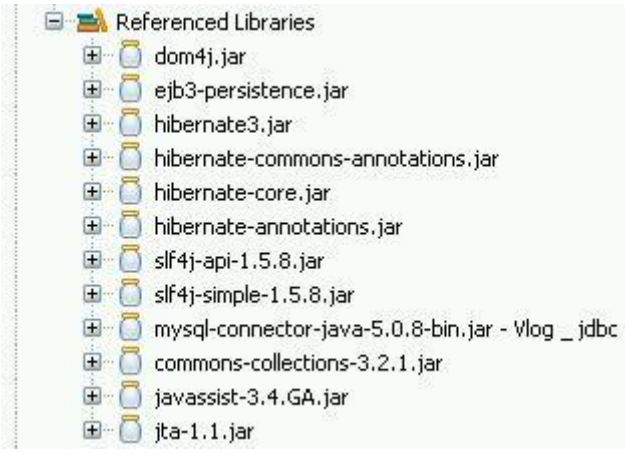
Acessando Dados com Java Hibernate Annotations

Objetivo: Mostrar um exemplo simples de acesso a dados com Java usando Hibernate Annotations.

Requerimentos: Para realizar esse tutorial você deve ter os seguintes JARs no seu projeto(No eclipse, de preferência), mais o jar para o driver do seu banco de dados utilizado:



Persistindo Pessoas
Iremos utilizar o mesmo exemplo de entidade dos tutoriais anteriores, a entidade pessoa, que ficou como segue:

Quote:

Pessoa:
ID, RG, NOME, IDADE, CIDADE, ESTADO

Crie sua tabela no banco de dados. Nesse tutorial foi utilizado MySql como banco de dados, se também utilizar MySql, pode criar a tabela com o script abaixo:

```
1.  CREATE TABLE pessoa (
2.    pessoa_id int(11) NOT NULL auto_increment,
3.    pessoa_rg varchar(20) default NULL,
4.    pessoa_nome varchar(20) default NULL,
5.    pessoa_idade int(2) default NULL,
6.    pessoa_cidade varchar(20) default NULL,
```

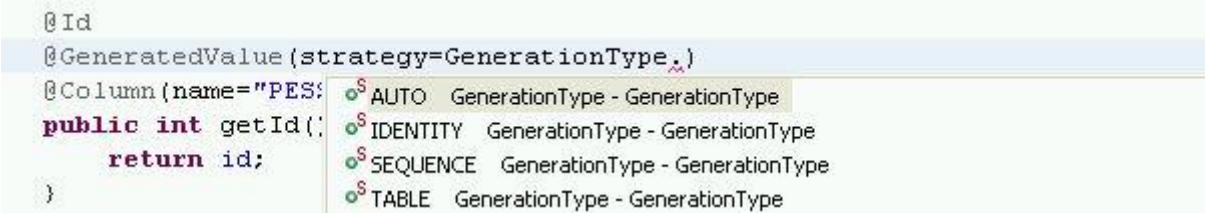
```
7.    pessoa_estado varchar(2) default NULL,
8.    PRIMARY KEY  (pessoa_id)
9.  )
```

Hibernate Annotations permite que você realize o mapeamento ORM sem utilizar XML, somente Annotations. Para isso você deve trabalhar com classes POJOs, ou seja, atributos privados, acessados pelos famosos métodos GET e SET.
Crie um pacote model e dentro a seguinte classe anotada:

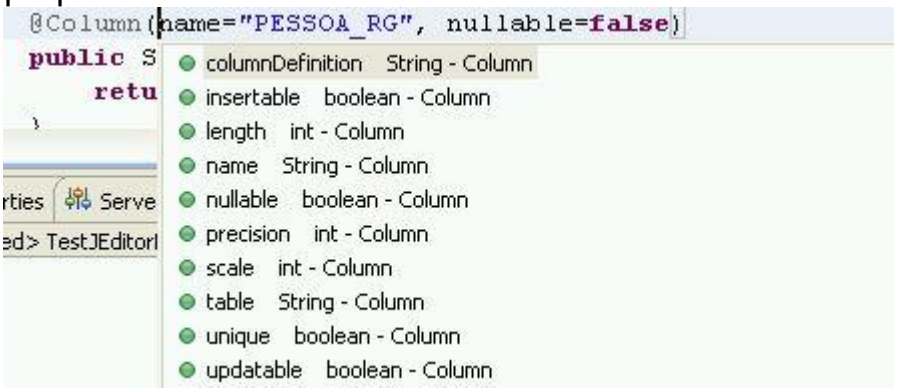
```
1.  package model;
2.  import javax.persistence.*;
3.
4.  @Entity
5.  @Table(name="PESSOA")
6.  public class Pessoa {
7.     private int id;
8.     private String rg;
9.     private String nome;
10.    private int idade;
11.    private String estado;
12.    private String cidade;
13.
14.    @Id
15.    @GeneratedValue
16.    @Column(name="PESSOA_ID")
17.    public int getId() {
18.       return id;
19.    }
20.    public void setId(int id) {
21.       this.id = id;
22.    }
23.
24.    @Column(name="PESSOA_RG", nullable=false)
25.    public String getRg() {
26.       return rg;
27.    }
28.    public void setRg(String rg) {
29.       this.rg = rg;
30.    }
31.
32.    @Column(name="PESSOA_NOME", nullable=false)
33.    public String getNome() {
34.       return nome;
35.    }
36.    public void setNome(String nome) {
37.       this.nome = nome;
38.    }
39.
40.    @Column(name="PESSOA_IDADE")
41.    public int getIdade() {
42.       return idade;
43.    }
44.    public void setIdade(int idade) {
45.       this.idade = idade;
46.    }
47.
48.    @Column(name="PESSOA_ESTADO")
49.    public String getEstado() {
50.       return estado;
51.    }
```

```
52.    public void setEstado(String estado) {
53.       this.estado = estado;
54.    }
55.
56.    @Column(name="PESSOA_CIDADE")
57.    public String getCidade() {
58.       return cidade;
59.    }
60.    public void setCidade(String cidade) {
61.       this.cidade = cidade;
62.    }
63. }
```

A seguir uma explicação básica sobre cada anotação:
@Entity: Usamos para marcar uma classe como entidade do banco de dados. Esta classe deve estar em um pacote e não ter argumentos em seu construtor.
@Table: Essa anotação serve para indicar em qual tabela iremos salvar os dados. Se você não usar essa anotação, o Hibernate usa o nome da classe para a tabela. O atributo **name** refere-se ao nome da tabela.
@Id: Usamos para mostra o identificador único(chave primária) de nossa classe persistente. No nosso caso, o identificador único é o campo ID.
GeneratedValue: Indica que o valor para o identificador único será gerado automaticamente. Você pode configurar a forma de geração dos valores através do atributor **strategy**. Se você não colocar uma estratégia, será usada a estratégia AUTO.



@Column: Utilizamos para especificar uma coluna da tabela do banco de dados. No exemplo acima, especificamos que o campo *RG* e *nome* . Se você não mapear a coluna, o nome da propriedade será usado como nome da coluna.



Configurando a fonte de dados

O próximo passo é configurar nossa fonte de dados. Nesse ponto usamos mapeamento XML, o único. Nos iremos adicionar os dados da fonte e as classes mapeadas. Observe bem os parâmetros da fonte de dados, essa parte é muito comum apresentar erros. Meu XML ficou assim:

```
1.  <?xml version='1.0' encoding='UTF-8'?>
2.  <!DOCTYPE hibernate-configuration PUBLIC
3.            "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4.            "http://hibernate.sourceforge.net/hibernate-configuration-
    3.0.dtd">
5.
6.  <hibernate-configuration>
7.      <session-factory>
8.          <!-- Database connection settings -->
9.          <property name="connection.driver_class">com.mysql.jdbc.Driver</proper
    ty>
10.         <property name="connection.url">jdbc:mysql://localhost:3306/simples_tu
    torial_annotations</property>
11.         <property name="connection.username">root</property>
12.         <property name="connection.password">senha</property>
13.
14.         <!-- JDBC connection pool (use the built-in) -->
15.         <property name="connection.pool_size">1</property>
16.
17.         <!-- SQL dialect -->
18.         <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

19.
20.         <!-- Echo all executed SQL to stdout -->
21.         <property name="show_sql">true</property>
22.
23.         <!-- Mapping files -->
24.         <mapping class="model.Pessoa"/>
25.     </session-factory>
26. </hibernate-configuration>
```

A fábrica de sessões

Quem já mexeu com Hibernate conhece muito bem a famosa fábrica de Sessões e a classe HibernateUtil. Crie um pacote chamado util e uma classe chamada **HibernateUtil**. Abaixo nossa classe HibernateUtil:

```
1.  package util;
2.
3.  import org.hibernate.SessionFactory;
4.  import org.hibernate.cfg.AnnotationConfiguration;
5.  public class HibernateUtil {
6.     private static final SessionFactory sessionFactory;
7.     static {
8.        sessionFactory = new AnnotationConfiguration().configure()
9.              .buildSessionFactory();
10.    }
11.
12.    public static SessionFactory getSessionFactory() {
13.       return sessionFactory;
```

```
14.    }
15. }
```

Agora nos resta testar para verificar se tudo que foi feito até aqui está certo. Uma classe de teste:

```
1.  import model.Pessoa;
2.  import org.hibernate.Session;
3.  import util.HibernateUtil;
4.  import org.hibernate.Transaction;
5.
6.  public class Teste {
7.     public static void main(String[] args) {
8.        Session sessao = HibernateUtil.getSessionFactory().openSession();
9.  Transaction t = sessao.beginTransaction();
10.
11.       Pessoa pessoa = new Pessoa();
12.       pessoa.setNome("William");
13.       pessoa.setRg("123456");
14.       pessoa.setCidade("São José dos campos");
15.       pessoa.setEstado("SP");
16.       pessoa.setIdade(21);
17.       sessao.save(pessoa);
18.       t.commit();
19.       sessao.close();
20.
21.    }
22. }
```

Perceba que neste teste simplesmente estamos instanciando uma pessoa e inserindo no banco de dados. Não há um DAO para abstrair as operações CRUD para com o banco. O próximo passo é criar esse DAO, que possibilitará maior flexibilidade e encapsulamento das operações com o banco de dados.

Fonte: Vanilla Hibernate Annotation