

Introdução aos Frameworks

Conceitos Gerais

Um dos principais objetivos da Engenharia de Software é o reuso.

Através da reutilização de software obtém-se o **aumento da qualidade e redução do esforço de desenvolvimento**

A orientação a objetos fornece funcionalidades para que classes possam ser reutilizadas, bem como métodos por meio de seus mecanismos de herança e polimorfismo.

Componentes de software definem unidades reutilizáveis que oferecem serviços através de interfaces bem definidas.

Padrões de Projeto (*Design Patterns*) é outra abordagem mais abstrata que objetiva a reutilização de projetos de soluções para problemas recorrentes.

Além destas formas de reutilização, a tecnologia de frameworks possibilita que uma família de produtos seja gerada a partir de uma única estrutura que captura os conceitos mais gerais da família de aplicações.

Esta primeira aula apresenta os principais conceitos sobre frameworks.

Primeiramente, são revistas algumas definições sobre frameworks.

Logo após, os frameworks são classificados em duas categorias: frameworks de aplicação orientado a objetos e frameworks de componentes.

São apresentados os papéis envolvidos no desenvolvimento e uso de frameworks e os benefícios e desafios decorrentes da adoção de frameworks.

Classificação dos Frameworks

Frameworks podem ser classificados de diversas formas.

Inicialmente, são classificados em dois grupos principais:

- ☐ Frameworks de aplicação orientado a objetos
- ☐ Framework de componentes

Classificação quanto à forma usada para estendê-los

Além da classificação por escopo, frameworks podem ainda ser classificados quanto à forma usada para estendê-los.

Frameworks caixa branca

ou white box, são instanciados usando herança, através da implementação de Template Methods, métodos abstratos que são implementados na

subclasse, ou da redefinição de métodos ganchos, métodos com uma implementação padrão que pode ser redefinida na subclasse.

Frameworks caixa preta

ou black box, são instanciados através de configurações e composições, através da definição de classes que implementam uma determinada interface ou contrato.

Os pontos de extensão dos frameworks caixa preta freqüentemente são definidos seguindo o padrão de projetos Strategy

Frameworks caixa branca exigem que o desenvolvedor responsável pela instanciação do framework possua um bom conhecimento sobre sua estrutura interna e normalmente são mais difíceis de usar.

Por outro lado, frameworks caixa preta não exigem tanto conhecimento sobre a estrutura interna do framework, mas são menos flexíveis

Frameworks caixa cinza, ou gray box, possuem as características conjuntas dos frameworks caixa branca e preta, de forma a tentar evitar as desvantagens dos dois.

Papéis Envolvidos no Uso e Desenvolvimento de um Framework

Há vários profissionais envolvidos na criação, manutenção e uso (instanciação) de um framework.

Estes papéis, que não precisam necessariamente ser exercidos por pessoas diferentes, são: projetista do framework, mantenedor do framework e o desenvolvedor de aplicações (usuário do framework).

O projetista do framework é responsável pelo desenvolvimento da estrutura básica do framework.

O projetista determina os pontos de extensão do frameworks (hot spots) e realiza o levantamento de requisitos.

O mantenedor do framework é responsável por redefinir e acrescentar novas funcionalidades ao projeto do framework, possibilitando que novas características sejam incorporadas ao framework no momento da instanciação pelos desenvolvedores de aplicação.

O desenvolvedor de aplicações usa o framework.

Ele satisfaz os pontos de extensão para gerar a aplicação, instância do framework.

Benefícios Decorrentes da Utilização de Frameworks

Os principais benefícios decorrentes da utilização de frameworks, advêm da modularidade, reusabilidade, extensibilidade e inversão de controle que os frameworks proporcionam.

Frameworks encapsulam detalhes de implementação voláteis através de seus pontos de extensão, interfaces estáveis e bem definidas, aumentando a modularidade da aplicação.

Os locais de mudanças de projeto e implementação da aplicação construída usando o framework são localizados, diminuindo o esforço para entender e manter a aplicação.

Além disso, frameworks incentivam o reuso, pois capturam o conhecimento de desenvolvedores em determinado domínio ou aspecto de infra-estrutura ou de integração de middleware.

As interfaces do framework promovem pontos de extensão que as aplicações estendem gerando instâncias que compartilham as funcionalidades definidas no framework.

Desta forma, aumentam a produtividade dos desenvolvedores, pois o processo não começa do zero, a qualidade e a confiabilidade do produto final, pois idealmente as funcionalidades do framework já foram testadas em várias instâncias, e a interoperabilidade de software, pois as instâncias de uma mesma família compartilham características em comum.

Frameworks oferecem pontos de extensão explícitos que possibilitam aos desenvolvedores estenderem suas funcionalidades para gerar uma aplicação.

Os pontos de extensão desacoplam as interfaces estáveis do framework e o comportamento de um determinado domínio de aplicação das variações requeridas por uma determinada aplicação em um dado contexto.

Por fim, alguns frameworks apresentam inversão de controle.

Inversão de controle, também é conhecida como princípio de Hollywood :
“Don’t call us, we will call you”

Trata-se de transferir o controle da execução da aplicação para o framework, que chama a aplicação em determinados momentos como na ocorrência de um evento.

Através da IoC o framework controla quais métodos da aplicação e em que contexto eles serão chamados em decorrência de eventos, como eventos de janela, um pacote enviado para determinada porta, entre outros.

Principais Frameworks

Abaixo seguem alguns dos principais frameworks Java:

Hibernate (Persistência de Dados)

Framework de persistência de dados, que usa conceitos de banco de dados, além do mapeamento objeto-relacional (classes Java para tabelas de databases).

<http://www.hibernate.org>

Struts (J2EE) -

Um dos frameworks mais usados em ambientes corporativos para construção de aplicações web.

Usa o modelo MVC e caracterizado por uma camada de controle com uso de J2EE e XML.

<http://jakarta.apache.org/struts/>

JavaServer Faces (J2EE) -

Baseado em tecnologia de servlets e JSP, pode ser usado como uma opção ao Struts.

<http://java.sun.com/j2ee/javaxserverfaces/>

Spring (POA)

Framework baseado em orientação a aspectos.

Possibilidade de uso em conjuntos com outros frameworks MVC, como o Struts e JSF.

<http://www.springframework.org/>

JDO (Persistência de Dados)

Interface que provê uma camada de abstração aplicação - armazenamento de dados.

<http://www.java.sun.com/products/JDO>

JUnit (testes)

Talvez o mais usado framework Java, incluído em IDEs free ou comerciais.

Para testes unitários em geral.

<http://junit.org>

Cactus (testes)

Framework específico para testes unitários de aplicações J2EE.

<http://jakarta.apache.org/cactus/index.html>

Log4J (log)

Amplamente usado e útil para geração de logs.

<http://logging.apache.org/log4j/docs/>

Jakarta commons-log (log)

Semelhante ao Log4J, sob o selo da Jakarta.

<http://jakarta.apache.org/commons/logging/>

Ant (build e deploy)

Framework também amplamente divulgado da Jakarta para automatização de processos de construção, além de testes e distribuição.

<http://jakarta.apache.org/ant>

Jasper Report / iReport (geradores de relatório) - framework para geração de modo dinâmico de relatórios. Compatível com formatos xml, pdf e html.

<http://ireport.sourceforge.net/>

Introdução ao Hibernate

Introdução ao Hibernate

O Hibernate é um framework de persistencia para mapeamento objeto/relacional para Java e .NET.

Com o Hibernate, o programador reduz muito a programação de acesso a banco de dados, obtendo uma produtividade significativa no desenvolvimento do sistema.

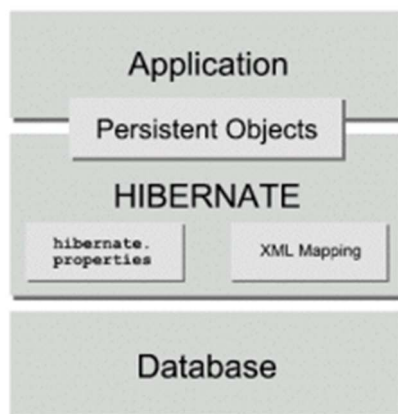
Este framework não é recomendavel para aplicações que utilizam stored procedures, triggers ou que implementam a maior parte da lógica da aplicação no banco de dados

O Hibernate é recomendado para aplicações onde a maior parte da lógica de negócios fica na própria aplicação, dependendo pouco de funções específicas do banco de Dados

Arquitetura do Hibernate

Visão Geral

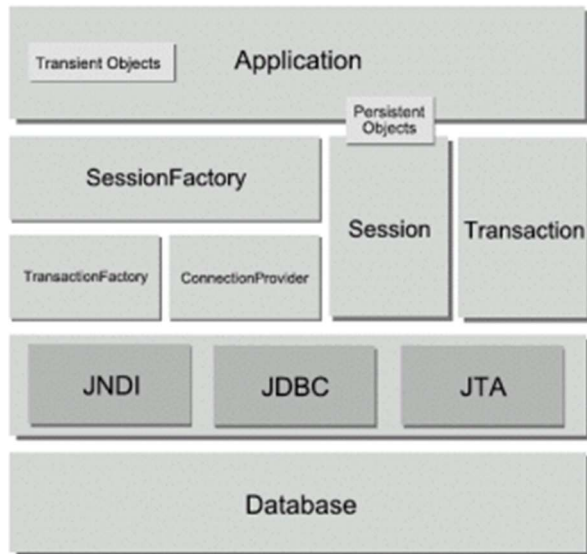
Uma visão bem ampla da arquitetura do Hibernate:



Esse diagrama mostra o Hibernate usando o banco de dados e a configuração de dados para prover persistência de serviços (e persistência de objetos) para o aplicativo.

Visão Detalhada

A arquitetura "completa" abstrai a aplicação de ter de lidar diretamente com JDBC/JTA e APIs e deixa o Hibernate tomar conta dos detalhes.



Algumas definições dos objetos do diagrama

SessionFactory

Um cache threadsafe (imutáveis) composto de identidades compiladas para um único banco de dados.

Uma fábrica para **Session** e um cliente de **ConnectionProvider**.

Pode conter um cachê opcional de dados (segundo nível) reutilizáveis entre transações, no nível de processo ou cluster.

Session

Objeto single-threaded, de vida curta, representando uma conversação entre o aplicativo e o armazenamento persistente.

Cria uma camada sobre uma conexão JDBC.

É uma fábrica de Transaction.

Possui um cachê obrigatório (primeiro nível) de objetos persistentes, usado para navegação no gráficos de objetos e pesquisa de objetos pelo identificador.

Objetos persistentes e coleções

Objetos, de vida curta, single threaded contendo estado persistente e função de negócios.

Esses podem ser JavaBeans/POJOs, onde única coisa especial sobre eles é que são associados a (exatamente uma) **Session**.

Quando a Session é fechada, eles são separados e liberados para serem usados dentro de qualquer camada da aplicação (Ex. diretamente como data transfer objects de e para a camada de apresentação)

Objetos e coleções desatachados e transientes

Instâncias de classes persistentes que ainda não estão associadas a uma **Session**.

Eles podem ter sido instanciados pela aplicação e não persistido (ainda) ou eles foram instanciados por uma **Session** que foi encerrada.

Transaction

Objeto opcional de vida curta, single threaded, usado pela aplicação para especificar unidades atômicas de trabalho.

Abstrai o aplicativo de lidar diretamente com transações JDBC, JTA ou CORBA.

Uma Session pode, em alguns casos, iniciar várias Transactions.

ConnectionProvider

Uma fábrica (Opcional) de (e combinações de) conexões JDBC.

Abstrai a aplicação de lidar diretamente com **Datasource** ou **DriverManager**

Não exposto para a aplicação, mas pode ser implementado ou estendido pelo programador.

TransactionFactory

Uma fábrica (Opcional) para instâncias de Transactions.

Não exposta a aplicação, mas pode ser estendida/ implementada pelo programador.

Extension Interfaces

O Hibernate oferece várias opções de interfaces estendidas que você pode implementar para customizar sua camada persistente.

Veja a documentação da API para maiores detalhes.

Estados de instância

Uma instância de uma classe persistentes pode estar em um dos três diferentes estados, que são definidos respeitando um *contexto persistente*.

O objeto Session do Hibernate é o contexto persistente:

Transiente

A instância não é, e nunca foi associada com nenhum contexto persistente. Não possui uma identidade persistente (valor da primary key).

Persistente

A instância está atualmente associada a um contexto persistente.

Possui uma identidade persistente (valor da primary key) e, talvez, correspondente a um registro no banco de dados.

Para um contexto persistente em particular, o Hibernate *garante* que a identidade persistente é equivalente a identidade Java (na localização em memória do objeto).

Desatachado

A instância foi associada com um contexto persistente, porém este contexto foi fechado, ou a instância foi serializada por outro processo.

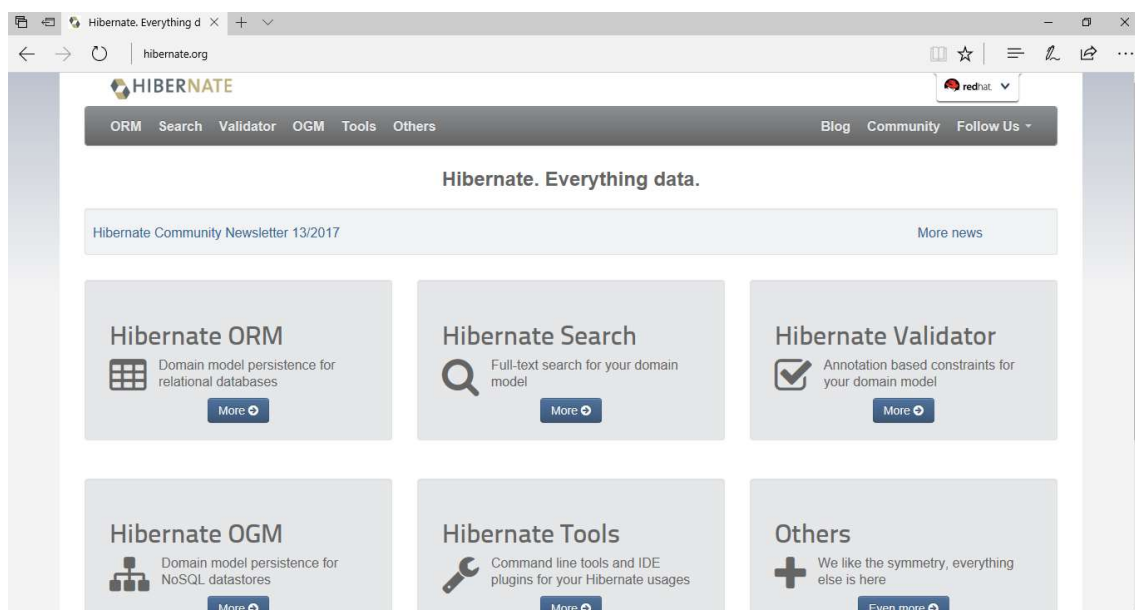
Possui uma identidade persistente, e, talvez, corresponda a um registro no banco de dados.

Para instâncias desatachadas, o Hibernate não garante o relacionamento entre identidade persistente e identidade Java.

Obtendo o Hibernate

Para obter o Hibernate visite o site:

<http://www.hibernate.org/>



Criando ao arquivo estilo.css

estilo.css

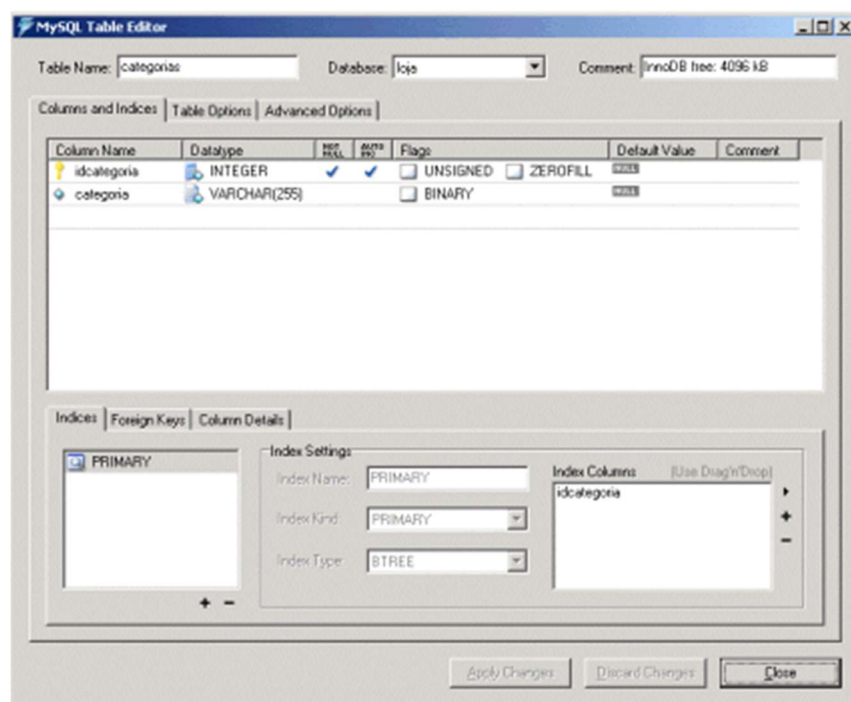
```
1 body{
2     font-size: xx-small;
3     margin: 0px;
4     font-family: Verdana;
5 }
6 TD{
7     font-size: xx-small;
8     margin: 0px;
9     font-family: Verdana;
10 }
11 H2{
12     color: blue;
13     font-family: Verdana;
14     font-size: x-small;
15 }
16 input{
17     font-size: xx-small;
18     margin: 0px;
19     font-family: Verdana;
20 }
```


Criando uma tabela no MySQL

Primeiro, nós iremos criar uma simples aplicação Hibernate uma base de dados MySQL, contendo uma simples tabela de categorias

```
DROP TABLE IF EXISTS loja.categorias;  
CREATE TABLE loja.categorias (  
  idcategoria int(11) NOT NULL auto increment,  
  categoria varchar(255) default NULL,  
  PRIMARY KEY (idcategoria)  
)
```

Verifique se a tabela foi criada corretamente



Inserindo alguns valores

Depois vamos inserir alguns valores para os testes iniciais

```
Insert into categorias (categoria) values ('Impressoras');  
Insert into categorias (categoria) values ('Monitores');  
Insert into categorias (categoria) values ('Periféricos');
```

Verificando se os dados foram gravados corretamente

idcategoria	categoria
1	Impressoras
2	Monitores
3	Periféricos

A primeira aplicação Hibernate

Criando a primeira aplicação com o Hibernate

Tabela categorias no MySQL



categorias
idcategoria: INTEGER
categoria: VARCHAR(50)

Neste primeiro exemplo o objetivo será criar uma aplicação web que altere os dados da categoria.

Para tanto é necessário 3 etapas fundamentais

- ☐ Criar a classe de persistencia
- ☐ Criar o arquivo de mapeamento
- ☐ Configurar o Hibernate

Classe de persistencia

Um Java Beans representando a tabela

Exemplo

Categorias.java

Arquivo de mapeamento

Um arquivo xml de mapeamento que informa ao Hibernate, qual tabela no banco de dados ele deverá acessar, e quais as colunas na tabela ele deverá usar.

Exemplo

Categorias.hbm.xml

Configurar o Hibernate

Um arquivo xml que contém informações de como se conectar ao banco de dados .

Também contém a lista de arquivos de mapeamento.

Exemplo

hibernate.cfg.xml

Classe de persistencia

Nossa primeira classe de persistência é uma simples classe JavaBean com algumas propriedades:

```
1 package br.ifmt.dai.loja;
2 public class Categorias {
3     private int idcategoria;
4     private String categoria;
5     public Categorias(){
6     }
7     public int getIdcategoria() {
8         return idcategoria;
9     }
10    public void setIdcategoria(int idcategoria) {
11        this.idcategoria = idcategoria;
12    }
13    public String getCategoria() {
14        return categoria;
15    }
16    public void setCategoria(String categoria) {
17        this.categoria = categoria;
18    }
19 }
```

Observações

Você pode ver que esta classe usa o padrão JavaBean convencional de nomes para os métodos getters e setters das propriedades, como também a visibilidade private dos campos.

Este é um padrão de projeto recomendado, mas não obrigatório.

O Hibernate pode também acessar campos diretamente, o benefício para os métodos de acesso é a robustez para o Refactoring.

O construtor sem argumento é obrigatório para instanciar um objeto desta classe por meio de reflexão.

A propriedade idcategoria mantém um valor único de identificação para um evento em particular.

Todas as classes persistentes de entidade (bem como aquelas classes dependentes de menos importância) precisam de uma propriedade de identificação, caso nós queiramos usar o conjunto completo de características do Hibernate.

De fato, a maioria das aplicações (em especial aplicações web) precisam distinguir os objetos pelo identificador, então você deverá considerar esta, uma característica em lugar de uma limitação.

Porém, nós normalmente não manipulamos a identidade de um objeto, consequentemente o método setter deverá ser privado.

O construtor sem argumentos é um item obrigatório para todas as classes persistentes;

O arquivo de mapeamento

O Hibernate precisa saber como carregar e armazenar objetos da classe de persistência.

Este é o ponto onde o arquivo de mapeamento do Hibernate entra em cena.

O arquivo de mapeamento informa ao Hibernate, qual tabela no banco de dados ele deverá acessar, e quais as colunas na tabela ele deverá usar.

A estrutura básica de um arquivo de mapeamento é parecida com o exemplo abaixo:

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
4 <hibernate-mapping>
5   <class name="br.ifmt.dai.loja.Categorias" table="categorias">
6     <id name="idcategoria" column="idcategoria" type="int">
7       <generator class="native"/>
8     </id>
9     <property name="categoria"/>
10  </class>
11 </hibernate-mapping>
12
```

Definindo a classe de persistência

Entre as duas tags `hibernate-mapping`, inclua um elemento `class`.

Todas as classes persistentes da entidade (novamente, poderá haver mais tarde, dependências entre as classes que não são classes-primárias de entidades) necessitam do tal mapeamento, para uma tabela no banco de dados SQL.

```
4 <hibernate-mapping>
5   <class name="br.ifmt.dai.loja.Categorias" table="categorias">
6     <id name="idcategoria" column="idcategoria" type="int">
7       <generator class="native"/>
8     </id>
9   </class>
10 </hibernate-mapping>
```

Agora, continuaremos com o mapeamento de uma única propriedade identificadora para as primary keys da tabela.

Definindo a chave primária

Além disso, nós não iremos nos importar com esta propriedade identificadora, nós iremos configurar uma estratégia de geração de id's para uma primary key de uma surrogate key:



```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
4 <hibernate-mapping>
5   <class name="br.ifmt.dai.loja.Categorias" table="categorias">
6     <id name="idcategoria" column="idcategoria" type="int">
7       <generator class="native"/>
8     </id>
9     <property name="categoria"/>
10  </class>
11 </hibernate-mapping>
12
```

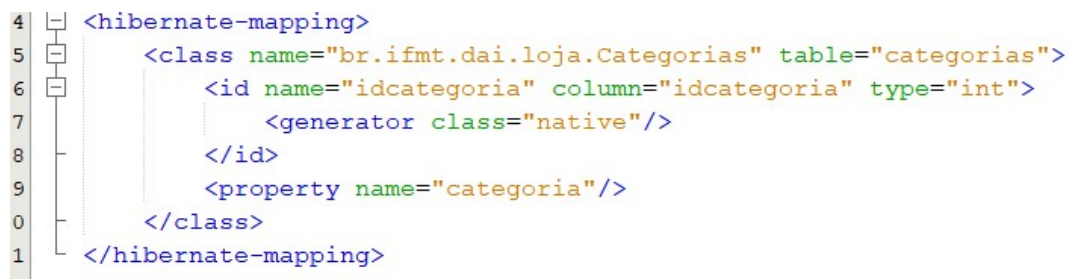
O elemento **id** é a declaração da propriedade identificadora, o **name="idcategoria"** declara o nome da propriedade Java o Hibernate irá usar os métodos **getter e setter** para acessar a propriedade. O atributo da coluna informa ao Hibernate qual coluna da tabela **categorias** nós iremos usar como primary key.

O elemento **generator** especifica a estratégia de geração do identificador, neste caso usaremos **native**, que escolhe a melhor estratégia dependendo do banco de dados (dialetos) configurado.

O Hibernate suporta identificadores gerados pelo banco de dados, identificadores únicos, é e claro, identificadores atribuídos na aplicação (ou qualquer outra estratégia através de uma extensão).

Definindo a chave primária

Além disso, nós não iremos nos importar com esta propriedade identificadora, nós iremos configurar uma estratégia de geração de id's para uma primary key de uma surrogate key:



```
4 <hibernate-mapping>
5   <class name="br.ifmt.dai.loja.Categorias" table="categorias">
6     <id name="idcategoria" column="idcategoria" type="int">
7       <generator class="native"/>
8     </id>
9     <property name="categoria"/>
10  </class>
11 </hibernate-mapping>
```

O elemento **id** é a declaração da propriedade identificadora, o **name="idcategoria"** declara o nome da propriedade Java o Hibernate irá usar os métodos **getter e setter** para acessar a propriedade. O atributo da coluna informa ao Hibernate qual coluna da tabela **categorias** nós iremos usar como primary key.

O elemento **generator** especifica a estratégia de geração do identificador, neste caso usaremos **native**, que escolhe a melhor estratégia dependendo do banco de dados (dialetto) configurado.

O Hibernate suporta identificadores gerados pelo banco de dados, identificadores únicos, é e claro, identificadores atribuídos na aplicação (ou qualquer outra estratégia através de uma extensão).

Configuração do Hibernate

Agora nós temos uma classe persistente e seu arquivo de mapeamento no lugar.

Está na hora de configurar o Hibernate.

O Hibernate é a camada da sua aplicação que se conecta com o banco de dados, para isso necessita de informação da conexão.

As conexões são feitas através de um pool de conexões JDBC, a qual teremos que configurar.

A distribuição do Hibernate contém diversas ferramentas de pooling da conexão JDBC open source, mas iremos usar o pool de conexão interna para este curso

Utilizando o arquivo de configuração XML



```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
3 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
4 <hibernate-configuration>
5   <session-factory>
6     <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
7     <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
8     <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/loja</property>
9     <property name="hibernate.connection.username">root</property>
10    <property name="hibernate.connection.password">ifmt2k17</property>
11    <property name="hibernate.connection.pool_size">1</property>
12    <property name="hibernate.current_session_context_class">thead</property>
13    <property name="hibernate.cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
14    <property name="hibernate.show_sql">>false</property>
15    <mapping resource="br/ifmt/dai/loja/Categorias.hbm.xml"/>
16  </session-factory>
17 </hibernate-configuration>
18
```

Nós configuraremos as **SessionFactory** do Hibernate uma factory global responsável por uma base de dados particular.

Se você tiver diversas bases de dados, use diversas configurações `<session-factory>`, geralmente em diversos arquivos de configuração (para um início mais fácil).

As primeiras quatro propriedades do elemento contêm a configuração necessária para a conexão ao JDBC.

A propriedade **dialect** do elemento especifica a variante particular do SQL que o Hibernate gera.

Finalmente, iremos adicionar os arquivos das classes de persistência mapeadas na configuração.

Criando uma página JSP

Através da página `salvarcategoria.jsp`, vamos gravar a alteração no banco de dados, sem a utilização de SQL

```
salvarecategoria.jsp
1 <%@page import="org.hibernate.jpa.boot.internal.EntityManagerFactoryBuilderImpl.MetadataSources"%>
2 <%@page contentType="text/html" pageEncoding="UTF-8"%>
3 <%@page import="org.hibernate.*"%>
4 <%@page import="org.hibernate.cfg.*"%>
5 <%@page import="org.hibernate.service.*"%>
6 <%@page import="br.ifmt.dai.loja.*"%>
7 <html>
8 <head>
9 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
10 <title>Primeiro exemplo do curso de J2EEA</title>
11 </head>
12 <body>
13 <h2>Primeiro exemplo do curso de Hibernate</h2>
14 <%
15     int idcategoria=0;
16     String categoria="";
17     Configuration conf=new Configuration();
18     if ((request.getParameter("idcategoria")!=null)&&(request.getParameter("categoria")!=null)){
19         try{
20             idcategoria=Integer.parseInt(request.getParameter("idcategoria"));
21             categoria=request.getParameter("categoria");
22             // ServiceRegistry sr=new ServiceRegistryBuilder().applySettings(
23             // conf.getProperties()).build();
24             SessionFactory sf;
25             // sf=conf.configure().buildSessionFactory(sr);
26             sf=new Configuration().configure("br/ifmt/loja/hibernate.cfg.xml").buildSessionFactory();
27             Session s=sf.openSession();
28             Transaction tx=s.beginTransaction();
29             Categorias c=new Categorias();
30             c.setIdcategoria(idcategoria);
31             c.setCategoria(categoria);
32             s.saveOrUpdate(c);
33             tx.commit();
34             s.close();
35             out.println("Gravou");
36         }catch(Exception x){
37             out.println(x.getMessage());
38         }
39     }else{
40         out.println("Não gravou");
41     }
42 %>
43 </body>
44 </html>
```

Primeiro foi criado um objeto do tipo `SessionFactory` (ou fabrica de sessões), a partir do `hibernate.cfg.xml`

```
24 SessionFactory sf;
25 // sf=conf.configure().buildSessionFactory(sr);
26 sf=new Configuration().configure("br/ifmt/loja/hibernate.cfg.xml").buildSessionFactory();
```

Depois foi criado um objeto que representa uma sessão a partir do `SessionFactory`

```
Session s = sf.openSession();
```

A partir do session podemos iniciar uma transação e executar uma ação no banco de dados

```
Categorias c = new Categorias();  
c.setIdcategoria(idcategoria);  
c.setCategoria(categoria);  
s.saveOrUpdate(c);
```

Principais métodos da Session

Métodos que podem ser utilizados a partir do objeto **Session**.

save(Object)

Inclui um objeto em uma tabela do banco de dados.

saveOrUpdate(Object)

Inclui um objeto na tabela caso ele ainda não exista (seja transiente) ou atualiza o objeto caso ele já exista (seja persistente).

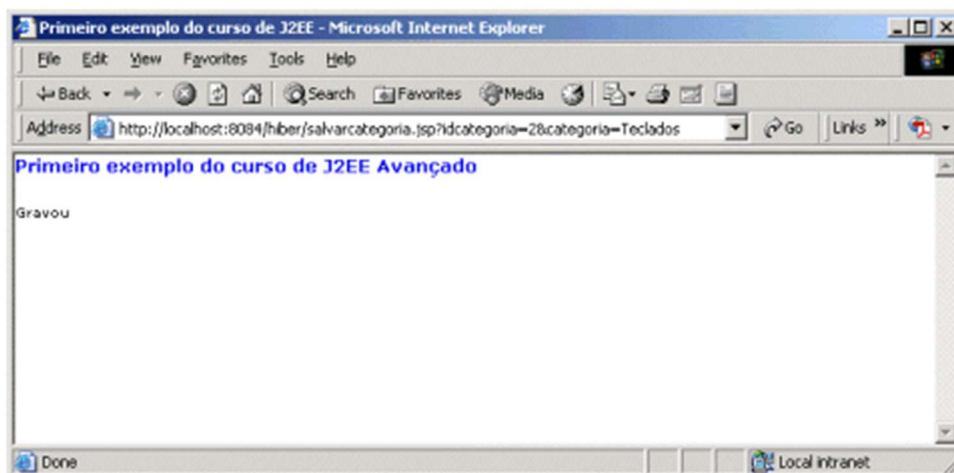
delete(Object)

Apaga um objeto da tabela no banco de dados

get(Class, Serializable id)

Retorna um objeto a partir de sua chave primária. A classe do objeto é passada como primeiro argumento e o seu identificador como segundo argumento.

Resultado da página



Avaliação da Aula

Baseado na tabela alunos.

Conforme estrutura abaixo

```
CREATE TABLE `loja`.`alunos` (  
  `idaluno` int(10) unsigned NOT NULL auto_increment,  
  `nome` varchar(45) NOT NULL,  
  `endereco` varchar(45) NOT NULL,  
  `cep` varchar(10) NOT NULL,  
  `cidade` varchar(45) NOT NULL,  
  `estado` varchar(2) NOT NULL,  
  `pais` varchar(45) NOT NULL,  
  PRIMARY KEY (`idaluno`)
```

1- Criar a classe de persistencia

Um Java Beans representando a tabela

Alunos.java

2- Criar Arquivo de mapeamento

Um arquivo xml de mapeamento que informa ao Hibernate, qual tabela no banco de dados ele deverá acessar, e quais as colunas na tabela ele deverá usar.

Alunos.hbm.xml

3- Configurar o Hibernate

Adicionar ao mesmo na lista de arquivos de mapeamento.

hibernate.cfg.xml

4- Criar um formulário para a entrada de dados dos alunos (apenas com um botão incluir)

Este formulario **deverá chamar um Servlet**, que através do Hibernate vai incluir no banco de dados

IncluirAlunos.java