



Treinamento de hibernate.

Versão 0.2

Histórico de Revisões

Nome	Alterações	Data	Versão
Hélio Kárum de Oliveria Bastos	Criação do Documento	29/10/12	0.1

Hélio Kárum de Oliveria Bastos	Finalização do Documento	02/11/12	0.2

Sumário

1 Introdução	4
2 Instalação	4
2.1 Instalação	4
2.1.1 Obtendo arquivos	4
2 Instalação	4
2.1 Obtendo Arquivos	4
2.1.1 Configurando o NetBeans	5
3 Treinamento Hibernate	6
3.1 Oque é Hibernate	7
3.1.1 Vantagens do Hibernate	7

3.2 Modelagem de dados	8
3.2.1 Mapeamento - Annotations	8
3.2.2 Relações Complexas	10
3.2.2.1 Many to One	10
3.2.2.2 Many to On Many	14
3.2.2.2 Tipo de Fecth	16
3.3 Data Access Object(DAO)	16
3.4 Arquivo de Configuração	18

1 Introdução

Este Documento tem com finalidade, um tutorial sobre Hibernate, visando a qualificação da equipe de desenvolvimento do Sistema de Gestão Bibliográfica.

2 Instalação

Neste Tópico visamos a obter as pendencias necessárias para a utilização do framework Hibernate.

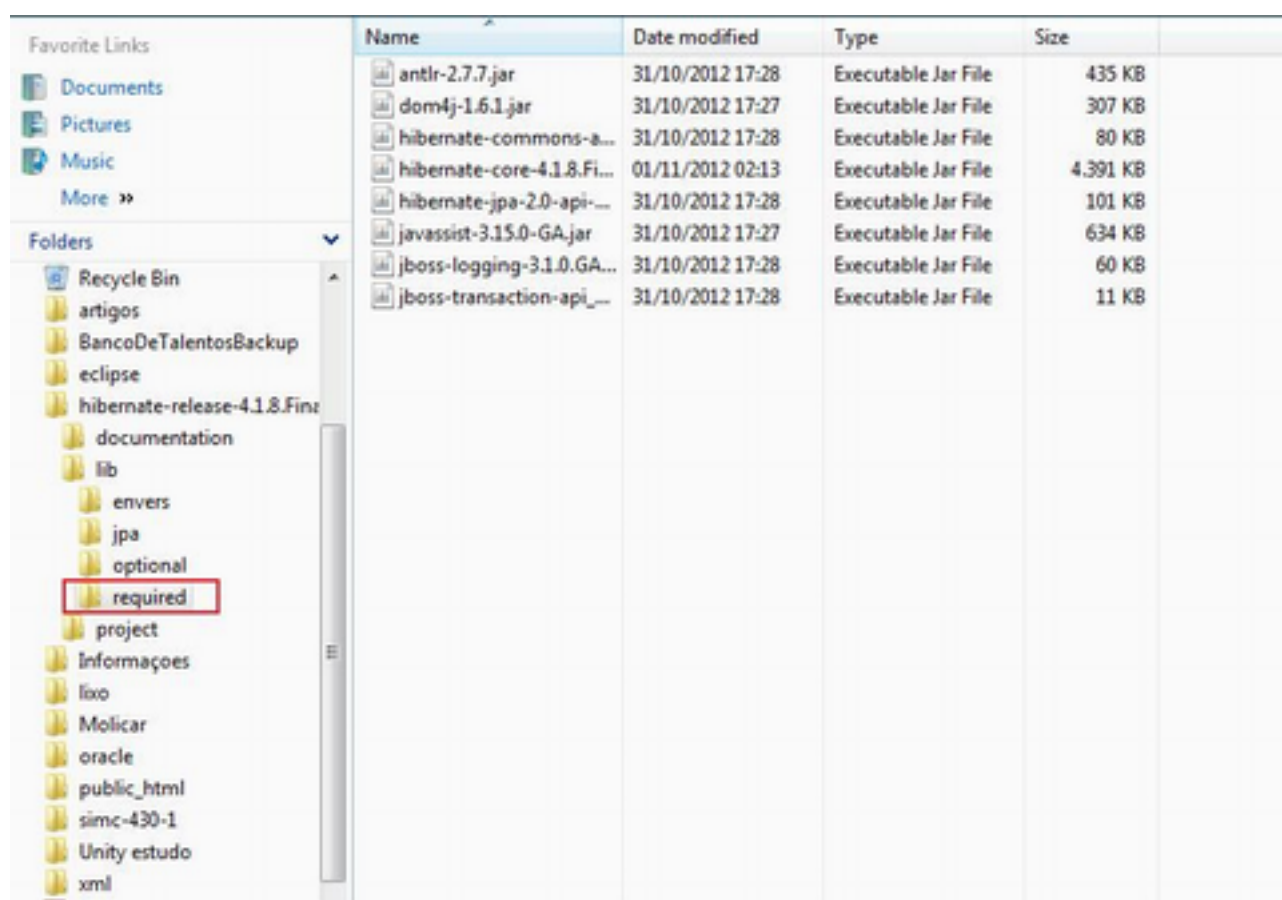
2.1 Obtendo arquivos.

Precisamos baixar todas as dependências que o Hibernate precisa para podermos trabalhar com o Framework.

Vamos ao site oficial do Hibernate e baixar o arquivo mais recente disponibilizado pelo distribuidor o site: <http://sourceforge.net/projects/hibernate/files/hibernate4/> contém todos os Releases.

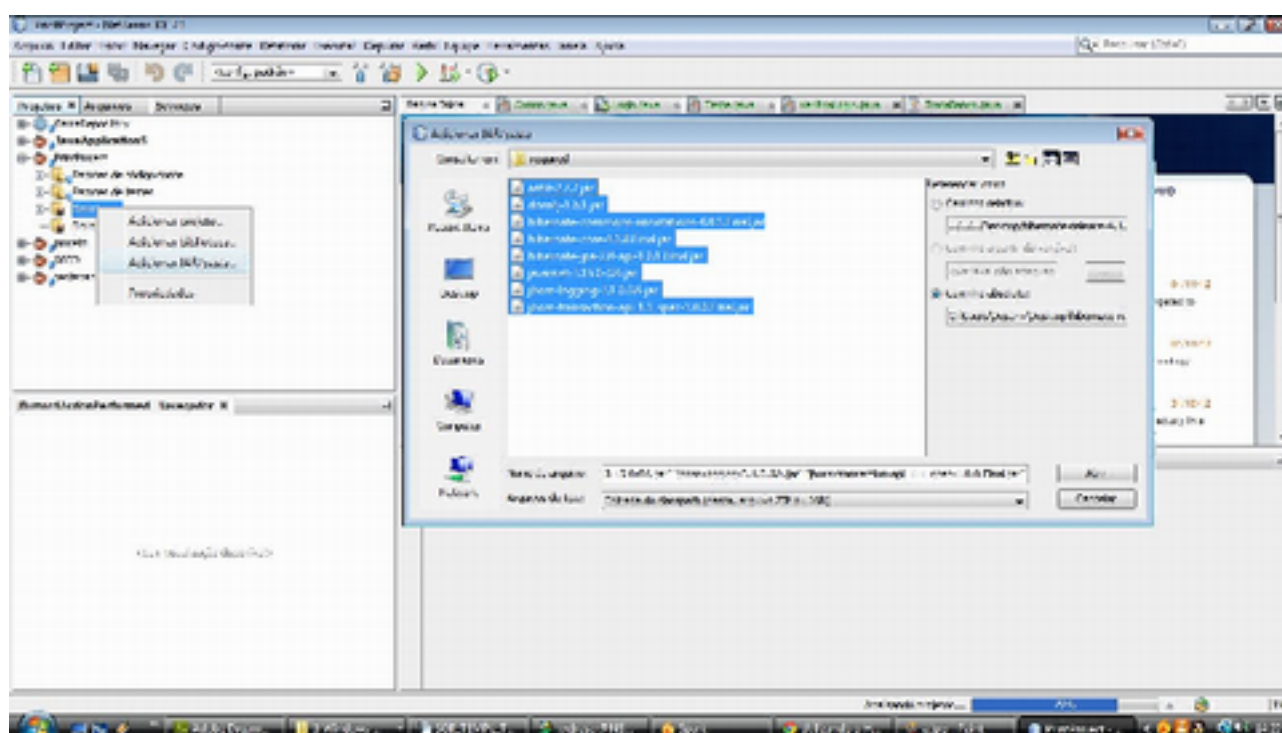
2.1.1 Configurando o NetBeans.

Após baixar o ultimo Release do Hibernate. Deparamos com um arquivo Zip que contém todos os arquivos necessários. Os arquivos Necessários estão contidos na pasta lib/required Como mostra a figura a baixo.



Name	Date modified	Type	Size
antlr-2.7.7.jar	31/10/2012 17:28	Executable Jar File	435 KB
dom4j-1.6.1.jar	31/10/2012 17:27	Executable Jar File	307 KB
hibernate-commons-a...	31/10/2012 17:28	Executable Jar File	80 KB
hibernate-core-4.1.8.Fi...	01/11/2012 02:13	Executable Jar File	4.391 KB
hibernate-jpa-2.0-api-...	31/10/2012 17:28	Executable Jar File	101 KB
javassist-3.15.0-GA.jar	31/10/2012 17:27	Executable Jar File	634 KB
jboss-logging-3.1.0.GA...	31/10/2012 17:28	Executable Jar File	60 KB
jboss-transaction-api-...	31/10/2012 17:28	Executable Jar File	11 KB

Abra o NetBeans, na Pasta Raiz do seu Projeto principal, na pasta biblioteca com o botão direito do mouse – Adicionar Pasta/Jar, e selecione onde estão os arquivos do Hibernate, como mostra a figura a baixo.



3 Treinamento – Hibernate.

3.1 O que é o Hibernate.

Hibernate é um framework de mapeamento relacional, que atua na camada de persistencia, permite a utilização de banco de dados relacional, porém, trabalhando com objetos.

3.1.1 Vantagens do Hibernate.

- **Controle transacional:** Permitir rollback quando ocorrer algum erro e commit quando o objeto for persistido com sucesso.
- **Identificadores de objeto:** Cada Objeto possui um Identificador.
- **Mapeamento O-R:** Permite mapear um banco se ele não for um banco orientado a objeto.
- **Cache:** Para melhoria de Desempenho os objetos são colocados em um cache, para a posteriori serem reutilizados.
- **Consultas sob demanda:** Objetos somente são instanciados quando são chamados, ou quando são necessários.
- **Proxy:** É um objeto proxy de forma que o usuário e o serviço de persistência conseguem identificar. Assim quando um objeto for carregado apenas as informações definidas pelo objeto proxy serão recuperadas.
- **Queries:** Pode-se utilizar a linguagem SQL comum para obtenção de dados..
- **Portabilidade:** É compatível com vários tipos de bancos, sua aplicação pode mudar de banco em somente uma linha.

3.2 . Modelagem de Dados.

Vamos começar com a criação de uma tabela. Veremos como Podemos criar uma tabela no banco a partir de uma classe JAVA.

3.2.1 Mapeamento - Annotations.

O hibernate disponibiliza um mapeamento simples para a modelagem de dados. Através do Annotations Podemos utilizar somente a um POJO em java para o mapeamento. Pegamos o POJO abaixo, ele será , mapeado para uma tabela em um banco, e se não existir a tabela podemos configurar o hibernate para criar a tabela.

```
@Entity
public class Amigos {

    @Id
    private Long id;

    @Column(name = "Nome", length = 50, nullable = false)
    private String nome;
    @Column(name = "endereco", length = 100)
    private String endereco;
    @Column(name = "telefone", length = 100)
    private String telefone;
    @Column(name = "celular", length = 100)
    private String celular;
    @Column(name = "email", length = 100)
    private String email;

    public Amigos() {
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getEndereco() {
        return endereco;
    }
}
```

Neste caso podemos observar que cada annotations, será gerada uma coluna diferente. O ID sempre será o identificador, ou chave primaria da tabela. Em cada Atributo podemos verificar que possui uma coluna, o tipo da coluna depende do Tipo do atributo java. Se tivermos uma atributo inteiro idade, sera gerada uma coluna de campo Integer no banco. Abaixo estão alguns configurações que o annotatios utiliza para o seu mapeamento, podendo usar ou não.

No exemplo citado acima , podemos perceber que são relações simples, mas e quando tivermos um tipo que não seja um primitivo, por exemplo um atributo pessoa, ou uma lista de pessoas?

3.2.2 Relações complexas.

Neste Tópico, veremos como relacionar relações no Hibernate via Annotations.

3.2.2.1 – Many To One.

Vamos analisar a seguinte Relação:

A tabela Gerada será a seguinte:

Para gera a Tabela acima, como já vimos anteriormente, precisamos criar Duas Classes POJOs, que no caso seria STUDENT e ADDRESS, a baixo as duas classes:


```
@Entity
@Table(name = "ADDRESS")
public class Address {

    private long addressId;
    private String street;
    private String city;
    private String state;
    private String zipcode;

    public Address() {
    }

    public Address(String street, String city, String state, String zipcode) {
        this.street = street;
        this.city = city;
        this.state = state;
        this.zipcode = zipcode;
    }

    @Id
    @GeneratedValue
    @Column(name = "ADDRESS_ID")
    public long getAddressId() {
        return this.addressId;
    }

    public void setAddressId(long addressId) {
        this.addressId = addressId;
    }

    @Column(name = "ADDRESS_STREET", nullable = false, length=250)
    public String getStreet() {
        return this.street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    @Column(name = "ADDRESS_CITY", nullable = false, length=50)
    public String getCity() {
        return this.city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    @Column(name = "ADDRESS_STATE", nullable = false, length=50)
    public String getState() {
        return this.state;
    }

    public void setState(String state) {
        this.state = state;
    }

    @Column(name = "ADDRESS_ZIPCODE", nullable = false, length=10)
    public String getZipcode() {
        return this.zipcode;
    }

    public void setZipcode(String zipcode) {
        this.zipcode = zipcode;
    }
}
```

Ao analisarmos as classes acima, percebemos que somente a classe STUDENT se refere a Classe Adress, contendo um Adress, Portanto a classe STUDENT é a classe pai.

3.2.2.2 – Many To Many.

Vamos analisar a seguinte Relação:

Queremos gera a seguinte tabela:

Percebemos que a tabela STUDENT_COURSE é uma tabela relacional, esta tabela o HIBERNATE irá gera-la para nós automaticamente. As outras duas tabelas precisamos criar os POJOS. Podemos observar como os a relação many-to-many está mapeada.

Percebemos que agora os Pojos se referenciam uns com os outros, criando assim a relação many-to-many.

3.2.2.3 – Tipo de Fetch.

EAGER

```
@OneToMany(fetch=FetchType.EAGER)
```

Quando temos a anotação acima percebemos que o tipo de busca é Eager. Imagine quando você usa o EAGER e dá um get num objeto, ele traz tudo que

está dentro do objeto, ou seja, se há um relacionamento 1 para N, no objeto será carregado todas as referências(N) dele. Se sua aplicação depende de performance, este pode ser um problema, pois isto ocupará um grande espaço de memória carregando todas as listas/dependências do objeto

LAZY

```
@ManyToOne(fetch = FetchType.LAZY)
```

Desse modo, quando acessamos o objeto ele não traz instantaneamente todas as dependências. Somente quando precisamos do atributo dependente é que a pesquisa (select ou JOIN) é feito, economizando memória. Para que o LAZY faça este select/JOIN é necessário declarar o fetch, dizer como será a pesquisa, se via Selec ou JOIN e você gera então a consulta.

3.3 . Data Access Object (DAO).

O Hibernate faz todas as operações automáticas de um CRUD, com algumas linhas podemos fazer as operações de salvar,deletar,procurar e alterar.

Primeiramente precisamos de uma factory, isto quer dizer que precisamos de uma conexão anteriormente e a um manager que irá manipular as sessões, mas este é um assunto para o próximo tópico, por enquanto vamos nos focar nos DAO do hibernate. Na figura a baixo podemos analisar certamente como estão os sendo realizados os DAO.

Podemos também utilizar a forma antiga, passando Selects para o banco retornar nossas consultas, isto é feito através do Criteria. A baixo duas formas de usar o Criteria.

3.4 . Arquivo de Configuração.

Precisamos utilizar um arquivo de configuração para o hibernate saber em qual classes irá realizar as operações, assim como o banco, a senha, e algumas outras configurações. A baixo um exemplo de configuração.

