**FACULDADES SPEI**

**TRABALHO MULTI-DISCIPLINAR  
ARTIGO TÉCNICO (JAVA HIBERNATE)**

**CURITIBA**

**16/05/2016**

**DOUGLAS TOMACHESKI DE ABREU E SILVA**

**GERSON LEONEL**

**TRABALHO MULTI-DISCIPLINAR**

**ARTIGO TÉCNICO** (JAVA HIBERNATE)

Trabalho realizado no Quinto Período do curso de Superior em Técnico de Análise e Desenvolvimento de Sistemas das FACULDADES SPEI. Supervisionado pelos professores Jean Marcelo da Costa Sales, Ivan Marcelo Pagnoncelli e Paulo Cezar de Oliveira.

**CURITIBA**

**16/05/2016**

Conteúdo

[INTRODUÇÃO 2](#_Toc451257151)

[1 O QUE É JAVA HIBERNATE? 3](#_Toc451257152)

[2 PARA QUE SERVE? 3](#_Toc451257153)

[3 COMO IMPLEMENTAR NO CÓDIGO? 3](#_Toc451257154)

[4 CONFIGURANDO O XML 5](#_Toc451257155)

[5 MAPEANDO E ASSOCIANDO 6](#_Toc451257156)

[6 HERANÇAS 7](#_Toc451257157)

[7 CONSULTAS AO BANCO DE DADOS 7](#_Toc451257158)

[CONCLUSÃO 9](#_Toc451257159)

[REFERÊNCIAS 10](#_Toc451257160)

# INTRODUÇÃO

Este artigo vai tratar a respeito da tecnologia do Java Hibernate, para o desenvolvimento do Trabalho Multi-Disciplinar LANGLEY, Simulação de Empréstimos. Feito sobre a orientação das Faculdades SPEI dos professores Jean Marcelo Costa Sales, Ivan Marcelo Pagnoncelli e Paulo Cezar de Oliveira.

O objetivo geral deste trabalho é trazer um conhecimento geral a respeito da tecnologia Hibernate, desde seu inicio até seus métodos de implementação.

O trabalho está dividido em 7 capítulos, o primeiro tratando sobre o Hibernate, o segundo explicando sua utilidade, o terceiro mostrando técnicas de importação e criação de projetos com ele, o quarto configurando ele com o que será criado no projeto, o quinto tratando sobre mapeamento das entidades, o sexto explicando como ele trabalha com heranças e o sétimo demonstrando as consultas que podem ser realizadas através dele.

A metodologia utilizada para a realização do artigo foi por desenvolvimento e implementação de código e pesquisas em páginas WEB.

# O QUE É JAVA HIBERNATE?

Hibernate, assim como o JPA (Java Persistence API) é uma framework disponível para a linguagem de programação orientada a objeto Java (existindo uma versão para .Net) usada para mapear um banco de dados com os modelos de objetos de uma aplicação, pelo uso de arquivos de configuração XML ou por anotações Java (Java Annotations).

Criado em 2001 por Gavin King, um astuto programador Java nesta época (e posterior mente criador do Seam e do Ceylon), foi adquirido pela Red Hat, mais especificamente a JBoss Inc, que contratou seus desenvolvedores para continuar a desenvolver o Hibernate e dar o seu suporte até hoje.

# PARA QUE SERVE?

O Hibernate tem como principal objetivo deixar a relação entre os apps em Java e os bancos de dados relacionais mais simples, especialmente nas operações CRUD (Create, Read, Update and Delete) que são cruciais para qualquer gerenciador de banco de dados.

Assim como o JPA, ele trabalha sobre o comportamento de persistência de POJO (Plain Old Java Object) e suas classes ficam configuradas pelo XML que acompanha a implementação do Hibernate no programa. Este XML é o que vai dar todo o embasamento entre a Aplicação e a Base de dados através do Hibernate.

# COMO IMPLEMENTAR NO CÓDIGO?

Para se implementar o Hibernate, é necessário o uso de um ambiente de desenvolvimento Java (uma IDE como o NetBeans ou Eclipse), os arquivos base do Hibernate contendo os JARs (arquivos compilados em Java) e um arquivo JAR de Driver JDBC (Banco de Dados Java) correspondente com um tipo de banco SQL para armazenar os dados (MySQL tem um driver, SQL tem outro, é necessário adquirir o correspondente ao qual você tenha instalado). Na IDE, ao criar um novo projeto, o desenvolvedor deve importar nos parâmetros de criação os JARs do Hibernate e do JDBC para que este fique incluso no programa. Após isso, o Hibernate irá gerar os códigos no SQL correspondente ao Driver adquirido.

No código, para utilizar o Hibernate (após configurado), será necessário criar uma sessão através de uma Session (importando corretamente a biblioteca do Session), como mostra o exemplo a seguir:

**SessionFactory session = new Configuration().configure().buildSessionFactory();**

**Session exemplo = session.openSession();**

**Transaction tran = session.beginTransaction();**

Client cliente = new Cliente();

cliente.setNome(“Douglas”);

cliente.setEndereco(“Rua Teste”);

cliente.setEmail(“email@email.com”);

**session.save(cliente);**

**tran.commit();**

**session.close();**

O uso do Transaction é para realizar a captura dos dados para que estes sejam registrados no banco. Tanto o Transaction quanto o Session devem ser finalizados no código após o uso.

É possível utilizar o Hibernate com outras APIs e frameworks similares a fim de uma profundidade maior no desenvolvimento das aplicações Java. Um dos exemplos mais recorrentes é o uso dele composto com o JPA, por causa do excelente funcionamento do Hibernate com a persistência de objetos (mencionada anteriormente).

# CONFIGURANDO O XML

Junto com a criação do programa na configuração, o XML vem junto pré-configurado de acordo com os parâmetros passados na criação do projeto no começo. Mas ele não registra os objetos criados automaticamente, então estes devem ser adicionados posteriormente ao XML definindo a classe à sua tabela no banco e definir cada uma de suas propriedades da seguinte maneira (utilizando o exemplo anterior como base)

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

<!DOCTYPE hibernate-mapping PUBLIC "

-//Hibernate/Hibernate Mapping DTD 2.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>

<class name="Cliente" table="CLIENTE">

<id column="CLIENTE\_ID" name="id" type="java.lang.Long">

<generator class="sequence"/>

</id>

<property

column="NOME"

length="15"

name="nome"

not-null="true"

type="java.lang.String"

/>

<property

column="ENDERECO"

name="endereco"

length="50"

not-null="true"

type="java.lang.String"

/>

<property

column="EMAIL"

name="email"

length="30"

not-null="true"

type="java.lang.String"

/>

</class>

</hibernate-mapping>

Ao configurar as classes, é necessário criar o XML de configuração do Hibernate para estabelecer a conexão entre a aplicação e o banco, seguindo o mesmo exemplo:

<!DOCTYPE hibernate-configuration PUBLIC"

-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLMyISAMDialect</property>

<property name="hibernate.connection.driver\_class">org.gjt.mm.mysql.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/exemplo</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">r00Tp@$wd</property>

<mapping-resource="Cliente.hbm.xml"/>

</session-factory>

</hibernate-configuration>

# MAPEANDO E ASSOCIANDO

Ao desenvolver uma base de dados conectado à uma aplicação, um bom analista de banco de dados terá que sempre mapear as relações entre as entidades do banco. Os mapeamentos se dão entre as relações de “*tantos-para-tantos*” entre um relacionamento de duas entidades. No Hibernate, estes mapeamentos ficam implementados nos XML com os atributos *<many-to-one>* e *<one-to-one>* da seguinte forma.

Exemplo de Cliente -> Dependentes

<many-to-one name="cliente"

column="CLIENTE\_ID"

class="Cliente"

not-null="true"/>

Mapeamento de Cliente -> Usuário

<one-to-one name="usuario" class="Usuario"/>

Ele também trabalha com mapeamento de coleções Java, para isso deve-se utilizar a importação correta da coleção que você vai utilizar. As coleções são mapeadas no XML utilizando os elementos <set>, <list>, <map>, <bag>, <array> ou <primitive-array>, sendo um pra cada coleção existente.

# HERANÇAS

O Java Hibernate trabalha com uma característica muito importante de POO, a herança de classes e objetos. Ele suportaaté três métodos de mapeamentos de herança: uma tabela por hierarquia de classes, uma por subclasse e uma por polimorfismo. No XML, esse mapeamento é feito usando as tags *<discriminator>* indicando a classe principal e *<subclass>* para as subclasses derivadas da principal.

Utilizando um exemplo novo: A classe Pessoa tem subclasses Cliente, Fornecedor e Funcionário:

<class name="Pessoa" table="PESSOA">

<id name="id" type="long" column="PESSOA\_ID">

<generator class="native"/>

</id>

<**discriminator** column="TIPO\_PESSOA" type="string"/>

<property name="nome" column="NOME"/>

...

<**subclass** name="Cliente" discriminator-value="CLT">

...

</subclass>

<**subclass** name="Funcionario" discriminator-value="FCN">

...

</subclass>

<**subclass** name="Fornecedor" discriminator-value="FRC">

...

</subclass>

</class>

# CONSULTAS AO BANCO DE DADOS

Para se trabalhar com consultas no Hibernate. É possível utilizar três métodos de implementação: Usando HQL (Hibernate Query Language), Usando Criteria Queries ou SQL Nativo.

Em HQL, ela segue a formula do SQL, mas direcionado para uso de Orientação à Objetos, possibilitando descrever consultas com polimofismos e sobre coleções.

Um exemplo abaixo, quando implementado corretamente, vai retornar todos os objetos da classe pessoa e as subclasses que tenham o nome igual MARIA:

from Pessoa pessoa where upper(pessoa.nome) like 'MARIA%'

////

Query q = session.createQuery("from Pessoa pessoa where upper(pessoa.nome) like :NOME");

q.setProperties(nome);

List pessoas = q.list();

Na API Criteria Queries, ela é intuitiva e extendível para a execução de consultas. Para implementar ela, é necessário criar um objeto Criteria na aplicação e nele definir os critérios de consulta, mostrado aqui neste exemplo, onde a consulta deve retornar todas as classes de nome MARIA e de idades entre 18 e 40 anos:

Criteria consulta = sess.createCriteria(Pessoa.class);

consulta.add( Expression.like("nome", "Maria%") );

consulta.add( Expression.between("idade", 18, 40) );

List resultado = consulta.list();

# CONCLUSÃO

Neste artigo, abordamos sobre a tecnologia de persistência Hibernate, no âmbito técnico desta tecnologia com suas sintaxes, métodos, comportamentos, implementações e configurações.

Foi um trabalho muito relevante para o projeto pois o mesmo conseguiu dar uma visão de projeto muito mais fácil e ampla sobre o assunto e a conexão entre a aplicação e o banco de dados.

# REFERÊNCIAS

**UFCG, PERSISTÊNCIA USANDO HIBERNATE.**

Disponível em:

<http://www.dsc.ufcg.edu.br/~jacques/cursos/daca/html/hibernate/hibernate.htm>

Acessado em: 14 de Maio de 2017

**PARTILHO, Java e Hibernate – Aula de Criar Aplicação no NetBeans.**

Disponível em:

< http://partilho.com.br/java-netbeans/java-e-hibernate-netbeans/>

Acessado em: 14 de Maio de 2017

**CAELUM, Uma introdução prática ao JPA com Hibernate.**

Disponível em:

<https://www.caelum.com.br/apostila-java-web/uma-introducao-pratica-ao-jpa-com-hibernate/>

Acessado em: 15 de Maio de 2017

**Alex Blewitt - INFOQ, Gavin King, criador do Hibernate e do Seam, apresenta a linguagem Ceylon.**

Disponível em:

< http://www.infoq.com/br/news/2011/04/ceylon/>

Acessado em: 15 de Maio de 2017