**[Tutorial do Hibernate Hello World para Iniciantes com o Eclipse e o MySQL](https://www.codejava.net/frameworks/hibernate/hibernate-hello-world-tutorial-for-beginners-with-eclipse-and-mysql)**

Última atualização em 19 de novembro de 2017 |  [Impressão](https://www.codejava.net/frameworks/hibernate/hibernate-hello-world-tutorial-for-beginners-with-eclipse-and-mysql?tmpl=component&print=1&page=)[O email](https://www.codejava.net/component/mailto/?tmpl=component&template=protostar&link=8060944e2bf4b0eedc64f3ef7353562cf153eae9)

[**Master Microservices com Spring Boot e Spring Cloud**](https://click.linksynergy.com/link?id=d1Fh24*EiQI&offerid=323058.1352468&type=2&murl=https%3A%2F%2Fwww.udemy.com%2Fmicroservices-with-spring-boot-and-spring-cloud%2F&u1=WebTitleMS)

Este tutorial do Java Hibernate ajuda você a começar a usar a estrutura do Hibernate facilmente com o IDE do Eclipse e o banco de dados MySQL: Codificando seu primeiro programa Java que usa o Hibernate. Ao concluir este tutorial, você aprenderá:

* Como adicionar dependência do Hibernate no arquivo de projeto do Maven.
* Como criar um arquivo de configuração do Hibernate.
* Como usar anotações JPA para mapear uma classe Java para uma tabela no banco de dados.
* Como carregar o Hibernate Session Factory.
* Como executar operações CRUD (Criar, Ler, Atualizar e Excluir) com a Sessão de Hibernação

Os seguintes softwares e tecnologias são usados:

* JDK 8
* Eclipse Neon (4.6.0)
* Hibernate ORM 5.2.6.Final
* MySQL Server Community Edition 5.5
* MySQL Connector J - biblioteca de drivers JDBC para MySQL

**1. Por que o Hibernate?**

Em suma, o [Hibernate](http://hibernate.org/" \t "_blank) é uma ferramenta de Mapeamento Objeto-Relacional (ORM) para a linguagem Java. Isso significa que você pode mapear classes Java para tabelas de banco de dados e mapear tipos de dados Java para tipos de dados SQL. A estrutura do Hibernate economiza muito tempo para desenvolver aplicativos extensivos para o banco de dados, pois faz todo o trabalho pesado do banco de dados, para que você tenha mais tempo para se concentrar na implementação da lógica de negócios. Imagine que você pode executar operações CRUD e muito mais sem escrever nenhuma instrução SQL.

Além disso, o [Hibernate Query Language (HQL)](https://www.codejava.net/frameworks/hibernate/hibernate-query-language-hql-example" \t "_blank) permite que você escreva consultas orientadas a objetos para operações avançadas.

O Hibernate é uma implementação da Java Persistence API (JPA), portanto, seu código possui alta interoperabilidade com outros aplicativos Java EE.

Dito isto, a estrutura do Hibernate é uma escolha ideal para a camada de persistência no desenvolvimento do Java EE e, de fato, é amplamente utilizada e confiável por milhares de programadores Java.

Essa foi uma breve introdução sobre o Hibernate. Agora, vamos ver como construir seu primeiro aplicativo Java baseado no Hibernate.

**2. Configurando o banco de dados MySQL**

Você desenvolverá um aplicativo Java que gerencia uma coleção de livros no banco de dados. Portanto, execute o seguinte script SQL na ferramenta MySQL Workbench (ou no programa MySQL Command Line Client):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | CREATE DATABASE 'bookstore';  USE 'bookstore';    CREATE TABLE `book` (    `book\_id` int(11) NOT NULL AUTO\_INCREMENT,    `title` varchar(128) NOT NULL,    `author` varchar(45) NOT NULL,    `price` float NOT NULL,    PRIMARY KEY (`book\_id`),    UNIQUE KEY `book\_id\_UNIQUE` (`book\_id`),    UNIQUE KEY `title\_UNIQUE` (`title`)  ) ENGINE=InnoDB; |

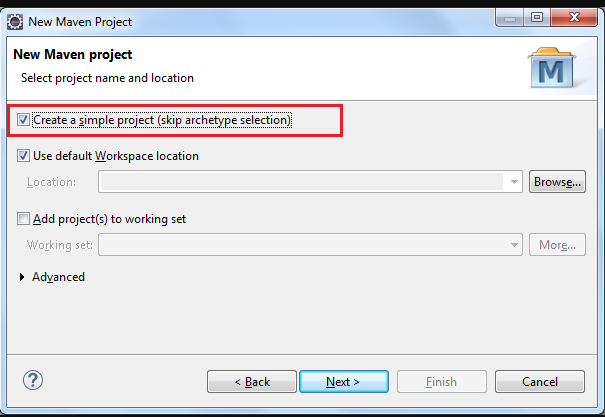
Isso cria um banco de dados chamado **livraria** com uma tabela chamada **book** . Esta tabela consiste em 4 colunas: **book\_id** (ID de incremento automático), **título** , **autor** e **preço** .

**3. Criando o projeto no Eclipse**

Nesta etapa, você criará um projeto Maven no Eclipse e adicionará dependência para o Hibernate no arquivo de configuração do projeto do Maven ( pom.xml ).

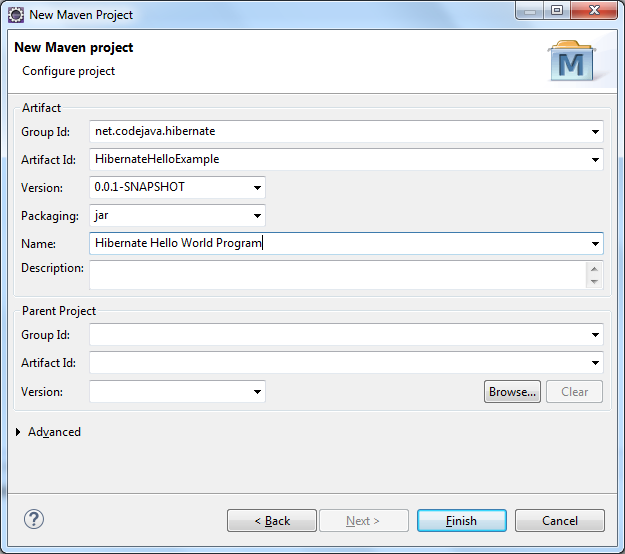
No Eclipse IDE, clique no menu **Arquivo> Novo> Projeto Maven** . Se você não vir essa opção, clique em **Arquivo> Novo> Projeto…** e selecione **Projeto Maven** no assistente para **Novo Projeto** e clique em **Avançar** .

Na próxima tela, marque a opção *Criar um projeto simples (ignorar seleção de arquétipo)* :



E clique em **Next** . Na próxima tela, insira informações para o artefato da seguinte forma:

* ID do grupo: net.codejava.hibernate
* ID do artefato: HibernateHelloExample
* Nome: Programa Hibernate Hello World



Clique em **Concluir** . O Eclipse gera a estrutura para o projeto, conforme mostrado abaixo:

Você pode ver a *Biblioteca do Sistema JRE* apontando para a versão antiga do Java SE 5. Nesse caso, clique com o botão direito e selecione Propriedades para mudar para o JDK 8.

Crie um novo pacote Java **net.codejava.hibernate** sob o diretório **src / main / java** clicando com o botão direito do mouse no projeto e selecione **Novo> Pacote** .

Agora, clique duas vezes no arquivo **pom.xml** para adicionar dependências para as bibliotecas Java do Hibernate e do MySQL Connector. Adicione o seguinte código XML logo antes do elemento </ project> :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <dependencies>      <dependency>          <groupId>org.hibernate</groupId>          <artifactId>hibernate-core</artifactId>          <version>5.2.6.Final</version>      </dependency>        <dependency>          <groupId>mysql</groupId>          <artifactId>mysql-connector-java</artifactId>          <version>5.1.40</version>      </dependency>    </dependencies> |

Clique em **Salvar** ( **Ctrl + S** ) e o Maven faz download automaticamente dos arquivos JAR de dependência do núcleo do Hibernate e do driver JDBC do MySQL Connector Java. Você pode ver os arquivos JAR adicionados sob a entrada **Maven Dependencies** do projeto.

**4. Escrevendo a Classe de Entidade**

Nesta etapa, você escreverá uma classe Java (classe de entidade) para mapear o **livro de** tabelas no banco de dados, usando anotações.

Crie a classe Book sob o pacote net.codejava.hibernate com 4 campos de acordo com as colunas na tabela de banco de dados:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class Book {      private long id;      private String title;      private String author;      private float price;  } |

Em seguida, use os recursos de geração de código do Eclipse para gerar getters e setters para esses campos (Atalho: **Alt + Shift + S** e digite **r** ).

Também gere um construtor vazio para a classe Book (atalho: **Alt + Shift + S** , depois digite **C** ). A classe Book agora se parece com isso:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | public class Book {      private long id;      private String title;      private String author;      private float price;        public Book() {      }        public long getId() {          return id;      }        public void setId(long id) {          this.id = id;      }        public String getTitle() {          return title;      }        public void setTitle(String title) {          this.title = title;      }        public String getAuthor() {          return author;      }        public void setAuthor(String author) {          this.author = author;      }        public float getPrice() {          return price;      }        public void setPrice(float price) {          this.price = price;      }  } |

Agora, você precisa usar algumas anotações para mapear essa classe para o **livro de** tabelas no banco de dados. Essas anotações vêm do JPA, então adicione esta declaração de importação primeiro:

|  |  |
| --- | --- |
| 1 | import javax.persistence.\*; |

Use o **@Entity** e **@Table** anotações antes da classe para mapeá-lo para a mesa:

|  |  |
| --- | --- |
| 1  2  3 | @Entity  @Table(name = "book")  public class Book { |

Em seguida, adicione as seguintes anotações logo antes do getter do campo **id** :

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @Id  @Column(name = "book\_id")  @GeneratedValue(strategy = GenerationType.IDENTITY)  public long getId() {      return id;  } |

A anotação **@Id** informa ao Hibernate que esta é a coluna ID da tabela; a anotação **@Column** mapeia o campo para uma coluna na tabela do banco de dados; e a anotação **@GeneratedValue** informa ao Hibernates que essa coluna ID é de incremento automático.

Você sabe, o Hibernate é inteligente, pois pode mapear automaticamente os campos de uma classe para os campos de uma tabela se os campos tiverem o mesmo nome e mapear automaticamente os tipos de dados Java para os tipos de dados SQL. Isso significa que você não precisa mapear explicitamente o **título** , o **autor** e o **preço dos** campos restantes .

E aqui está o código completo da classe Book :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | package net.codejava.hibernate;    import javax.persistence.\*;    /\*\*   \* Book.java   \* This class maps to a table in database.   \* @author www.codejava.net   \*   \*/    @Entity  @Table(name = "book")  public class Book {      private long id;      private String title;      private String author;      private float price;        public Book() {      }        @Id      @Column(name = "book\_id")      @GeneratedValue(strategy = GenerationType.IDENTITY)      public long getId() {          return id;      }        public void setId(long id) {          this.id = id;      }        public String getTitle() {          return title;      }        public void setTitle(String title) {          this.title = title;      }        public String getAuthor() {          return author;      }        public void setAuthor(String author) {          this.author = author;      }        public float getPrice() {          return price;      }        public void setPrice(float price) {          this.price = price;      }    } |

**5. Criando o arquivo XML de configuração do Hibernate**

Esta etapa envolve a criação do arquivo de configuração do Hibernate ( hibernate.cfg.xml ) para informar ao Hibernates como se conectar ao banco de dados e quais classes Java devem ser mapeadas para as tabelas do banco de dados.

No Eclipse, crie um arquivo XML denominado **hibernate.cfg.xml** no diretório src / java / resources com o seguinte código:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE hibernate-configuration PUBLIC          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"          "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>    <session-factory>      <!-- Database connection settings -->      <property name="connection.driver\_class">com.mysql.jdbc.Driver</property>      <property name="connection.url">jdbc:mysql://localhost:3306/bookstore</property>      <property name="connection.username">root</property>      <property name="connection.password">P@ssw0rd</property>      <property name="show\_sql">true</property>        <mapping class="net.codejava.hibernate.Book" />      </session-factory>  </hibernate-configuration> |

Como você pode ver, este arquivo de configuração especifica as informações de conexão do banco de dados (classe do driver JDBC, URL, nome de usuário e senha). O elemento <mapping> especifica que uma classe de entidade Java precisa ser mapeada. Aqui nós especificamos a classe Book sob o pacote net.codejava.hibernate .

A propriedade show\_sql é configurada como true para informar instruções SQL de impressão do Hibernate para cada consulta feita.

**6. Carregando a Session Factory do Hibernate**

Agora, vamos criar a classe principal para este programa. Criar o BookManager classe sob a net.codejava.hibernate pacote com a seguinte estrutura:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | public class BookManager {      protected SessionFactory sessionFactory;        protected void setup() {          // code to load Hibernate Session factory      }        protected void exit() {          // code to close Hibernate Session factory      }        protected void create() {          // code to save a book      }        protected void read() {          // code to get a book      }        protected void update() {          // code to modify a book      }        protected void delete() {          // code to remove a book      }        public static void main(String[] args) {          // code to run the program      }  } |

Adicione as seguintes instruções de importação antes da classe:

|  |  |
| --- | --- |
| 1  2  3  4  5 | import org.hibernate.Session;  import org.hibernate.SessionFactory;  import org.hibernate.boot.MetadataSources;  import org.hibernate.boot.registry.StandardServiceRegistry;  import org.hibernate.boot.registry.StandardServiceRegistryBuilder; |

No Hibernate, você realiza operações de banco de dados por meio de uma Session que pode ser obtida de uma SessionFactory . O SessionFactory carrega o arquivo de configuração do Hibernate, analisa o mapeamento e cria a conexão com o banco de dados. Escreva o seguinte código no método setup () para carregar o Hibernate SessionFactory com as configurações carregadas do arquivo hibernate.cfg.xml :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | final StandardServiceRegistry registry = new StandardServiceRegistryBuilder()          .configure() // configures settings from hibernate.cfg.xml          .build();  try {      sessionFactory = new MetadataSources(registry).buildMetadata().buildSessionFactory();  } catch (Exception ex) {      StandardServiceRegistryBuilder.destroy(registry);  } |

Uma vez que o SessionFactory do Hibernate é construído, você pode abrir uma Session e começar uma transação como esta:

|  |  |
| --- | --- |
| 1  2 | Session session = sessionFactory.openSession();  session.beginTransaction(); |

Em seguida, você pode chamar vários métodos na sessão para realizar operações de banco de dados, como save () , get (), update () e delete () . E finalmente confirme a transação e feche a sessão assim:

|  |  |
| --- | --- |
| 1  2 | session.getTransaction().commit();  session.close(); |

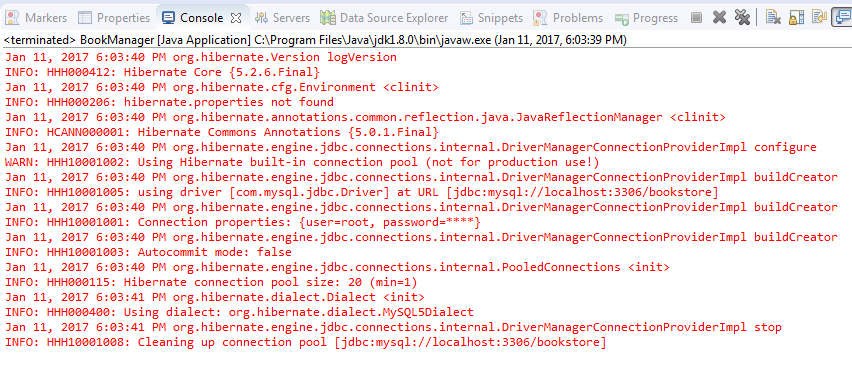
Feche a fábrica da sessão no método exit () :

|  |  |
| --- | --- |
| 1 | sessionFactory.close(); |

Agora, vamos atualizar o método main () assim:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public static void main(String[] args) {      BookManager manager = new BookManager();      manager.setup();        manager.exit();  } |

Em seguida, execute este programa (atalho: **Ctrl + F11** ) para verificar se a fábrica da sessão foi carregada com sucesso. Se você vir algo assim na visualização **Console** , isso significa que o programa carrega a fábrica da sessão com êxito:



**7. Executando operações CRUD com o Hibernate**

Agora, vamos ver como o Hibernate pode economizar muito tempo no que diz respeito à realização de operações triviais no banco de dados.

**Executando a operação CREATE:**

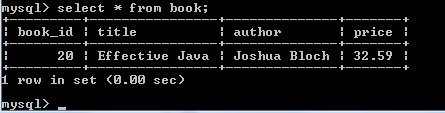
Chame o método **session.save (Object)** para persistir um objeto mapeado para o banco de dados. Atualize o método create () da classe BookManager da seguinte forma:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | protected void create() {      Book book = new Book();      book.setTitle("Effective Java");      book.setAuthor("Joshua Bloch");      book.setPrice(32.59f);        Session session = sessionFactory.openSession();      session.beginTransaction();        session.save(book);        session.getTransaction().commit();      session.close();  } |

Em seguida, atualize o método main () :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public static void main(String[] args) {      BookManager manager = new BookManager();      manager.setup();        manager.create();        manager.exit();  } |

Agora execute o programa novamente. Você deve ver que o Hibernate imprime uma instrução SQL na visualização Console assim:



Usando o MySQL Workbench ou o MySQL Command Line Client para verificar dados, você verá um novo registro inserido na tabela de **livros** :

Da mesma forma, repita os mesmos passos para outras operações descritas abaixo.

**Executando a operação READ:**

O **session.get (Class, long)** retorna um objeto da classe especificada que mapeia uma linha na tabela do banco de dados. Atualize o método read () da classe BookManager com o seguinte código:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | protected void read() {      Session session = sessionFactory.openSession();        long bookId = 20;      Book book = session.get(Book.class, bookId);        System.out.println("Title: " + book.getTitle());      System.out.println("Author: " + book.getAuthor());      System.out.println("Price: " + book.getPrice());        session.close();  } |

**Executando a operação UPDATE:**

Chame o método **session.update (Object)** para atualizar um objeto mapeado para o banco de dados. Adicione o seguinte código ao método update () da classe BookManager :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | protected void update() {      Book book = new Book();      book.setId(20);      book.setTitle("Ultimate Java Programming");      book.setAuthor("Nam Ha Minh");      book.setPrice(19.99f);        Session session = sessionFactory.openSession();      session.beginTransaction();        session.update(book);        session.getTransaction().commit();      session.close();  } |

**Executando a operação DELETE:**

Chame o método **session.delete (Object)** para remover um objeto mapeado do banco de dados. Adicione o seguinte código ao método delete () da classe BookManager :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | protected void delete() {      Book book = new Book();      book.setId(20);        Session session = sessionFactory.openSession();      session.beginTransaction();        session.delete(book);        session.getTransaction().commit();      session.close();  } |

Esse é um ótimo tutorial do Hibernate hello world usando o Eclipse IDE e o banco de dados MySQL. Você pode baixar o projeto de amostra na seção Anexos. Feliz aprendendo o Hibernate!

Assista ao vídeo tutorial: