Vamos falar de Hibernate?

Hibernate é um framework ORM (Object-Relationship Management) muito bacana que nos permite ter muita produtividade na manipulação de objetos a serem persistidos no banco de dados. Vamos fazer um pequeno exemplo?

Para iniciar você vai precisar de:

- Um arquivo de configuração (geralmente chamado de **hibernate.cfg.xml**)
- Uma classe que configure suas conexões a partir deste XML (HibernateUtil.java ou DataSource.java, como preferir)
- Classes anotadas corretamente para poder fazer este mapeamento
- Uma aplicação que faça uso de tudo isso

Então vamos por a mão na massa: Nosso exemplo vai fazer uma inserção de uma simples classe "Cliente" em um banco MySQL (com uma tabela de nome distinto só pra não ficar tudo igual).

Parte 0: Nossa tabela no banco MySQL

```
create table tblCliente(
   idCliente int not null auto_increment,
   nome varchar(100),
   email varchar(100),
   dataCadastro date,
   constraint pk_cliente primary key (idCliente)
);
```

Com a tabela criada, agora vamos direto pra parte JAVA. Considere que seu projeto já está criado e com as bibliotecas do Hibernate referenciadas ok? Inclusive seu Connector MySQL.

Parte 1: Configure seu arquivo hibernate.cfg.xml

```
<hibernate-configuration>
   <session-factory>
       <property name="hibernate.dialect">
              org.hibernate.dialect.MySQLDialect
```

```
        </property>
        <property name="hibernate.connection.driver_class">
                com.mysql.jdbc.Driver
        </property>
        <property name="hibernate.connection.url">
                jdbc:mysql://localhost:3306/meu_bd
        </property>
        <property name="hibernate.connection.username">
                *seu_usuario do banco*
        </property>
        <property name="hibernate.connection.password">
                *sua_senha do banco*
        </property>

        <mapping class="Cliente"/>
    </session-factory>
</hibernate-configuration>
```

Observe que algumas tags são auto-explicativas, mas apenas para destacar:

- **dialect**: o Dialeto org.hibernate.dialect.MySQLDialect para usar as instruções do MySQL
- **connection.url**: *jdbc:mysql://localhost:3306/meubanco* – se você usou a configuração default do mysql, a porta é 3306 e “meubanco” é o banco de dados que você criou
- **connection.username**: seu usuário
- **connection.password:** sua senha
- **mapping**: o nome da classe que está anotada (no nosso caso, a classe Cliente)

uma vez isso configurado, vamos para a Etapa 2: Usar esta configuração para criar nosso SessionFactory. Bora lá com a classe que denominamos DataSource.java

Parte 2: Nosso DataSource

```
public class DataSource {
   private static SessionFactory sessionFactory;

   // trecho para ser executado assim que a classe
   // for referenciada
   static {
      try {
         // classe config para ler o XML
         Configuration config = new Configuration();
         config.configure("hibernate.cfg.xml");
         // Hibernate 4 usa o ServiceRegistry para criar a sessionFact
ory
         ServiceRegistry serviceRegistry;
         serviceRegistry = new StandardServiceRegistryBuilder()
                                 .applySettings(config.getProperties()
)
                                 .build();
         sessionFactory = config.buildSessionFactory(serviceRegistry);
      }catch (Exception ex) {
         System.err.println("Erro na conexao");
         ex.printStackTrace();
      }
   } // fim do bloco static

   public static SessionFactory getSessionFactory(){
      return sessionFactory;
   }
 } // fim da classe DataSource
```

Ok, classe que configura está ok, agora precisamos da nossa classe cliente devidamente anotada, vamos lá

Parte 3: A Classe Cliente

```
@Entity // para indicar que é uma classe que será persistida
@Table(name="tblCliente") // para indicar o nome da tabela
```

```
public class Cliente{
   @Id // indica que o atributo é uma chave primária
   @GeneratedValue(strategy=GeneratedType.AUTO) // auto-increment
   @Column(name="idCliente") // nome da coluna na tabela
    private int idCliente;

   @Column(name="nome")
    private String nome;

   @Column(name="email")
    private String email;

   @Column(name="dataCadastro")
   @Temporal(TemporalType.DATE)  // indica que é um dado do tipo Data
    private java.util.Date dataCadastro;

    // aqui vão os métodos GET e SET para cada atributo
 }
```

Agora, finalmente, basta criarmos nossa classe que irá utilizar tudo isso e persistir nosso Cliente no Banco de Dados:

Parte 4: A Aplicação

```
public class App{
  public static void main(String args[]){
     try{
        Cliente cliente = new Cliente();
        cliente.setNome("Jose Osorio de Almeida");
        cliente.setEmail("jose@osorio.com");
        cliente.setDataCadastro(Calendar.getInstance().getTime());
        // vou abrir uma conexão com o banco.
        Session session = DataSource.getSessionFactory().openSession()
;
        // como é uma operação de gravação, preciso de uma Transação
        Transaction transaction = session.beginTransaction();
        session.save(cliente); // fiz o insert no banco
        transaction.commit();  // efetivei a operação
```

```
            System.out.println("Cadastro Efetuado!");
        }
        catch(Exception ex){
            System.err.println("Erro!");
        }
    }
}
```

Pronto! Nosso primeiro exemplo está na mão! Claro que Hibernate tem muito mais do que apenas isso. Este é só um pequeno exemplo de como gravar uma classe simples em uma base de dados. Mas é o primeiro passo para você já começar a fazer seus sistemas!