

Programação Orientada a Objetos

Professor: Anderson Elias



Estácio

1

Acessando Banco de Dados com JDBC - JPA (SQL)

Contato:
anderson.elias@estacio.br



Estácio

2



*"O êxito da vida não se mede pelo caminho
que você conquistou, mas sim pelas
dificuldades que superou no caminho."*

Abraham Lincoln

Contato:
anderson.elias@estacio.br



Estácio

3

Roteiro – Modelagem Relacional

- Introdução:
 - *JDBC*;
 - *JPA*.
- Instalação e Configuração;
- Criação de Classe de Conexão;
- Comando SQL com JDBC;
- Inserindo registro na tabela;
- Outros recursos de JDBC;
- Recursos avançados de persistência em Java.

4

Introdução

Java Database Connectivity (JDBC)

- Conjunto de classes e interfaces (API) escritas em Java;
- Envia instruções SQL para qualquer banco de dados relacional;
- É uma API de baixo nível e serve de base para API's de alto nível;
- Para cada banco de dados há um driver JDBC.

5

Introdução

Java Persistence API (JPA)

- API padrão da linguagem Java que descreve uma interface comum para frameworks de persistência de dados;
- Define um meio de mapeamento objeto-relacional para objetos Java.

6

Conteúdo

Instalação e configuração

- Driver JDBC;
- Criação da classe de conexão.

Processamento de comandos SQL com JDBC

- Gerenciamento de conexões;
- Execução simples de consultas;
- Tratamento de exceções.

7

Instalação e configuração

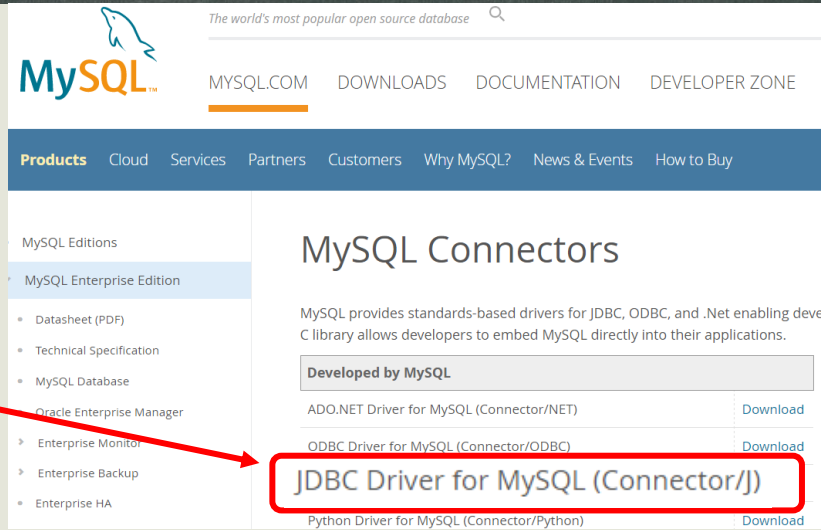
Para o SGBD MySQL

- Baixar o driver **JDBC** em
<http://www.mysql.com/products/connector/>
- Crie um **novo projeto** Java no Eclipse;
- Crie uma pasta **lib** na **raiz do projeto**, e copie para a pasta criada o arquivo:
`mysql-connector-java-<versao>-bin.jar`
- Inclua o arquivo no **classpath** do projeto criado.

8

Instalação e configuração

Baixar este Conector:



The world's most popular open source database

MySQL.COM DOWNLOADS DOCUMENTATION DEVELOPER ZONE

Products Cloud Services Partners Customers Why MySQL? News & Events How to Buy

MySQL Editions

- MySQL Enterprise Edition
 - Datasheet (PDF)
 - Technical Specification
 - MySQL Database
 - Oracle Enterprise Manager
 - Enterprise Monitor
 - Enterprise Backup
 - Enterprise HA

MySQL Connectors

MySQL provides standards-based drivers for JDBC, ODBC, and .Net enabling developers to embed MySQL directly into their applications.

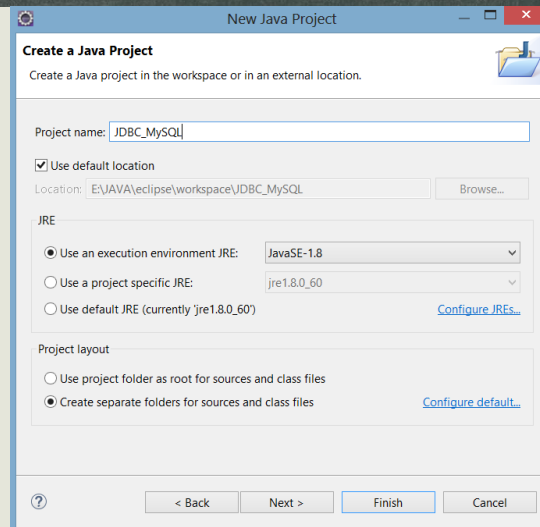
Developed by MySQL

ADO.NET Driver for MySQL (Connector/.NET)	Download
ODBC Driver for MySQL (Connector/ODBC)	Download
JDBC Driver for MySQL (Connector/J)	
Python Driver for MySQL (Connector/Python)	Download

9

Instalação e configuração

Criar um projeto em Java



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location: [Browse...](#)

JRE

☒ Use an execution environment JRE: [Configure JREs...](#)

☐ Use a project specific JRE:

☐ Use default JRE (currently 'jre1.8.0_60')

Project layout

☐ Use project folder as root for sources and class files

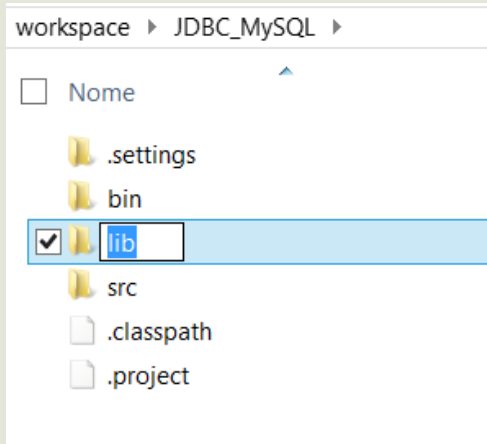
☒ Create separate folders for sources and class files [Configure default...](#)

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

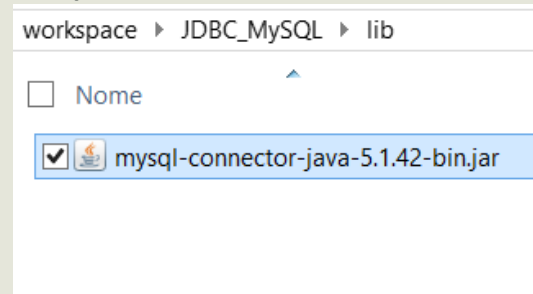
10

Instalação e configuração

Criar um projeto em Java



Copiar o arquivo de conexão .jar na pasta lib.

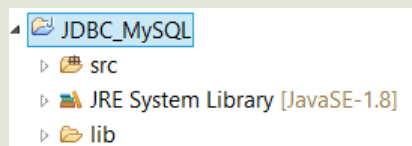


11

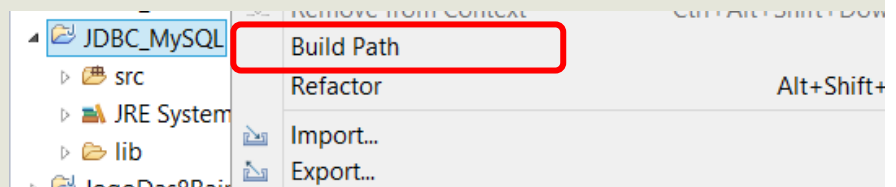
Instalação e configuração

Incluir o arquivo no classpath do projeto.

1) Refresh para aparecer a pasta lib recém criada.



2) Abrir o Build Path do Projeto.

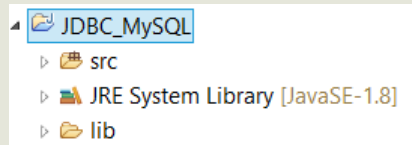


12

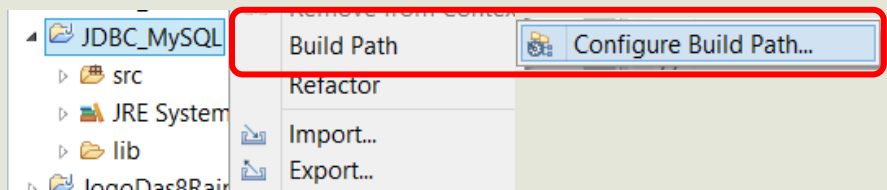
Instalação e configuração

Incluir o arquivo no classpath do projeto.

- 1) Refresh para aparecer a pasta lib recém criada.



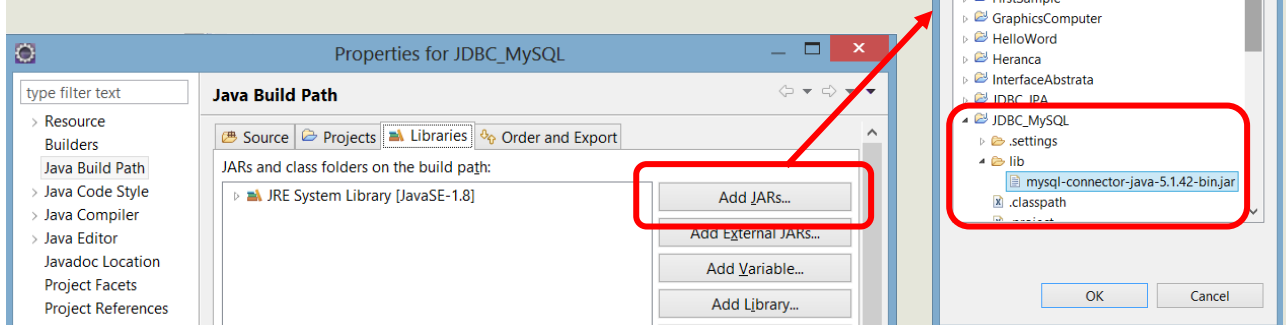
- 2) Abrir o Build Path do Projeto.



13

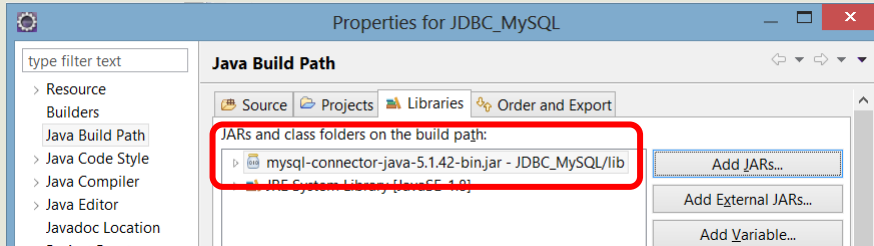
Instalação e configuração

Pressione o botão “Add JARs...” e aparecerá uma janela com todos os arquivos do projeto. Selecione o JAR desejado



14

Instalação e configuração

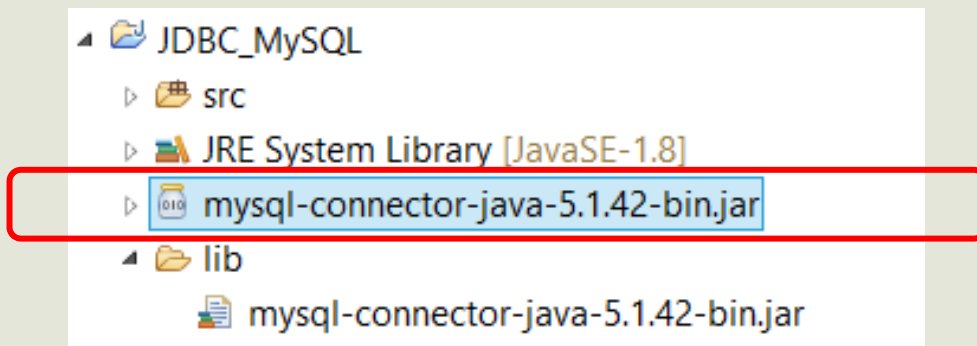


O JAR selecionado deve aparecer na guia “Libraries”

15

Instalação e configuração

Se tudo ocorreu bem, o JAR adicionado estará dentro de “Referenced Libraries” no “Package Explorer”:



16

Criação da classe de Conexão

Um padrão de projeto para aplicações de banco de dados:

- Data Access Object (DAO);
- “Separa” o código necessário para conexão com o SGBD.
- Reuso do componente de conexão

17

Criação da classe de Conexão

Estabelecendo uma conexão com o banco

- Classe `java.sql.Connection`:
 - *Representa uma instância de conexão com o servidor.*
- Classe `java.sql.DriverManager`:
 - *Retorna uma instância de Connection a partir da url do servidor.*
- Classe `java.sql.SQLException`:
 - *Representa o conjunto de exceções no acesso e operação com o banco.*

18

Criação da classe de Conexão

Vamos criar uma classe de conexão com o MySQL.

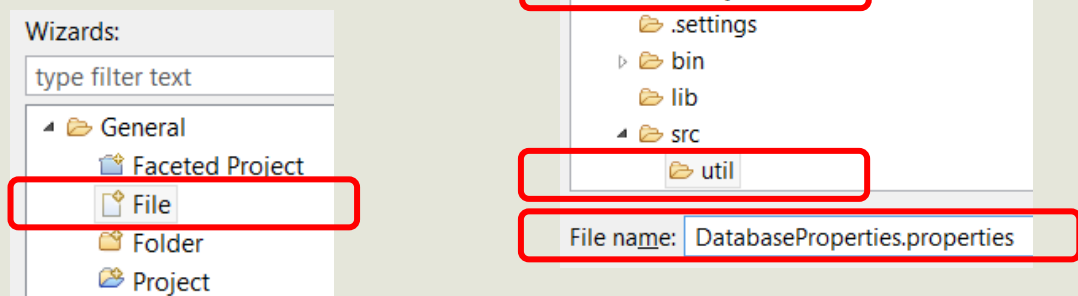
- Para evitar a declaração de informações de login e senha no meio do código, vamos utilizar um arquivo de propriedades.
- Vamos usar a API de properties de Java através das classes:
 - Properties
 - ResourceBundle

19

Criação da classe de Conexão

Iniciando a implementação

- Crie um pacote **util** em seu projeto.
- Crie um arquivo texto chamado de **DatabaseProperties.properties** e coloque-o no pacote **util**.



20

Criação da classe de Conexão

Definir as propriedades de conexão do banco.

- Abrir `DatabaseProperties.properties` e digitar o texto:

```
jdbcDriver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/livraria?user=root&password=1234
```

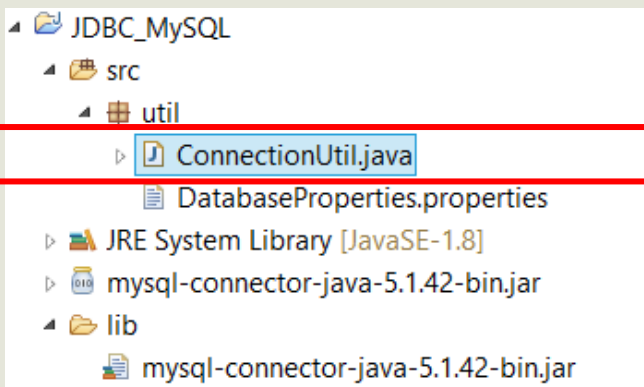
Observe que a string `jdbcDriver` se refere ao driver de conexão com o **MySQL**.

- A string `url` define o caminho até o servidor.
- A propriedades `user` e `password` definem o `login` e `senha` de conexão, respectivamente para o banco **livraria**.

21

Criação da classe de Conexão

Dentro do pacote util, crie a classe ConnectionUtil



Classe de Conexão com o Banco de Dados

22

Criação da classe de Conexão

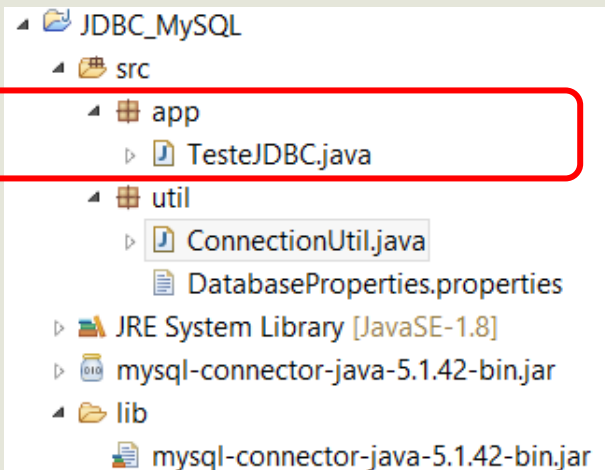
```
public class ConnectionUtil {
    public static Connection getConnection() {
        Properties properties = new Properties();
        try {
            ResourceBundle resources = ResourceBundle.getBundle("util.DatabaseProperties");
            for (@SuppressWarnings("rawtypes") Enumeration keys = resources.getKeys(); keys.hasMoreElements();) {
                final String key = (String) keys.nextElement();
                final String value = resources.getString(key);
                properties.put(key, value);
            }
            Class.forName(properties.getProperty("jdbcDriver")).newInstance();
            return (Connection) DriverManager.getConnection(properties.getProperty("url"));
        } catch (InstantiationException e) { e.printStackTrace(); }
        } catch (IllegalAccessException e) { e.printStackTrace(); }
        } catch (ClassNotFoundException e) { e.printStackTrace(); }
        } catch (SQLException e) { e.printStackTrace(); }
        }
        return null;
    }
}
```

23

Criação da classe de Conexão

Vamos a conexão com o banco

- Crie um pacote **app** na raiz do projeto.
- Crie uma classe **TesteJDBC** no pacote **app**
- Crie um método **main** na classe:



24

Criação da classe de Conexão

```
public class TesteJDBC {
    public static void main(String[] args) {
        try {
            Connection conn = null;
            conn = ConnectionUtil.getConnection();
            if (conn.getMetaData().getDatabaseProductName().equals("MySQL"))
                System.out.println("A conexão com o banco foi realizada com sucesso!");
            conn.close();
        } catch (SQLException e) {
            System.out.println("Problema na conexão com o SGBD: " + e.getMessage());
        }
    }
}
```

25

Comandos SQL com JDBC

Enviando comandos SQL para o SGBD.

- Classe **java.sql.Statement**:
 - Representa o processamento da consulta
- Classe **java.sql.ResultSet**:
 - Representa o resultado do processamento da consulta.

26

Comandos SQL com JDBC

Altere o método main:.

// após a linha.

```
System.out.println("A conexão com o banco foi realizada com sucesso!");
```

// Criando uma instância de Statement para processar a consulta.

```
Statement stm = (Statement) conn.createStatement();
```

// Criando a string de consulta.

```
ResultSet rs = stm.executeQuery("SELECT * FROM livraria.`livros`");
```

27

Comandos SQL com JDBC

Altere o método main:.

// Percorre a lista dos registros.

```
while (rs.next()) {
    int codigo = rs.getInt("codigo");
    String titulo = rs.getString("titulo");
    String preco = rs.getString("preco");
    System.out.println("Codigo:" + codigo + " | Título: " + titulo + ", | Preço: "+preco);
}
conn.close();
```

28

Inserindo registro na tabela

Uso do método `executeUpdate` de `Statement`:

//... Após o while, insira um registro na tabela **livros**.

```
stm = (Statement) conn.createStatement(); stm.executeUpdate(
    "INSERT INTO livros(titulo, preco) VALUES ('Prog. Imperativa', 37.52);");
```

// Reenviando a consulta

```
rs = stm.executeQuery("SELECT * FROM livraria.`livros`");
```

29

Inserindo registro na tabela

Uso do método `executeUpdate` de `Statement`:

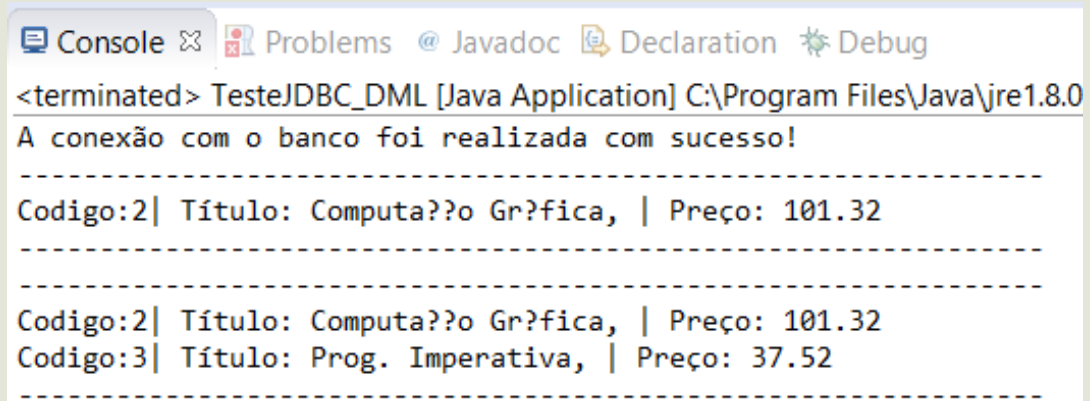
//... Executa o while, novamente e depois fecha a conexão `conn.close()`

```
while (rs.next()) {
    int codigo = rs.getInt("codigo");
    String titulo = rs.getString("titulo");
    String preco = rs.getString("preco");
    System.out.println("Codigo:" + codigo + " | Título: " + titulo + ", | Preço: "+preco);
}
conn.close();
```

30

Inserindo registro na tabela

Como Resultado, teremos isto no Console do Eclipse:

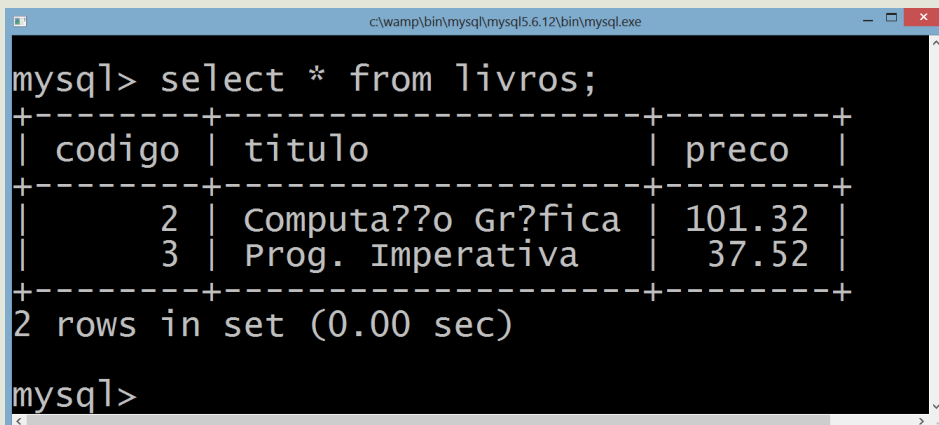


```
<terminated> TesteJDBC_DML [Java Application] C:\Program Files\Java\jre1.8.0
A conexão com o banco foi realizada com sucesso!
-----
Codigo:2| Título: Computação Gráfica, | Preço: 101.32
-----
Codigo:2| Título: Computação Gráfica, | Preço: 101.32
Codigo:3| Título: Prog. Imperativa, | Preço: 37.52
-----
```

31

Inserindo registro na tabela

Se verificarmos o Console do MySQL, teremos o mesmo:



```
c:\wamp\bin\mysql\mysql5.6.12\bin\mysql.exe
mysql> select * from livros;
+----+-----+-----+
| codigo | titulo | preco |
+----+-----+-----+
| 2 | Computação Gráfica | 101.32 |
| 3 | Prog. Imperativa | 37.52 |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

32

Recapitulando o que faz a **Statement**

1. Primeiro irá interpretar a consulta SQL fazendo **parse**;
2. Em seguida irá compilar a consulta SQL;
3. Depois irá planejar e otimizar o caminho de busca dos dados;
4. Por fim, irá executar a consulta otimizada, buscando e retornando os dados.

Sempre passar pelos quatro passos acima para cada consulta SQL enviada para o banco.

33

Outros recursos de JDBC

- Classe DataSource: Criação de pool de conexões
- Classe PreparedStatement:
 - Passagem de parâmetros nas consultas
 - Segurança -> evita ataques de SQL Injection!
- Controle de transações: Métodos commit e rollback de Connection.
- Tutorial (oficial de JDBC) na web:
<https://docs.oracle.com/javase/tutorial/jdbc/basics/>

34

Já a **PreparedStatement** faz o seguinte:

1. Irá interpretar a consulta SQL fazendo **parse**; já planejando e otimizando o caminho de busca dos dados, que seria feito só no passo 3 da Statement normal.
2. Depois compila e executar a consulta SQL;

Será mais rápida e com menos carga sobre o banco caso se deseja executar a mesma consulta várias vezes mudando apenas os parâmetros.

35

PreparedStatement - JDBC

Criar o Código abaixo para Salvar um novo autor:

```
System.out.println("\n Usando PreparedStatement \n");
try{
    PreparedStatement prepStm = conn.prepareStatement(
        "insert into autor (idade, matricula, nome) values (?, ?, ?)");
    prepStm.setInt(1, 33);
    prepStm.setString(2, "123");
    prepStm.setString(3, "João");
    prepStm.execute();
    prepStm.close();
    System.out.println("Autor Salvo!");
} catch (SQLException e) {
    e.printStackTrace();
}
```

36

PreparedStatement - JDBC

Criar o Código abaixo para Ler um novo autor:

```
PreparedStatement prepStm = conn.prepareStatement(
    "SELECT * FROM livraria.`autor` WHERE cod > ? AND idade <> ?");
prepStm.setInt(1, 2);
prepStm.setString(2, "123");
ResultSet rsPS = prepStm.executeQuery();
while (rsPS.next()) {
    int cod = rsPS.getInt("cod");
    String idade = rsPS.getString("idade");
    String matricula = rsPS.getString("matricula");
    String nome = rsPS.getString("nome");
    System.out.println("Codigo: " + cod + " / idade: " + idade + " /
        matricula: "+matricula+ " / nome: "+nome);
}
```

37

Recursos avançados de persistência em Java

• Java Persistence API (JPA)

- API padrão da plataforma para mapeamento Objeto-Relacional;
- Principais implementações:
 - Hibernate;
 - EclipseLink;
 - Apache OpenJPA
- Permite ao desenvolvedor mapear, armazenar, atualizar e recuperar registros do banco através de objetos em Java (e vice-versa)
- Tutorial de JPA com EclipseLink:

<http://www.vogella.com/tutorials/JavaPersistenceAPI/article.html>

38

Java Persistence API (JPA)

- É uma API padrão da linguagem Java que descreve uma interface comum para frameworks de persistência de dados. A JPA define um meio de mapeamento objeto-relacional para objetos Java simples e comuns, denominados beans de entidade. *(WikiPédia)*
- Iremos agora utilizar o JPA mapeando com Hibernate para realizar um CRUD em uma tabela (Entidade) do nosso banco de dados MySQL.

39

Java Persistence API (JPA)

- Precisamos inicialmente realizar a configuração das bibliotecas necessárias para fazer a conexão com nosso banco MySQL através do Hibernate.
- Arquivo de conexão para linkar a aplicação ao MySQL



mysql-connector-java-5.1.5-bin.jar

- XML de Persistência do JPA/Hibernate.

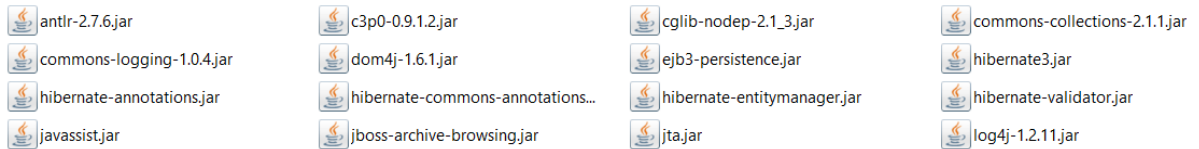


persistence.xml

40

Java Persistence API (JPA)

- Precisamos também das bibliotecas externas do Hibernate para nossa aplicação.



41

Java Persistence API (JPA)

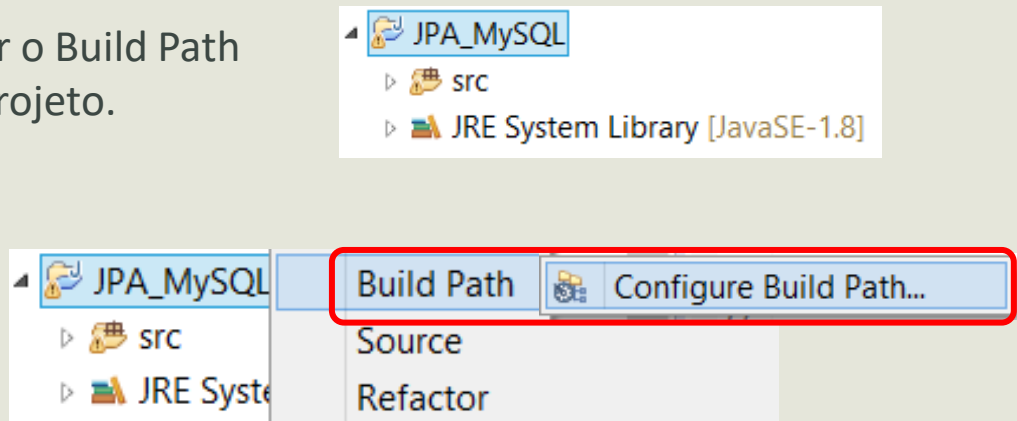
- Configurando as bibliotecas:
- Inicialmente iremos criar um novo projeto Java.
- Em seguida configurar as bibliotecas;
- Depois criar nosso projeto seguindo **MVC** para persistir com Hibernate.

42

JPA - Instalação e configuração

Incluir o arquivo no classpath do projeto.

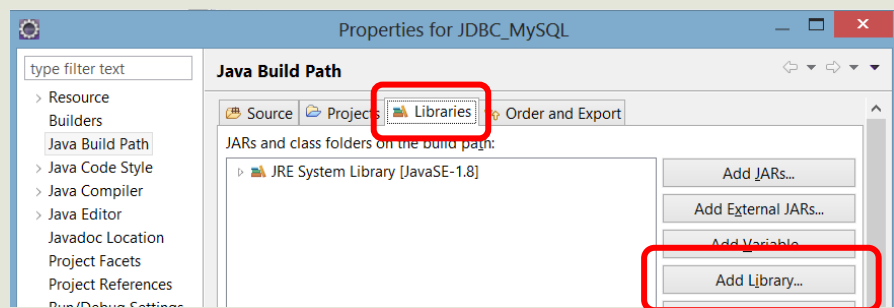
- 1) Abrir o Build Path do Projeto.



43

JPA - Instalação e configuração

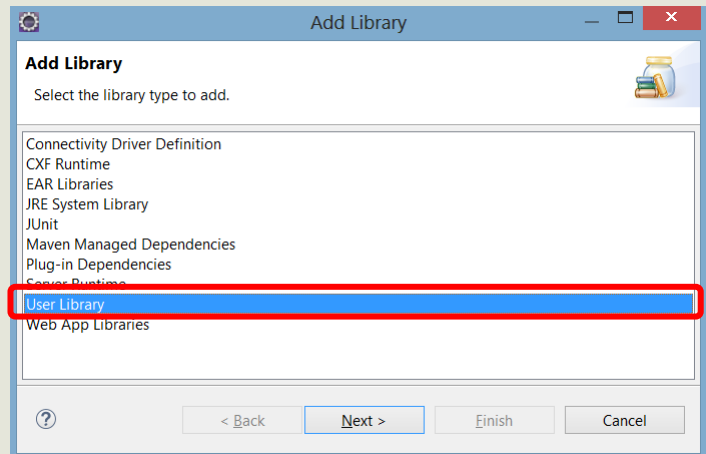
Pressione o botão “**Add Library...**” e aparecerá uma janela com todos os arquivos do projeto. Selecione o JAR desejado



44

JPA - Instalação e configuração

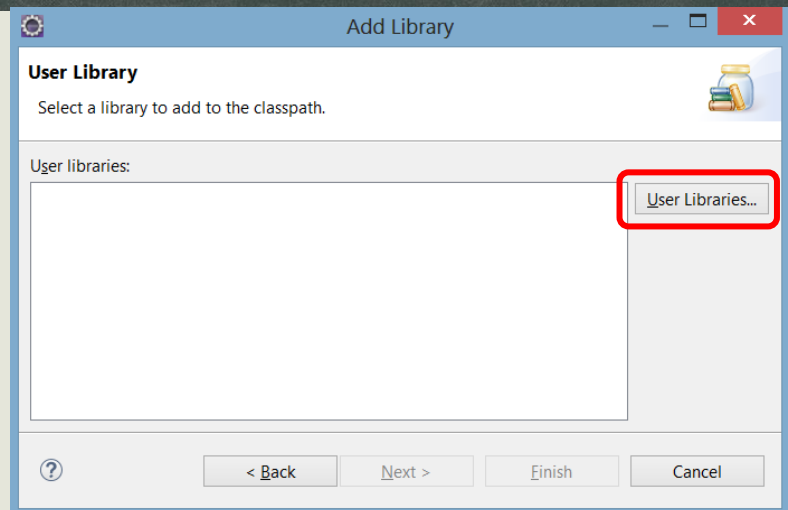
Selecione a opção “**User Library...**” e clique em Next.



45

JPA - Instalação e configuração

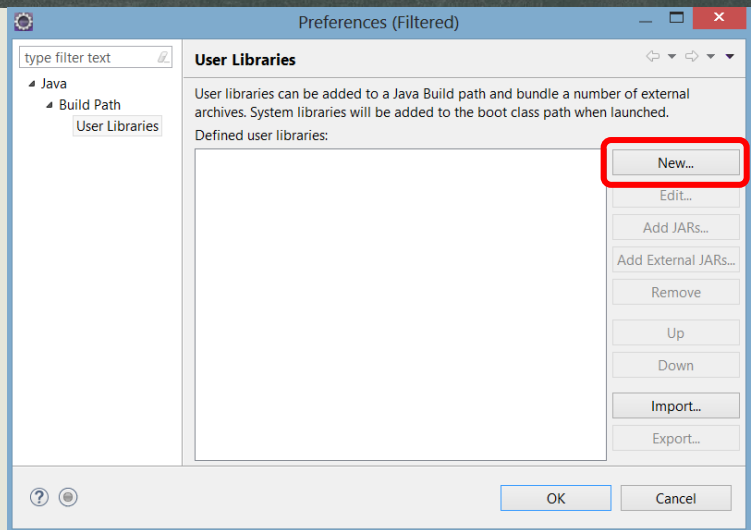
Clique no botão “**User Libraries...**”.



46

JPA - Instalação e configuração

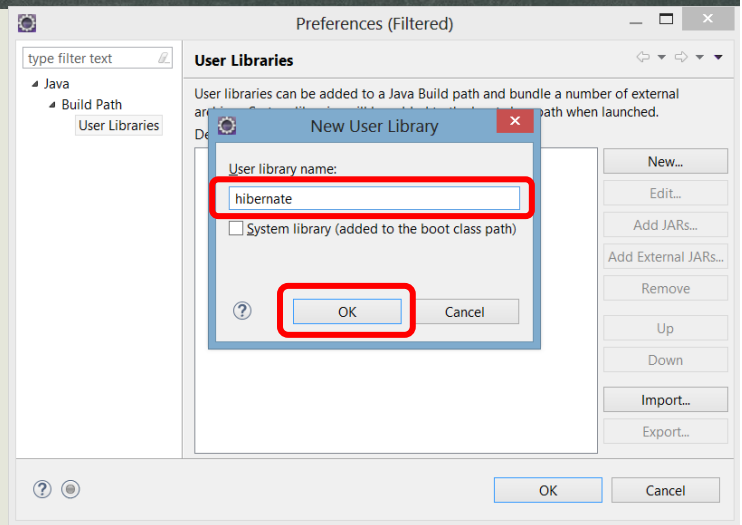
Clique no botão “**New...**” para darmos um nome a nossa biblioteca e selecionar as bibliotecas do Hibernate em seguida.



47

JPA - Instalação e configuração

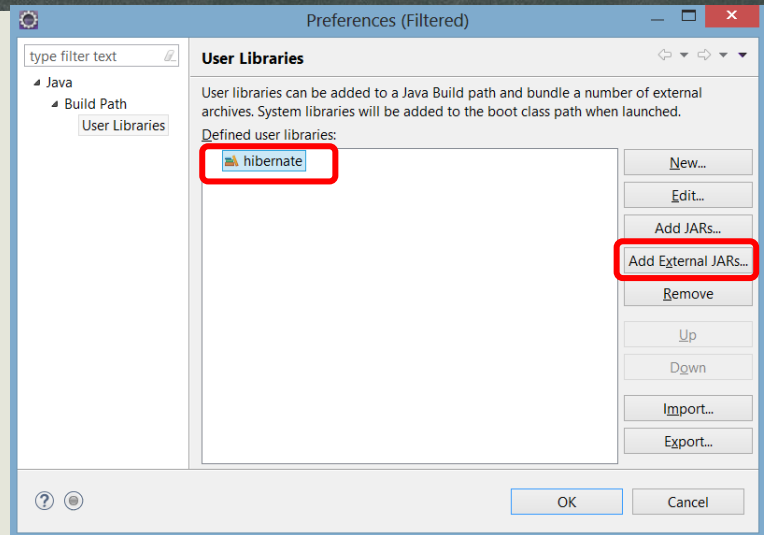
Informe um nome para sua biblioteca. Por Exemplo: “**hibernate**” em seguida clique em OK.



48

JPA - Instalação e configuração

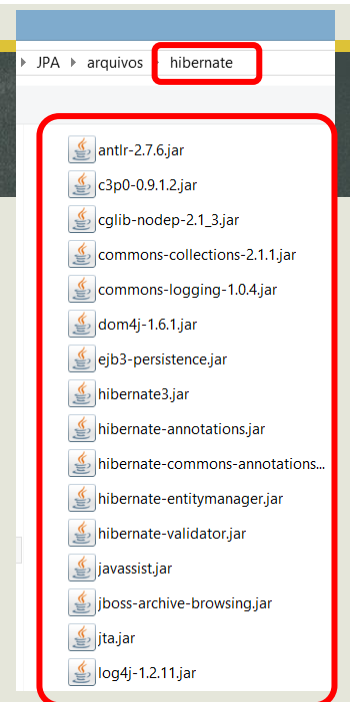
Definida o nome de nossa biblioteca, iremos agora selecionar ela “**hibernate**” e depois clicar em Add External JARs... Para referenciarmos todas as bibliotecas do Hibernate.



49

JPA - Instalação e configuração

Selecione todos os arquivos .jar (bibliotecas) do hibernate fornecida antes.

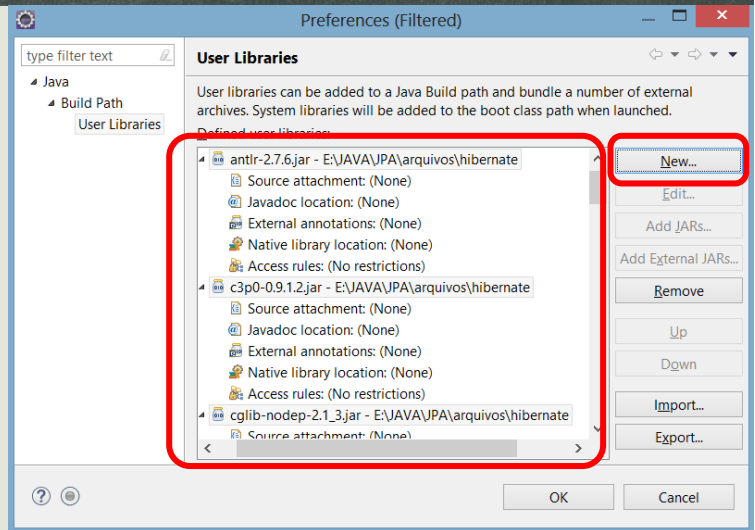


50

JPA - Instalação e configuração

Todas as bibliotecas necessárias para fazermos o mapeamento com o Hibernate está completa.

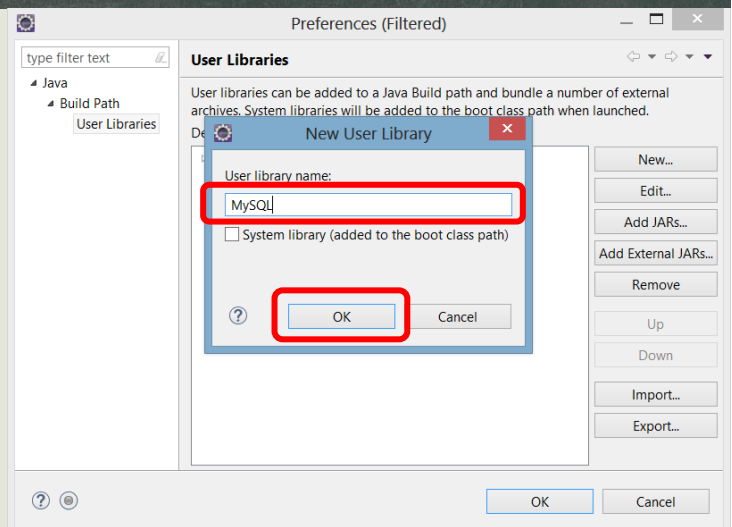
Agora faremos o mesmo para a conexão com o MySQL. Clique em **New...**



51

JPA - Instalação e configuração

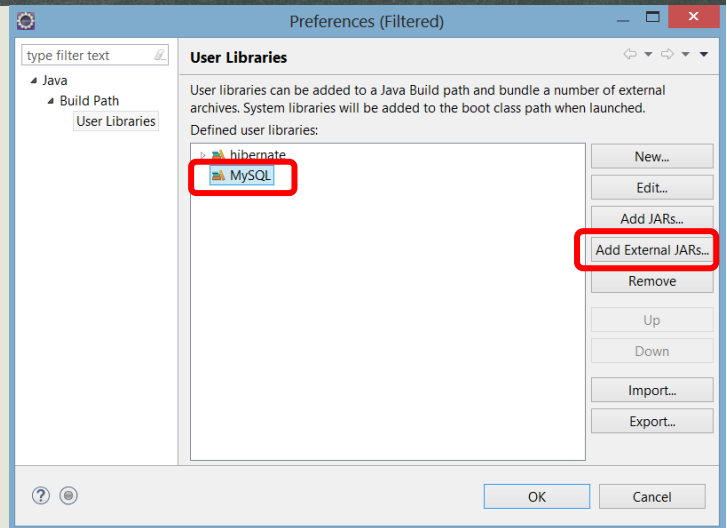
Informe um nome para sua biblioteca. Por Exemplo: **"MySQL"** em seguida clique em OK.



52

JPA - Instalação e configuração

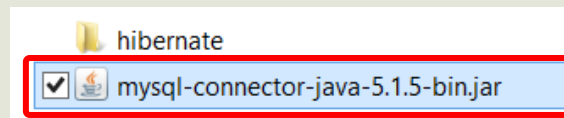
Selecione “MySQL” e depois clicar em Add External JARs... Para referenciarmos a biblioteca de conexão do MySQL.



53

JPA - Instalação e configuração

Selecione o arquivo .jar (biblioteca) do MySQL fornecida antes.



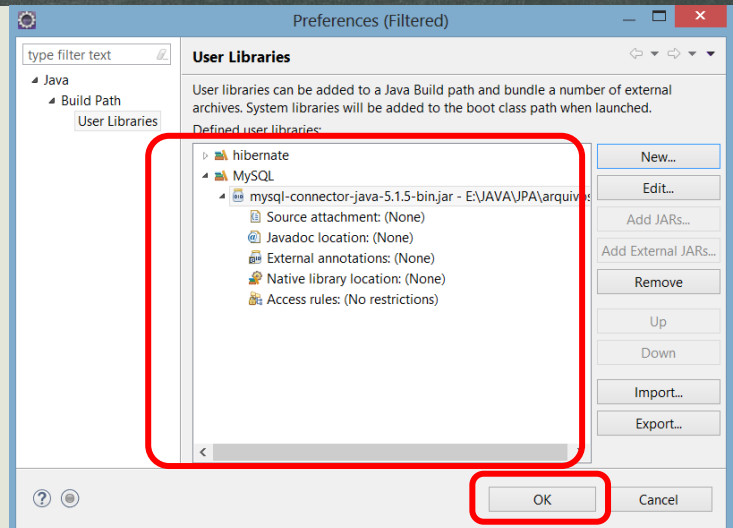
54

JPA - Instalação e configuração

Todas as bibliotecas necessárias para nosso projeto foram devidamente selecionadas.

(Hibernate + MySQL)

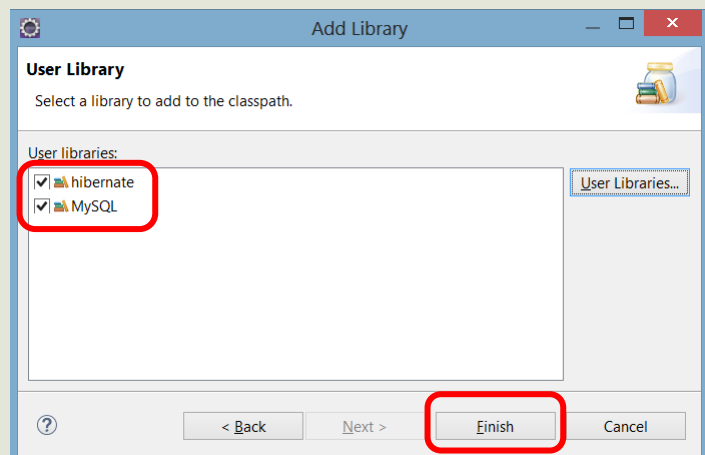
Clique em OK.



55

JPA - Instalação e configuração

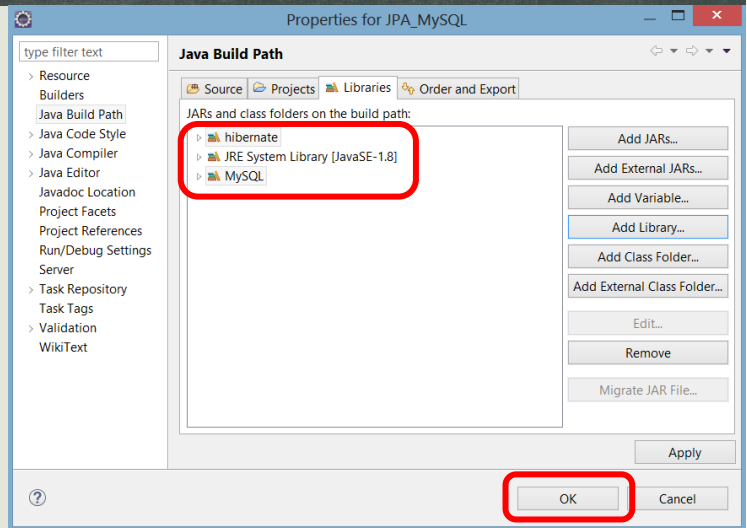
Certifique-se de que ambas estão marcadas no CheckBox e clique em Finish.



56

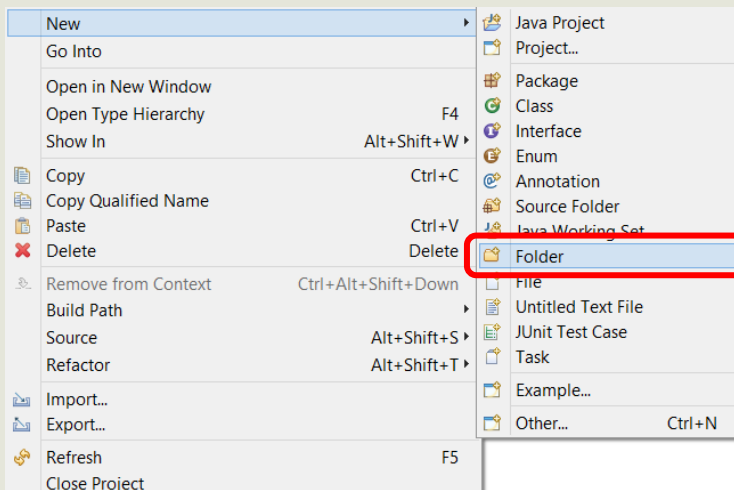
JPA - Instalação e configuração

Depois de selecionadas, nossas bibliotecas externas aparecerão na aba (Libraries) como mostra a janela ao lado. Após isto, clique em OK para voltarmos ao projeto java.



57

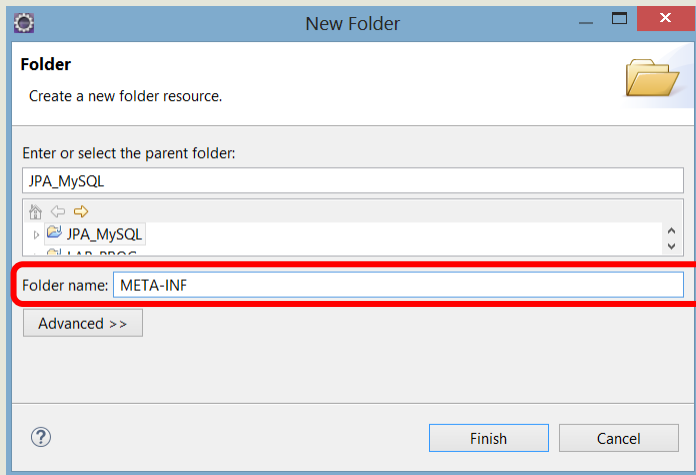
JPA - Instalação e configuração



Vamos criar agora uma pasta em nosso projeto para acomodar os arquivos de configuração do Hibernate.

58

JPA - Instalação e configuração



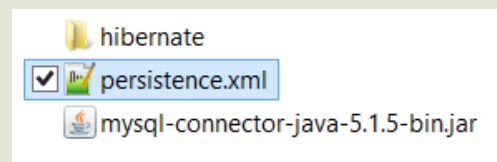
Esta pasta deve estar dentro da pasta **src**.

Vamos nomear a pasta como META-INF.

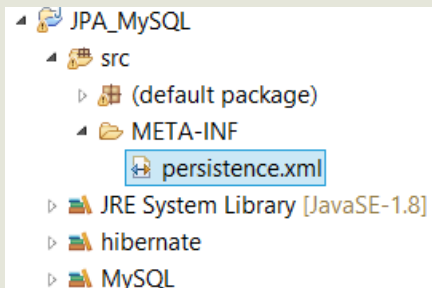
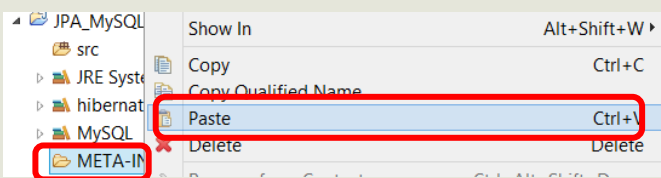
59

JPA - Instalação e configuração

Vá até o local onde guardou o arquivo persistence.xml, copie e depois volte ao projeto java, selecione a nova pasta META-INF, e cole.



Seu projeto deve ficar assim:



60

JPA - Instalação e configuração

No persistence.xml atenção a área de conexão com o banco.

```
<persistence-unit name="autor" transaction-type="RESOURCE_LOCAL">
```

Unidade de Persistência que usaremos em breve

```
<!-- Conexão com o banco de dados -->
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
<property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver" />
<property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/livraria" />
<property name="hibernate.connection.user" value="root" />
<property name="hibernate.connection.password" value="12345" />
<property name="hibernate.hbm2ddl.auto" value="update"/>
```

Usuário

Senha

Banco de Dados

61

JPA - Instalação e configuração

Vamos criar uma classe **autor** e em seguida informe que é uma entidade e depois uma tabela. Importe os pacotes.

Seguindo a "Notation" correta para mapeamento com as tabelas do banco.

```
1
2 public class Autor {
3
4 }
```

```
1 import javax.persistence.Entity;
2 import javax.persistence.Table;
3
4 @Entity
5 @Table(name="autor")
6 public class Autor {
7
8 }
```

A tabela **autor** será criada no momento que compilarmos nosso projeto.

62

JPA - Instalação e configuração

Criando os atributos da Entidade, iniciando pelo campo que será nosso identificador único e que ele será gerado automaticamente do tipo auto-incremento.

```
1 import javax.persistence.Entity;
2 import javax.persistence.GeneratedValue;
3 import javax.persistence.GenerationType;
4 import javax.persistence.Id;
5 import javax.persistence.Table;
6
7 @Entity
8 @Table(name="autor")
9 public class Autor {
10     @Id
11     @GeneratedValue(strategy=GenerationType.IDENTITY)
12     private int cod;
13 }
```

63

JPA - Instalação e configuração

Definindo outros campos de nossa Entidade.
Sempre seguindo a “**Notation**” Correta.

Importar também Column.

Já que Autor será um objeto que irá representar uma tabela de nosso banco, precisamos gerar os: **Gets and Sets** para todos atributos.

```
@Entity
@Table(name="autor")
public class Autor {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int cod;
    @Column
    private String nome;
    @Column
    private int idade;
    @Column
    private String matricula;

    //Gets and Sets
}
```

64

JPA - Instalação e configuração

Criando nossa Classe que irá realizar as operações de CRUD em nossa Entidade Autor.

Criar uma nova classe chamada **AutorController** (Padrão MVC).

Após isto iremos criar nossa entidade de gerenciamento.

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

public class AutorController {
    //Realiza conexão com o MySQL
    EntityManagerFactory emf;
    //Realizar as operações de CRUD
    EntityManager em;
}
```

65

JPA - Instalação e configuração

Iremos agora criar um **construtor** para que nesse seja criada uma sessão com o nosso banco.

```
public AutorController(){
    emf = Persistence.createEntityManagerFactory("autor");
    em = emf.createEntityManager();
}
```

Mesmo nome da nossa Unidade de Persistência do Persistencia.xml

```
<persistence-unit name="autor" transaction-type="RESOURCE_LOCAL">
```

Criando nosso gerenciador.

66

JPA - Instalação e configuração

Criando um método para salvar um autor em nosso banco.

```
public void salvar(Autor autor){ //Recebe um autor
    em.getTransaction().begin(); //Iniciar uma transação com o banco
    em.merge(autor); //Salva de fato o autor no banco
    em.getTransaction().commit(); //Confirma a transação
    em.close(); //Encerra.
}
```

67

JPA - Instalação e configuração

Agora iremos criar um método para remover um autor.

```
public void remover(Autor autor){

    em.getTransaction().begin(); //Recebe um autor
    //Query para realizar a exclusão do autor quando encontrar a matrícula.
    Query q = em.createNativeQuery("DELETE autor FROM autor WHERE matricula = "
                                    +autor.getMatricula());

    q.executeUpdate();
    em.getTransaction().commit(); //Confirma a transação
    em.close(); //Encerra
}
```

Importar Query de javax.persistence.

68

JPA - Instalação e configuração

Criar uma classe chamada **AutorTeste** para testar para inserir e um autor em nossa base de dados.

```
public static void main(String[] args){
    //Instanciando um objeto autor
    Autor autor = new Autor();
    //Passando alguns dados
    autor.setNome("Jose");
    autor.setIdade(28);
    autor.setMatricula("123456");

    // Instanciando um objeto de nossa conexão
    // Neste momento será criada a tabela na base.
    AutorController ac = new AutorController();

    //Passando um objeto autor com os dados
    //para o método salvar de nosso controller
    ac.salvar(autor);
}
```

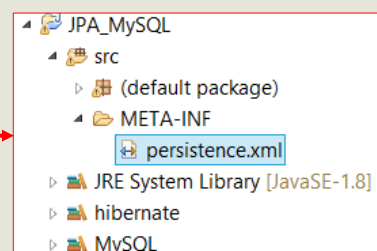
69

JPA - Instalação e configuração

ATENÇÃO – Caso este erro apareça.

```
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.annotations.Version).
log4j:WARN Please initialize the log4j system properly.
Exception in thread "main" javax.persistence.PersistenceException: No Persistence provider for EntityManager named autor
    at javax.persistence.Persistence.createEntityManagerFactory(Persistence.java:55)
    at javax.persistence.Persistence.createEntityManagerFactory(Persistence.java:33)
    at AutorController.<init>(AutorController.java:13)
    at AutorTeste.main(AutorTeste.java:12)
```

Basta recriar a pasta [META-INF] na raiz da pasta **src**. e copiar o arquivo de persistence.xml lá.



70

JPA - Instalação e configuração

```
mysql> use livraria;
Database changed
mysql> show tables;
+-----+
| Tables_in_livraria |
+-----+
| autor               |
| livros              |
+-----+
2 rows in set (0.00 sec)
```

Caso execute correto:

← Tabela autor

Dados inseridos →

```
mysql> select * from autor;
+----+-----+-----+-----+
| cod | idade | matricula | nome |
+----+-----+-----+-----+
| 1   | 28    | 123456    | Jose |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

Estrutura da Tabela autor →

```
mysql> desc autor;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| cod        | int(11)       | NO   | PRI | NULL    | auto_increment |
| idade      | int(11)       | YES  |     | NULL    |                |
| matricula  | varchar(255)  | YES  |     | NULL    |                |
| nome       | varchar(255)  | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.03 sec)
```

71

JPA - Instalação e configuração

Agora iremos apagar um autor de nossa tabela.

Comente a chamada do método salvar e chame o método remover.

```
//ac.salvar(autor);

// Passando um objeto para realizar a exclusão
// isto será apenas para o autor que estiver
// no estado do objeto.
ac.remover(autor);
```

Dados foram removidos

```
mysql> select * from autor;
Empty set (0.00 sec)
```

72

JPA - Instalação e configuração

Fazendo uma consulta no banco.

1º na classe AutorController no método salvar, comente a linha em.close(); para que seja possível add outros autores.

```
//em.close(); //Encerra.
```

Agora ajuste a classe AutorTeste como segue ao lado.

```
Autor autor = new Autor();
autor.setNome("Jose");
autor.setIdade(28);
autor.setMatricula("123456");

Autor autor1 = new Autor();
autor1.setNome("Maria");
autor1.setIdade(31);
autor1.setMatricula("1234567");

Autor autor2 = new Autor();
autor2.setNome("Pedro");
autor2.setIdade(25);
autor2.setMatricula("12345678");

AutorController ac = new AutorController();

ac.salvar(autor);
ac.salvar(autor1);
ac.salvar(autor2);
```

73

JPA - Instalação e configuração

Verificando no banco, podemos ver como ficou.

```
mysql> select * from autor;
+----+-----+-----+-----+
| cod | idade | matricula | nome |
+----+-----+-----+-----+
| 1   | 28    | 123456    | Jose |
| 2   | 31    | 1234567   | Maria |
| 3   | 25    | 12345678  | Pedro |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

74

JPA - Instalação e configuração

Fazendo uma consulta para retornar dados do banco.

Na classe AutorController, crie um método que retorne uma lista contendo os dados da tabela Autor.

Atenção para a SQL

```
public List<Autor> listar(){
    em.getTransaction().begin();
    //Faz a consulta no banco.
    Query query = em.createQuery("SELECT autor FROM Autor autor");
    //Atribui o resultado da consulta para uma lista
    List<Autor> lista = query.getResultList();
    em.getTransaction().commit(); //Confirma a transação
    em.close(); //Encerra
    return lista; // Retorna a lista
}
```

75

JPA - Instalação e configuração

Voltando para a classe AutorTeste, implemente dentro do método main o seguinte código para listar os autores no console:

```
AutorController ac = new AutorController();

List<Autor> autores = ac.listar();

for (int i = 0; i < autores.size(); i++) {
    System.out.println("Código: " + autores.get(i).getCod());
    System.out.println("Nome: " + autores.get(i).getNome());
    System.out.println("Idade: " + autores.get(i).getIdade());
    System.out.println("Matricula: " + autores.get(i).getMatricula());
    System.out.println("-----");
}
```

76

JPA - Instalação e configuração

Como resultado teremos no console:

```
Código: 1
Nome: Jose
Idade: 28
Matricula: 123456
-----
```

Observe que o código foi gerado automaticamente através da definição do atributo cod da classe Autor.

```
Código: 2
Nome: Maria
Idade: 31
Matricula: 1234567
-----
```

```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private int cod;
```

```
Código: 3
Nome: Pedro
Idade: 25
Matricula: 12345678
-----
```

77

Dúvidas?



Estácio



78