

Hibernate

“É uma experiência eterna de que todos os homens com poder são tentados a abusar.”
– Baron de Montesquieu

Neste capítulo, você aprenderá a:

- Usar a ferramenta de ORM Hibernate;
- Gerar as tabelas em um banco de dados qualquer a partir de suas classes de modelo;
- Automatizar o sistema de adicionar, listar, remover e procurar objetos no banco;
- Utilizar anotações para facilitar o mapeamento de classes para tabelas;
- Criar classes de DAO bem simples utilizando o hibernate.

16.1 - 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.

Já projetos grandes onde o plano de ação padrão tomado pelo Hibernate não satisfaz as necessidades da empresa (como o uso de select *, joins etc), ele possui dezenas de otimizações que podem ser feitas para atingir tal objetivo.

16.2 - Criando seu projeto

Para criar seu projeto, é necessário baixar os arquivos .jar necessários para rodar o Hibernate e colocá-los no *classpath* do mesmo.

O site oficial do hibernate é o **www.hibernate.org** e lá você pode baixar a última versão estável do mesmo na seção Download. Após descompactar esse arquivo, basta copiar todos os jars para o nosso projeto.

Ainda falta baixar as classes correspondentes ao `HibernateAnnotations`, que utilizaremos para gerar o mapeamento entre as classes Java e o banco de dados. Eles são encontrados também no site do hibernate e contem outros jars que devemos colocar no nosso projeto.

Antigamente (até a versão 2 do hibernate) o mapeamento era feito somente através de arquivos xml, que era bem chato, e utilizávamos de uma ferramenta chamada Xdoclet que criava tais xmls. Hoje em dia o Xdoclet foi substituído pelas Annotations.

Banco de dados

O Hibernate traduz suas necessidades em código SQL para qualquer banco de dados. Continuaremos utilizando o MySQL em nossos exemplos, portanto não esqueça de copiar o arquivo .jar correspondente ao driver para o diretório lib de sua aplicação.

16.3 - Modelo

Utilizaremos uma classe que modela um produto para este capítulo:

```
package br.com.caelum.hibernate;

public class Produto {

    private Long id;
    private String nome;
    private String descricao;
    private Double preco;

    // adicione seus getters e setters aqui!
}
```

16.4 - Configurando a classe/tabela Produto

Para configurar a nossa classe `Produto`, basta adicionar alguns comentários especiais na definição da classe e nas nossas variáveis membro. O que faremos não são comentários de verdade, mas sim o que chamamos de anotações.

A grande diferença entre os dois – anotações e comentários – é que as anotações são bem estruturadas, seguem um padrão e são mantidas em tempo de execução, enquanto os comentários são perdidos em tempo de compilação, que impossibilita descobrir em tempo de execução o que havia sido comentado.

O código a seguir coloca nossa classe na tabela “Produto” e seta algumas propriedades e o id.

Atenção: toda classe que vai trabalhar com o Hibernate precisa de um (ou mais) campo(s) que será a chave primária (composta ou não).

Fora isso, existem diversas opções que podemos colocar como configurar para não aceitar campos `null` ou mudar o nome da coluna por exemplo. Para ler:

```
package br.com.caelum.hibernate;

@Entity
public class Produto {

    @Id @GeneratedValue
    private Long id;
```

```
@Column(name = "descricao", nullable = true, length = 50)
private String descricao;

private Double preco;
private String nome;

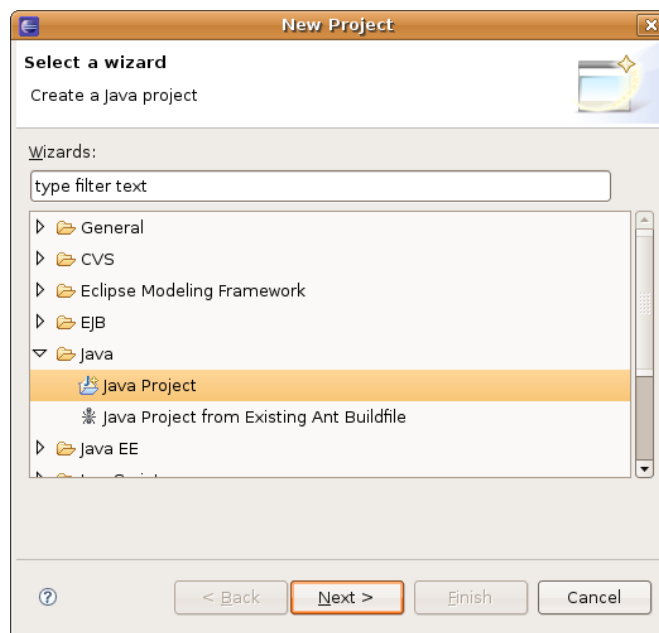
//metodos...
}
```

A especificação do EJB3 define tais anotações e possui diversas opções que podemos utilizar em nosso projeto. Sempre que possuir alguma duvida em relação as anotações lembre-se de ler a tal especificação.

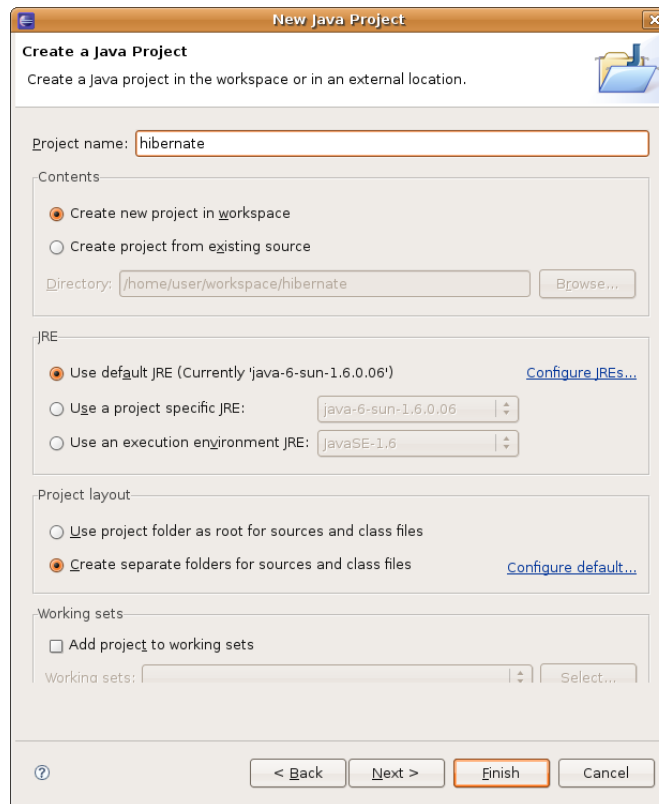
16.5 - Exercícios

1) Crie um novo projeto:

- Vá em *File -> New -> Project*.
- Escolha a opção *Java Project*.

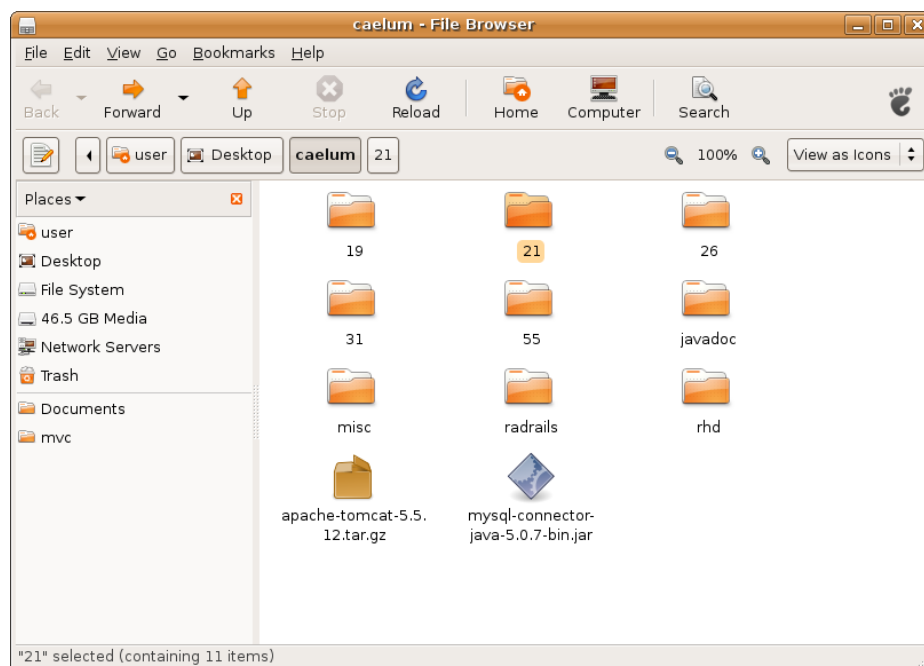


- Escolha **hibernate** como nome do projeto e clique em *Finish*. Confirme a mudança de perspectiva.

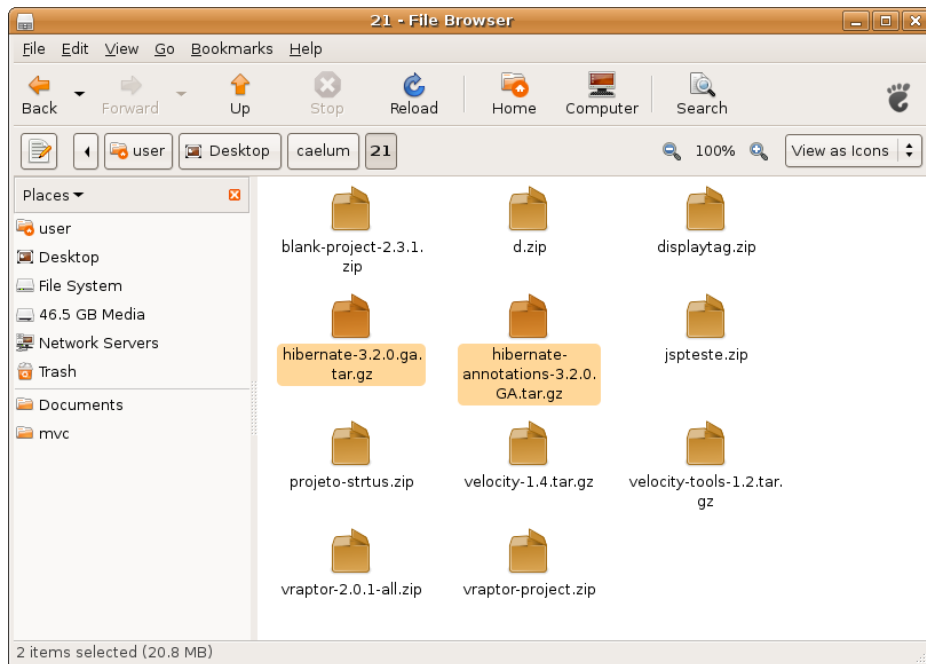


2) Descompacte o hibernate e o hibernate-annotations:

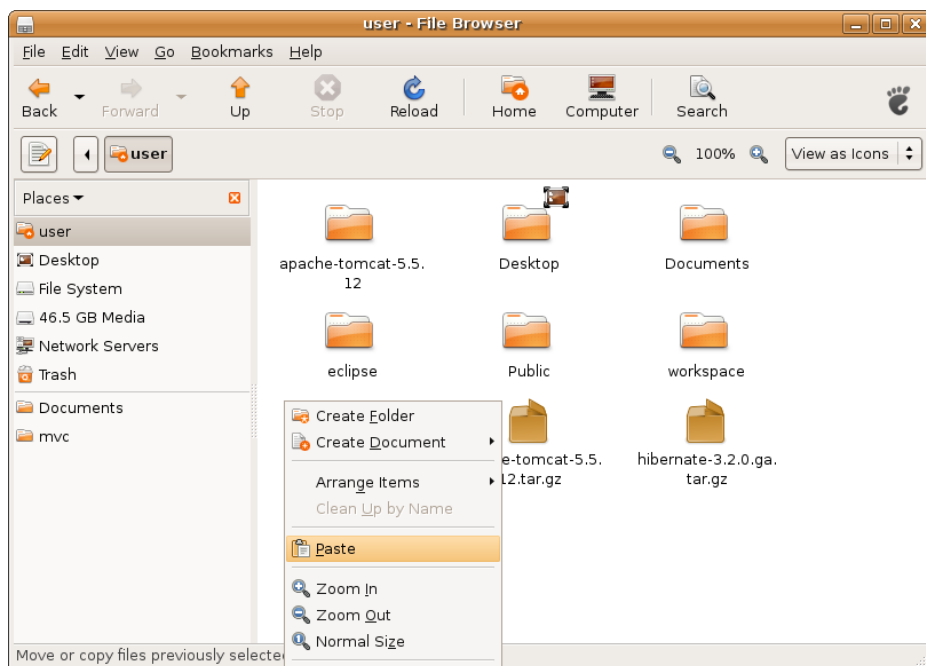
- a) Entre no diretório caelum, clicando no ícone da caelum no seu Desktop;
- b) Vá para a pasta 21:



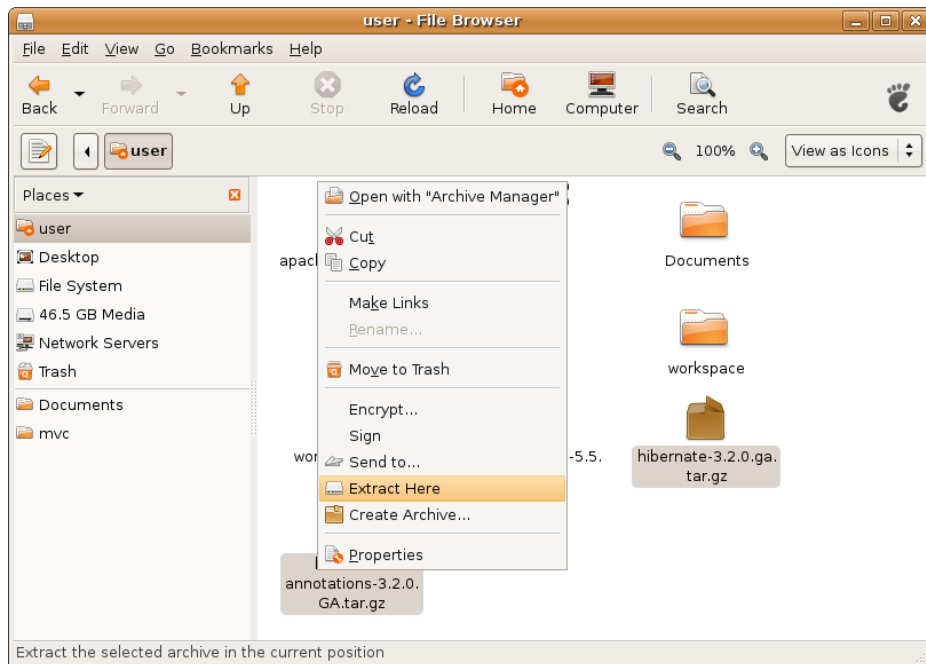
- c) Marque os dois arquivos do hibernate, clique com o botão direito e escolha *Copy*;



d) Vá para sua pasta padrão: webXXX, clique com o botão direito e escolha *Paste*;



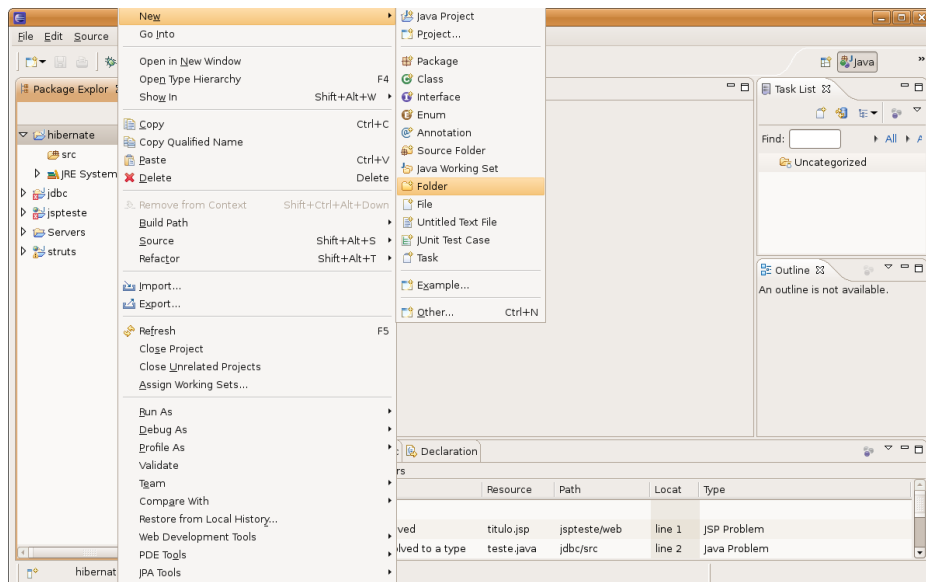
e) Marque os dois arquivos, clique com o botão direito e escolha *Extract here*;



f) Na pasta do seu usuário foram criadas duas pastas do hibernate, nelas estão os jar's que você utilizará no seu projeto.

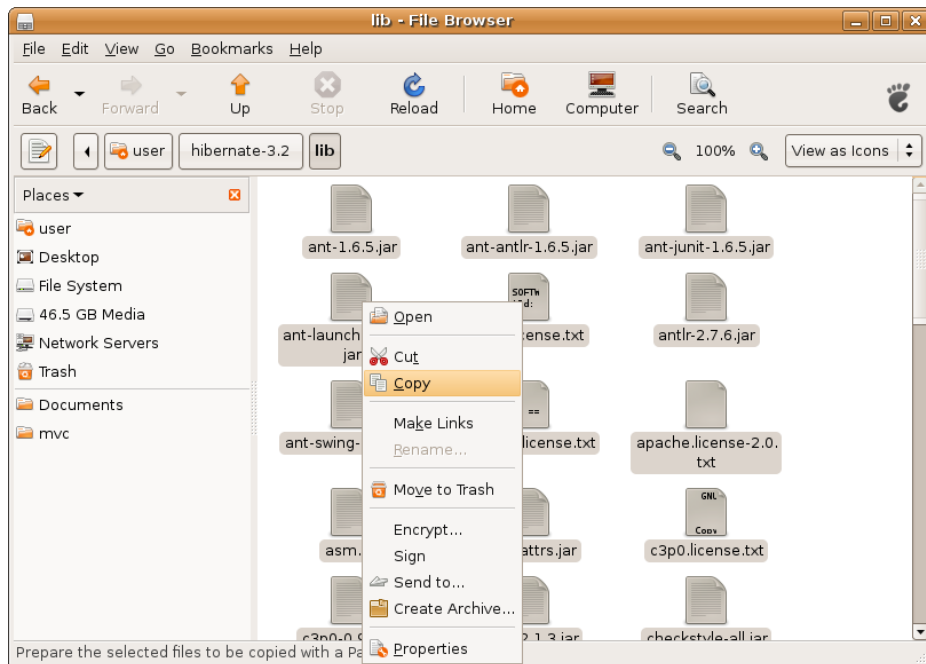
3) Copie os jars do hibernate para a pasta **lib** do seu projeto.

a) Clique com o botão direito no nome do projeto do hibernate e escolha *New -> Folder*. Escolha **lib** como nome dessa pasta.

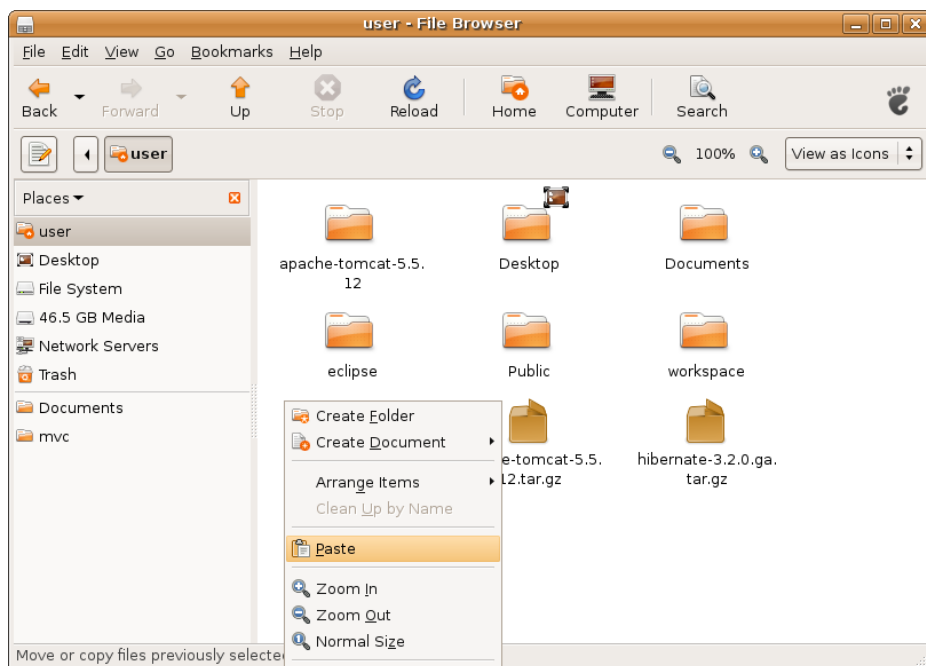


b) Vá para a pasta do seu usuário e copie os jar's *hibernate-3.2/hibernate3.jar* e *hibernate-annotations-3.2.0-GA/hibernate-annotations.jar* para pasta *workspace/hibernate/lib*

c) Volte para a pasta do seu usuário. Abra a pasta *hibernate-3.2/lib*. Selecione todos os arquivos (Ctrl+A), clique da direita em um deles e escolha *Copy*.

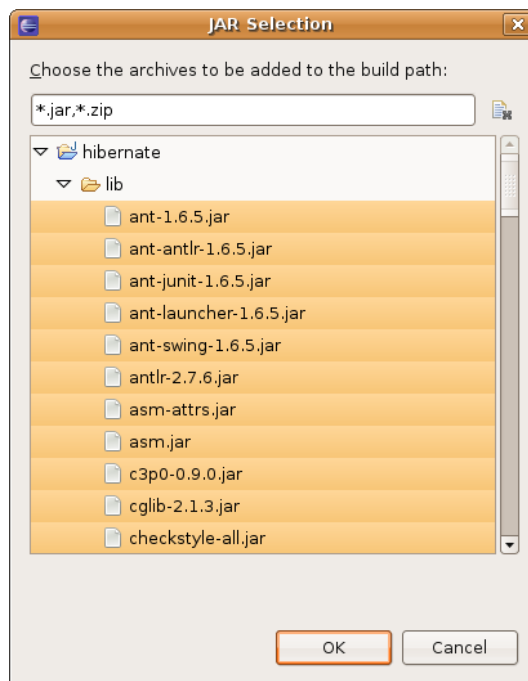
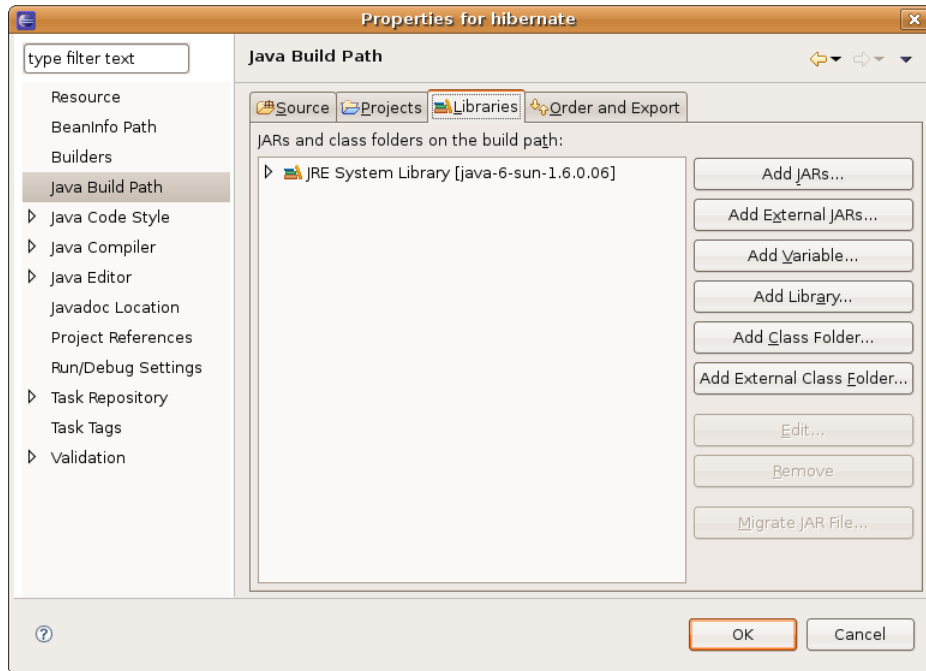


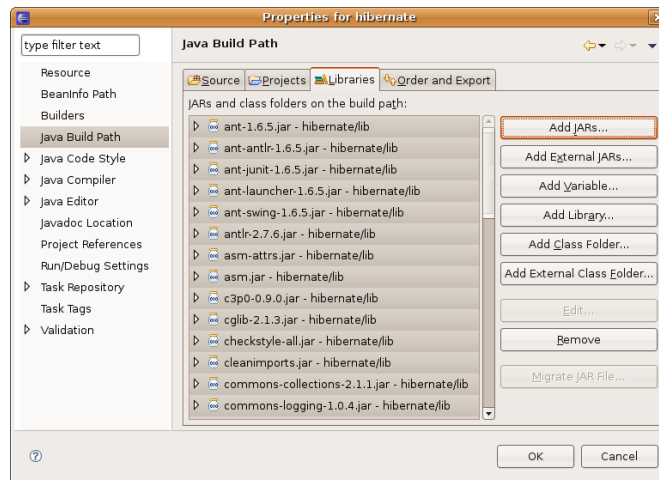
- d) Vá para a pasta do seu usuário e então em *workspace/hibernate/lib*. Clique da direita e escolha *Paste* para colar os jars nessa pasta.



- e) Agora repita esses dois últimos passos para a pasta *hibernate-annotations-3.2.0-GA/lib*.
- 4) Copie o jar do driver do mysql para a pasta lib do projeto também. Você pode achá-lo na pasta *workspace/jdbc*.
- 5) Vamos adicionar os jars no classpath do eclipse:
- a) Clique da direita no nome do seu projeto, escolha Refresh.
 - b) Clique novamente da direita, escolha o menu *Properties*;

- c) Escolha a opção *Java Build Path*;
- d) Escolha a aba *Libraries*;
- e) Escolha a opção *Add JARs* e selecione todos os jars do diretório *lib*;





6) Crie uma classe chamada Produto no pacote br.com.caelum.hibernate.

7) Adicione as seguintes variáveis membro:

```
private Long id;  
private String nome;  
private String descricao;  
private Double preco;
```

8) Gere os getters e setters usando o eclipse. (Source -> Generate Getters and Setters ou Ctrl + 3 -> ggas)

9) Anote a sua classe como uma entidade de banco de dados. Lembre-se de importar as anotações do pacote javax.persistence.

```
@Entity  
public class Produto {  
  
}
```

10) Anote seu field id como chave primária e como campo auto-gerado:

```
@Id  
@GeneratedValue  
private Long id;
```

16.6 - Propriedades do banco

Precisamos criar nosso arquivo de configuração, o hibernate.properties.

Os dados que vão nesse arquivo são específicos do hibernate e podem ser bem avançados, sobre controle de cache, transações, connection pool etc, tópicos que são abordados no curso FJ-26.

Para nosso sistema, precisamos de quatro linhas básicas, que configuram o banco, o driver, o usuário e senha, que já conhecemos, e uma linha adicional, que diz para o hibernate qual dialeto de SQL ele deve "falar": o dialeto do MySQL.

Uma das maneiras mais práticas é copiar o arquivo de mesmo nome que está no diretório etc, do hibernate descompactado que você baixou, no diretório src de sua aplicação.

Por padrão a configuração está de tal maneira que o hibernate irá usar um banco de dados do tipo Hyper-sonicSQL. Comente as linhas do mesmo (colocando # no começo). Se você copiar tal arquivo, descomente a parte que utiliza o mysql e configure corretamente a mesma, por exemplo:

```
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class = com.mysql.jdbc.Driver
hibernate.connection.url = jdbc:mysql://localhost/teste
hibernate.connection.username = root
hibernate.connection.password =
```

16.7 - Exercícios

1) Crie o arquivo **hibernate.properties** no seu diretório **src**.

```
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.connection.driver_class = com.mysql.jdbc.Driver
hibernate.connection.url = jdbc:mysql://localhost/teste
hibernate.connection.username = root
hibernate.connection.password =
```

16.8 - Configurando

Nosso primeiro passo é configurar o hibernate, portanto iniciamos instanciando uma `org.hibernate.cfg.AnnotationConfiguration`.

```
// Cria uma configuração para a classe Produto
AnnotationConfiguration cfg = new AnnotationConfiguration();
```

A partir daí podemos adicionar quantas classes desejarmos a nossa configuração.

```
// Adiciona a classe Produto
cfg.addAnnotatedClass(Produto.class);
```

No nosso caso, iremos adicionar somente a nossa classe `Produto`, gerando o seguinte resultado para configurar o Hibernate:

```
// Cria uma configuração para a classe Produto
AnnotationConfiguration cfg = new AnnotationConfiguration();
// Adiciona a classe Produto
cfg.addAnnotatedClass(Produto.class);
```

Só isso não é suficiente para que o Hibernate esteja configurado com a classe `Produto`. O Hibernate requer que descrevamos como a classe se relaciona com as tabelas no banco de dados e fizemos isso através das anotações do Hibernate.

Essa configuração poderia ser feita através de um arquivo XML chamado `hibernate.cfg.xml`, e em vez de utilizarmos as chamadas ao método `addAnnotatedClass` executaríamos o método a seguir:

```
// lê o arquivo hibernate.cfg.xml
cfg.configure();
```

Tal arquivo não será utilizado nesse curso mas sim no curso de Laboratório de MVC e Hibernate com JSF avançado (FJ-26), onde serão mostrados todos os detalhes do hibernate.

16.9 - Criando as tabelas

Vamos criar um programa que gera as tabelas do banco. Dada uma configuração, a classe `SchemaExport` é capaz de gerar o código DDL de criação de tabelas em determinado banco (no nosso caso, o MySQL).

Para exportar tais tabelas, fazemos uso do método `create` que recebe dois argumentos booleanos. O primeiro diz se desejamos ver o código DDL e o segundo se desejamos executá-lo.

```
new SchemaExport(cfg).create(true, true);
```

16.10 - Exercícios

- 1) Crie a classe `GeraTabelas`.

```
package br.com.caelum.hibernate;

public class GeraTabelas {

    public static void main(String[] args) {
        // Cria uma configuração para a classe Produto
        AnnotationConfiguration cfg = new AnnotationConfiguration();
        cfg.addAnnotatedClass(Produto.class);

        new SchemaExport(cfg).create(true, false);
    }
}
```

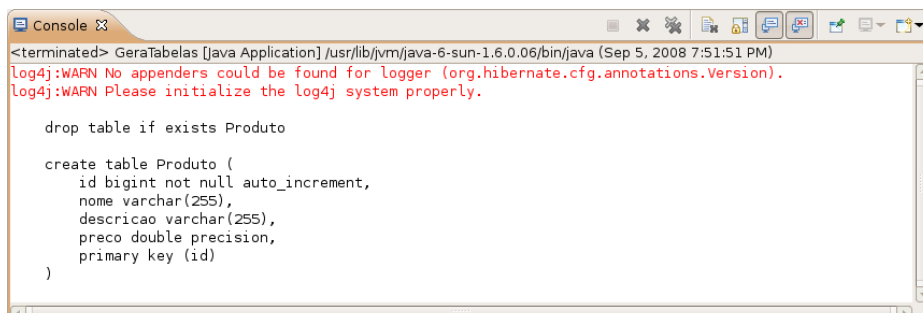
- 2) Adicione as seguintes linhas no seu arquivo `hibernate.properties`.

```
hibernate.show_sql = true
hibernate.format_sql = true
```

Essas linhas fazem com que todo sql gerado pelo hibernate apareça no console.

- 3) Crie suas tabelas executando o código anterior. Clique da direita no meio do código e vá em *Run As -> Java Application*.

O Hibernate deve reclamar que não configuramos nenhum arquivo de log para ele (dois warnings) e mostrar o código SQL que ele executou no banco.



```
<terminated> GeraTabelas [Java Application] /usr/lib/jvm/java-6-sun-1.6.0.06/bin/java (Sep 5, 2008 7:51:51 PM)
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.annotations.Version).
log4j:WARN Please initialize the log4j system properly.

drop table if exists Produto

create table Produto (
  id bigint not null auto_increment,
  nome varchar(255),
  descricao varchar(255),
  preco double precision,
  primary key (id)
)
```

Se o seu resultado não foi esse, você pode seguir a dica a seguir para configurar o log do hibernate.

16.11 - Dica: log do Hibernate

Você pode configurar o log do hibernate para verificar exatamente o que ele está fazendo, sendo tal atitude recomendada para todos os seus projetos.

Uma vez que essa parte de configuração do log não é o foco do curso, deixamos a dica aqui para você copiar tal configuração do próprio zip do hibernate para seu projeto.

- 1) Vá para o Desktop, escolha o File Browser;
- 2) Entre no diretório que você descompactou o hibernate;
- 3) Entre no diretório `etc`;
- 4) Copie o arquivo `log4j.properties`;
- 5) Volte para sua Home;
- 6) Entre no diretório `workspace/hibernate/src`;
- 7) Cole o arquivo `log4j.properties` nesse diretório;
- 8) Execute um refresh do seu projeto no eclipse (clique da direita no nome do projeto, Refresh).

Agora já podemos pegar uma fábrica de sessões do tipo `SessionFactory`, para isso basta chamar o método `buildSessionFactory` do objeto `cfg`.

```
package br.com.caelum.hibernate;

public class TesteDeConfiguracao {

    public static void main(String[] args) {

        AnnotationConfiguration cfg = new AnnotationConfiguration();
        cfg.addAnnotatedClass(Produto.class);
        SessionFactory factory = cfg.buildSessionFactory();
        factory.close();
    }
}
```

O Hibernate gera sessões através dessa `factory`. Essas sessões são responsáveis por se conectar ao banco de dados e persistir e buscar objetos no mesmo.

A maneira mais simples de buscar uma nova sessão e fechar a mesma é:

```
package br.com.caelum.hibernate;

public class TesteDeConfiguracao {

    public static void main(String[] args) {

        AnnotationConfiguration cfg = new AnnotationConfiguration();
```

```
        cfg.addAnnotatedClass(Produto.class);

        SessionFactory factory = cfg.buildSessionFactory();

        // cria a sessão
        Session session = factory.openSession();

        // fecha a sessão
        session.close();

        factory.close();
    }
}
```

16.12 - HibernateUtil

Vamos criar agora uma classe `HibernateUtil` que cuidará de:

- Instanciar a `SessionFactory` do Hibernate;
- Nos dar `Sessions` do hibernate quando solicitada.

```
public class HibernateUtil {

    private static SessionFactory factory;

    static {
        AnnotationConfiguration cfg = new AnnotationConfiguration();
        cfg.addAnnotatedClass(Produto.class);
        factory = cfg.buildSessionFactory();
    }

    public Session getSession() {
        return factory.openSession();
    }
}
```

O bloco estático das linhas 4 a 8 cuidará de configurar o Hibernate e pegar uma `SessionFactory`. Lembre-se que o bloco estático é executado automaticamente quando a classe é carregada pelo Class Loader e só neste momento; ele não será executado outras vezes, como quando você der `new HibernateUtil()`.

O método `getSession` devolverá uma `Session`, conseguida através do `SessionFactory` do Hibernate.

16.13 - Exercícios

- 1) Crie a sua classe `HibernateUtil` no pacote `br.com.caelum.hibernate`. No momento de importar `Session` lembre-se que não é a *classic*!

```
package br.com.caelum.hibernate;
```

```
public class HibernateUtil {  
  
    private static SessionFactory factory;  
  
    static {  
        AnnotationConfiguration cfg = new AnnotationConfiguration();  
        cfg.addAnnotatedClass(Produto.class);  
        factory = cfg.buildSessionFactory();  
    }  
  
    public Session getSession() {  
        return factory.openSession();  
    }  
}
```

16.14 - Erros comuns

O erro mais comum ao criar a classe `HibernateUtil` está em importar `org.hibernate.classic.Session` ao invés de `org.hibernate.Session`. Uma vez que o método `openSession` devolve uma `Session` que não é do tipo *classic*, o Eclipse pede para fazer um casting. Não faça o casting! Remova o seu import e adiciona o import correto.

16.15 - Salvando novos objetos

Através de um objeto do tipo `Session` é possível gravar novos objetos do tipo `Produto` no banco. Para tanto basta criar o objeto e depois utilizar o método `save`.

```
Produto p = new Produto();  
p.setNome("Nome aqui");  
p.setDescricao("Descrição aqui");  
p.setPreco(100.50);  
  
Session session = new HibernateUtil().getSession();  
session.save(p);  
System.out.println("ID do produto: " + p.getId());  
session.close();
```

16.16 - Exercícios

1) Crie uma classe chamada `AdicionaProduto` no pacote `br.com.caelum.hibernate`.

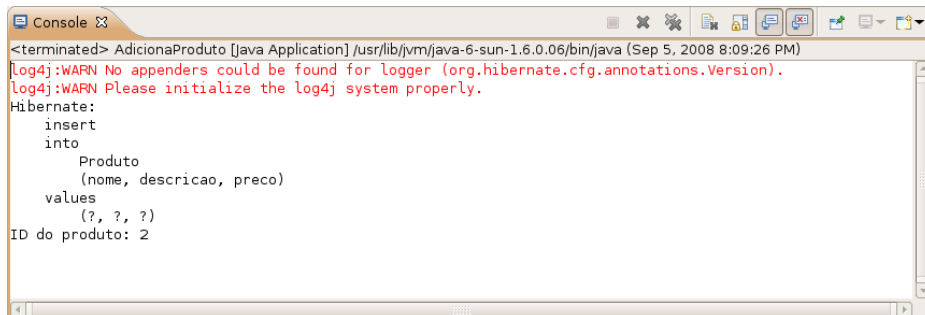
```
package br.com.caelum.hibernate;  
  
//Imports aqui CTRL+SHIFT+O  
  
public class AdicionaProduto {  
  
    public static void main(String[] args) {  
  
        Produto p = new Produto();
```

```
p.setNome("Nome aqui");
p.setDescricao("Descrição aqui");
p.setPreco(100.50);

Session session = new HibernateUtil().getSession();
session.save(p);
System.out.println("ID do produto: " + p.getId());

session.close();
}
```

2) Rode a classe e adicione cerca de 5 produtos no banco. Saída possível:



```
<terminated> AdicionaProduto [Java Application] /usr/lib/jvm/java-6-sun-1.6.0.06/bin/java (Sep 5, 2008 8:09:26 PM)
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.annotations.Version).
log4j:WARN Please initialize the log4j system properly.
Hibernate:
insert
into
    Produto
(nome, descricao, preco)
values
    (?, ?, ?)
ID do produto: 2
```

16.17 - Buscando pelo id

Para buscar um objeto pela chave primária, no caso o seu id, utilizamos o método `load`, conforme o exemplo a seguir:

```
Session session = new HibernateUtil().getSession();
Produto encontrado = (Produto) session.load(Produto.class, 1L);
System.out.println(encontrado.getNome());
```

16.18 - Criando o ProdutoDAO

Com os métodos que já conhecemos, podemos criar uma classe DAO para o nosso Produto:

```
public class ProdutoDAO {

    private Session session;

    public ProdutoDAO (Session session) {
        this.session = session;
    }

    public void salva (Produto p) {
        this.session.save(p);
    }

    public void remove (Produto p) {
        this.session.delete(p);
    }
}
```

```
}

public Produto procura (Long id) {
    return (Produto) this.session.load(Produto.class, id);
}

public void atualiza (Produto p) {
    this.session.update(p);
}
}
```

Através desse DAO, podemos salvar, remover, atualizar e procurar Produtos.

16.19 - Exercícios

- 1) Faça uma classe chamada ProdutoDAO dentro do pacote br.com.caelum.hibernate.dao

```
package br.com.caelum.hibernate.dao;

//imports aqui CTRL+SHIFT+O

public class ProdutoDAO {

    private Session session;

    public ProdutoDAO (Session session) {
        this.session = session;
    }

    public void salva (Produto p) {
        this.session.save(p);
    }

    public void remove (Produto p) {
        this.session.delete(p);
    }

    public Produto procura (Long id) {
        return (Produto) this.session.load(Produto.class, id);
    }

    public void atualiza (Produto p) {
        this.session.update(p);
    }
}
```

- 2) Adicione diversos objetos diferentes no banco de dados usando sua classe ProdutoDAO.

```
Session session = new HibernateUtil().getSession();
ProdutoDAO dao = new ProdutoDAO(session);
Produto produto = new Produto();
//...
```



```
dao.salva(produto);  
session.close();
```

- 3) Busque um id inválido, que não existe no banco de dados e descubra qual o erro que o Hibernate gera. Ele retorna null? Ele joga uma Exception? Qual?

session.flush() e transações

Note que os métodos `remove` e `update` precisam de `session.flush()` (ou transações). Isso na verdade é um detalhe do driver do MySQL com o JDBC. Sendo assim, sempre utilizamos o método `session.flush()` ou transações após a remoção ou atualização de algum objeto, independentemente de qual banco estamos utilizando. Na prática não existe obrigação do driver enviar o statement enquanto você não forçá-lo com o `flush`.

O uso clássico de uma transação é bem simples:

```
Transaction tx = session.beginTransaction();  
// executa tarefas  
tx.commit();
```

16.20 - Buscando com uma cláusula where

O Hibernate possui uma linguagem própria de queries para facilitar a busca de objetos. Por exemplo, o código a seguir mostra uma pesquisa que retorna todos os produtos com id maior que 2:

```
Session session = new HibernateUtil().getSession();  
List<Produto> lista = null;  
lista = session.createQuery("from br.com.caelum.hibernate.Produto where id>2").list();  
for (Produto atual : lista) {  
    System.out.println(atual.getNome());  
}
```

16.21 - ProdutoDAO: Listar tudo e fazer paginação

Vamos incluir mais três métodos na nossa classe `ProdutoDAO`.

Primeiro, vamos listar todos os produtos existentes no banco de dados. Para isso, vamos usar o método `createCriteria` de `Session` que cria um `Criteria`. Através de um `Criteria`, temos acesso a diversas operações no banco de dados; uma delas é listar tudo com o método `list()`.

Nosso método `listaTudo()` fica assim:

```
public List<Produto> listaTudo() {  
    return this.session.createCriteria(Produto.class).list();  
}
```

Mas o método acima devolve a lista com todos os produtos no banco de dados. Em um sistema com listagens longas, normalmente apresentamos a lista por páginas. Para implementar paginação, precisamos determinar que a listagem deve começar em um determinado ponto e ser de um determinado tamanho.

Usando o Criteria, como no `listaTudo` anteriormente, isso é bastante simples. Nosso método página fica assim:

```
public List<Produto> pagina (int inicio, int quantia) {  
    return this.session.createCriteria(Produto.class)  
        .setMaxResults(quantia).setFirstResult(inicio).list();  
}
```

O método `setMaxResults` determina o tamanho da lista (resultados por página) e o método `setFirstResult` determina em que ponto a listagem deve ter início. Por fim, basta chamar o método `list()` e a listagem devolvida será apenas daquela página!

E vamos adicionar também o método `listaAPartirDoTerceiro`:

```
public List<Produto> listaAPartirDoTerceiro () {  
    return this.session.createQuery("from Produto where id > 2").list();  
}
```

Poderíamos passar alguns parâmetros para essa busca, mas isso só será visto no cursos de Web Avançado.

16.22 - Exercícios

- 1) Modifique a sua classe `ProdutoDAO` e acrescente os métodos `listaTudo()` e `pagina()`

```
public List<Produto> listaTudo() {  
    return this.session.createCriteria(Produto.class).list();  
}  
  
public List<Produto> pagina (int inicio, int quantia) {  
    return this.session.createCriteria(Produto.class)  
        .setMaxResults(quantia).setFirstResult(inicio).list();  
}  
  
public List<Produto> listaAParitrDoTerceiro () {  
    return this.session.createQuery("from Produto where id > 2").list();  
}
```

- 2) Crie uma classe chamada `TestaBuscas`:

```
public class TestaBuscas {  
  
    public static void main(String [] args){  
        Session session = new HibernateUtil().getSession();  
        ProdutoDAO produtoDao = new ProdutoDAO(session);  
  
        System.out.println("****Listando Tudo****");  
        for(Produto p : produtoDao.listaTudo()) {  
            System.out.println(p.getNome());  
        }  
  
        System.out.println("****Listando Paginado****");  
        for(Produto p : produtoDao.pagina(2,3)) {
```

```
        System.out.println(p.getNome());
    }

    System.out.println("****Listando a partir do terceiro****");
    for(Produto p : produtoDao.listaAPartirDoTerceiro()) {
        System.out.println(p.getNome());
    }
}
}
```

16.23 - Exercícios para o preguiçoso

- 1) Mude a propriedade `hibernate.show_sql` para `true` no arquivo `hibernate.properties` e rode a classe acima.
- 2) Teste um programa que faz somente o seguinte: busca um produto por id. O código deve somente buscar o produto e não imprimir nada! Qual o resultado?

```
Session session = new HibernateUtil().getSession();
Produto encontrado = (Produto) session.load(Produto.class, 1L);
```

- 3) Tente imprimir o nome do produto do teste anterior, o que acontece?

```
Session session = new HibernateUtil().getSession();
Produto encontrado = (Produto) session.load(Produto.class, 1L);
System.out.println(encontrado.getNome());
```

- 4) Antes de imprimir o nome do produto, tente imprimir uma mensagem qualquer, do tipo: "O select já foi feito". E agora? Como isso é possível?

```
Session session = new HibernateUtil().getSession();
Produto encontrado = (Produto) session.load(Produto.class, 1L);
System.out.println("O select já foi feito");
System.out.println(encontrado.getNome());
```

Então, onde está o código do select? Ele deve estar no método `getNome()`, certo?

- 5) Imprima o nome da classe do objeto referenciado pela variável `encontrado`:

```
Session session = new HibernateUtil().getSession();
Produto encontrado = (Produto) session.load(Produto.class, 1L);
System.out.println("O select já foi feito");
System.out.println(encontrado.getNome());
System.out.println(encontrado.getClass().getName());
```

O Hibernate retorna um objeto cujo tipo estende `Produto`: ele não deixa de ser um `Produto` mas não é somente um `Produto`.

O método `getNome` foi sobrescrito nessa classe para fazer a busca na primeira vez que é chamado, economizando tempo de processamento.

É claro que para fazer o *fine-tuning* do Hibernate é interessante conhecer muito mais a fundo o que o Hibernate faz e como ele faz isso.

16.24 - Exercício opcional

- 1) Crie um sistema para cadastro e listagem de produtos usando o struts. Siga o padrão que utilizamos para cadastro e listagem de contatos.