", "UERJ", "publica", 20000);

...
```

Teste de Inserção

Insert.java

```
...

        //UniversidadeTeste teste7 = new UniversidadeTeste();
        //teste7.saveUniversidade("111", "PUC Rio", "privada", 15000);

        //EnderecoTeste teste3 = new EnderecoTeste();
        //teste3.saveEndereco("Rua A", "22.290-000");

        //EnderecoTeste teste4 = new EnderecoTeste();
        //teste4.saveEndereco("Rua B", "22.290-000");

        //DepartamentoTeste teste2 = new DepartamentoTeste();
        //teste2.saveDepartamento("DCC", universidade, endereco);

        DepartamentoTeste teste5 = new DepartamentoTeste();
        teste5.saveDepartamento("MAT", universidade, endereco);

    }
}
```

Teste de Seleção

consultas.hbm.xml

```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<!-- Recupera todos as Universidades -->
<query name="Universidades">
<![CDATA[from Universidade uni
 order by uni.razao_social]]>
</query>

<!-- Recupera o nome da Universidade dado seu cnpj -->
<query name="UniversidadePorCnpj">
<![CDATA[from Universidade uni
 where uni.cnpj = :cnpj]]>
</query>

...
```

Teste de Seleção

consultas.hbm.xml

```
...

<!-- Recupera os departamentos de uma Universidade dado seu cnpj -->
<query name="DepartamentosPorCnpj">
<![CDATA[from Departamento dep, Universidade uni
 where dep.id_universidade = uni.id_universidade
    and uni.cnpj = :cnpj
 order by dep.nome]]>
</query>

<!-- Recupera o endereco de um departamentos dado seu nome e o cnpj da
    universidade ao qual pertence -->
<query name="EnderecoPorDepartamentoECnpj">
<![CDATA[from Endereco endereco, Departamento dep, Universidade uni
 where endereco.id_endereco = dep.id_endereco
    and dep.id_universidade = uni.id_universidade
    and uni.cnpj = :cnpj
    and dep.nome = :nome]]>

</query>


</hibernate-mapping>
```

Teste de Seleção

Select.java

```
package br.testes.hibernate;

import org.hibernate.Session;
import br.testes.hibernate.persistence.*;
import br.model.bean.Universidade;
import br.model.bean.Departamento;
import br.model.bean.Endereco;
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * Classe Select - Utiliza o Hibernate para selecionar dados nas tabelas do
 * banco universidades chamando as consultas nomeadas no arquivo
 * Consultas.hbm.xml
 */
public class Select {

    public static void main(String[] args) {
            try {
                Session session = HibernateFactory.getInstance().getSession();

                /* Consultas Nomeadas */
...

    } catch (Exception e) { System.out.println(e); }
}
```

Teste de Seleção

Select.java

```
...
        /* Selecione todas as Universidades ordenadas
           em ordem alfabética de sua razão social*/
        org.hibernate.Query query = session.getNamedQuery("Universidades");
        List result = query.list();
        ArrayList lista = new ArrayList();
        Iterator it = result.iterator();
        while( it.hasNext() ) {
                Universidade uni = (Universidade) it.next();
                lista.add(uni);
                System.out.println( "universidade: " + uni.getRazao_social());
        }

        /* Selecione a razão social da Universidade cujo cnoj = 1234*/
        org.hibernate.Query query =
         session.getNamedQuery("UniversidadePorCnpj").setString("cnpj", "1234");
        List result = query.list();
        ArrayList lista = new ArrayList();
        Iterator it = result.iterator();
        while( it.hasNext() ) {
                Universidade uni = (Universidade) it.next();
                lista.add(uni);
                System.out.println( "universidade: " + uni.getRazao_social());
        }
...
```

```
...
        /* Selecione o nome dos departamentos da universidade cujo cnpj = 1234*/
        org.hibernate.Query query =
         session.getNamedQuery("DepartamentosPorCnpj").setString("cnpj","1234");
        Iterator pairs = query.list().iterator();
        while( pairs.hasNext() ) {
                Object[] pair = (Object[]) pairs.next();
                Departamento dep = (Departamento) pair[0];
                Universidade uni = (Universidade) pair[1];
                System.out.println( "Universidade: " + uni.getRazao_social());
                System.out.println( "Departamento: " + dep.getNome());
        }

        /* Selecione o endereco do departamentos cujo nome = DCC e o cnpj da
           universidade ao qual pertence = 1234 */
        org.hibernate.Query query =
        session.getNamedQuery("EnderecoPorDepartamentoECnpj").setString("nome",
                              "DCC").setString("cnpj","1234");
        Iterator pairs = query.list().iterator();
        while( pairs.hasNext() ) {
                Object[] pair = (Object[]) pairs.next();
                Endereco endereco = (Endereco) pair[0];
                Departamento dep = (Departamento) pair[1];
                Universidade uni = (Universidade) pair[2];
                System.out.println( "Universidade: " + uni.getRazao_social());
                System.out.println( "Departamento: " + dep.getNome());
                System.out.println( "Endereco: " + endereco.getRua());
        }
...
```

Organização dos Arquivos

Organização dos Arquivos