

# Introdução ao Desenvolvimento Web



---

**INTRODUÇÃO AO HIBERNATE E SQL COM  
JAVA**

**PROF. PABLO VARGAS**

# Tópicos Abordados



- Persistência de dados
- JDBC
- Hibernate

# Persistência de dados



- Aplicações que precisam de dados persistentes.
  - O processo de armazenamento de dados é também chamado de **persistência**.
- Dados de valor considerado importante para o negócio devem ser armazenados.
- Em JAVA, consiste em manipular dados em um banco de dados relacional através de recursos da linguagem.
- A comunicação com os dados é realizada através do JDBC (Java Database Connectivity).

# JDBC

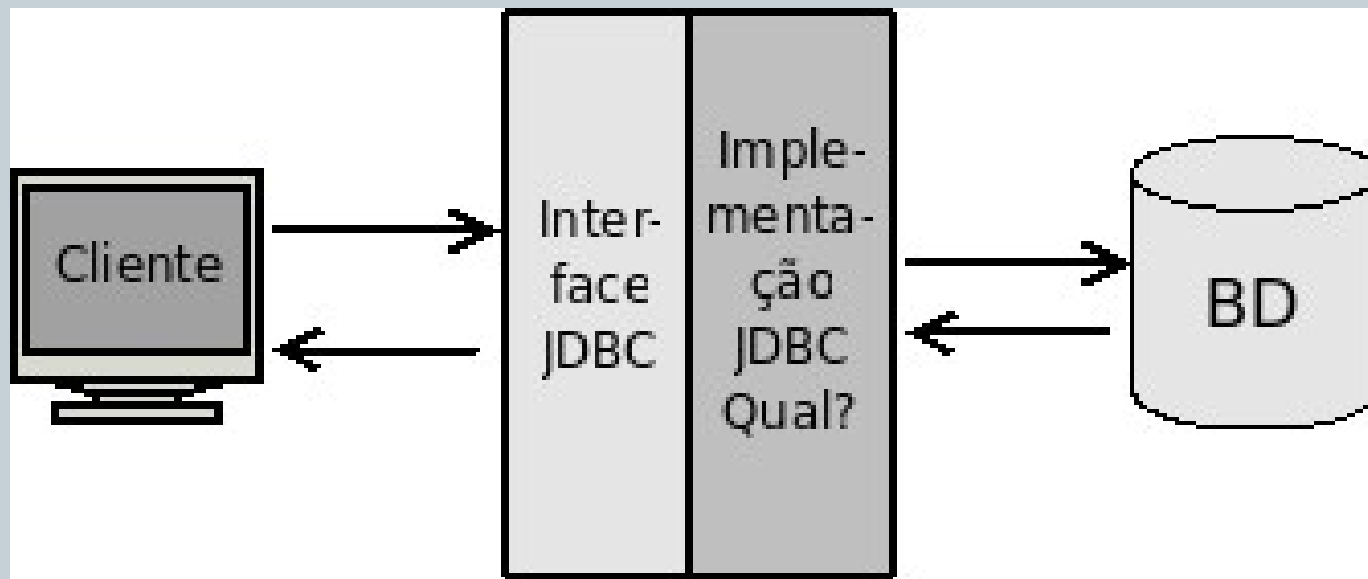


- É a biblioteca de persistência em banco de dados relacionais do Java.
- É a especificação de como ocorre a comunicação com o BD.
- Atraves dessa especificação, usuários de BD criam arquivos de extensão .jar que facilitam na configuração e manipulação de aplicações JAVA.
- Esses arquivos, possuem classes que implementam conexões e configurações.

# JDBC



- Conexão JAVA possui um conjunto de interfaces bem definidas localizado no pacote “**java.sql**”.

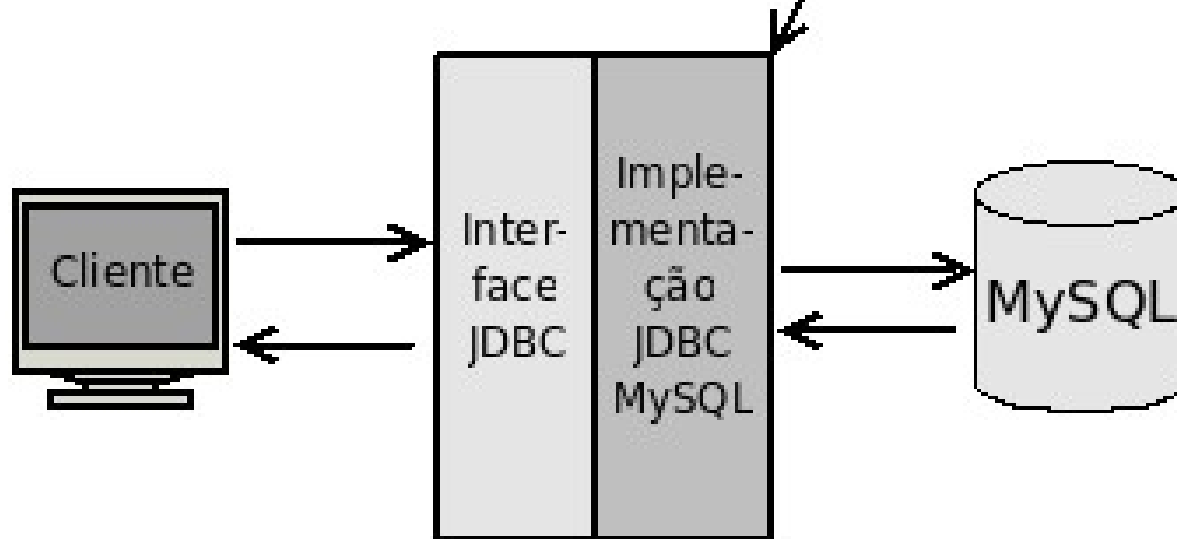


# JDBC



- Todos os principais bancos de dados do mercado possuem **drivers JDBC** para que você possa utilizá-los com Java.

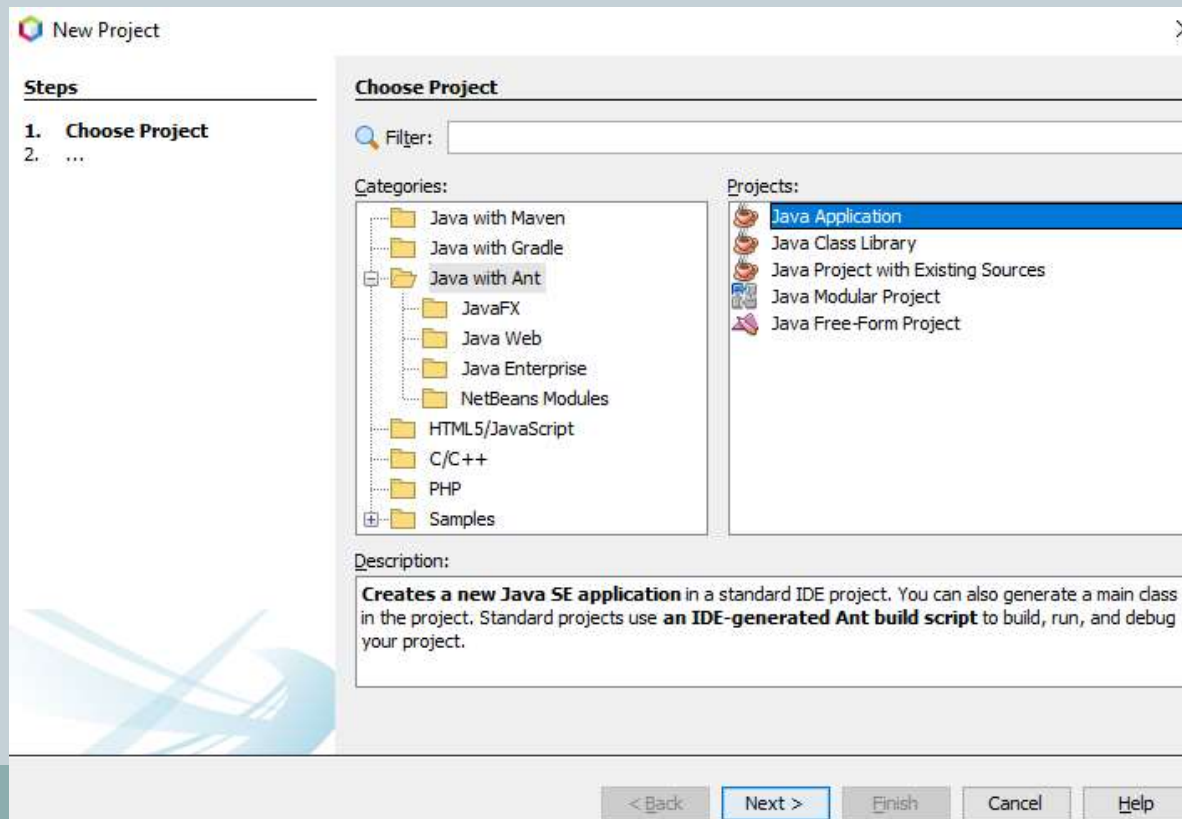
```
DriverManager.getConnection("jdbc:mysql://localhost/teste");
```



# JDBC



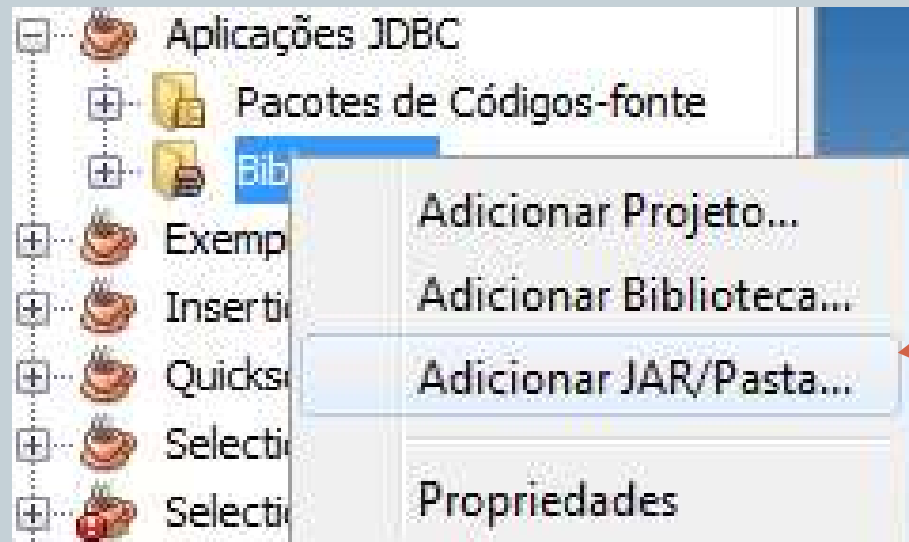
- Utilizando o JDBC junto com o NetBeans:
  - Aperte no teclado “Ctrl+Shift+n” e crie uma aplicação JAVA com o nome “Aplicações JDBC”.



# JDBC



- Adicione o arquivo JAR do JDBC ao projeto.

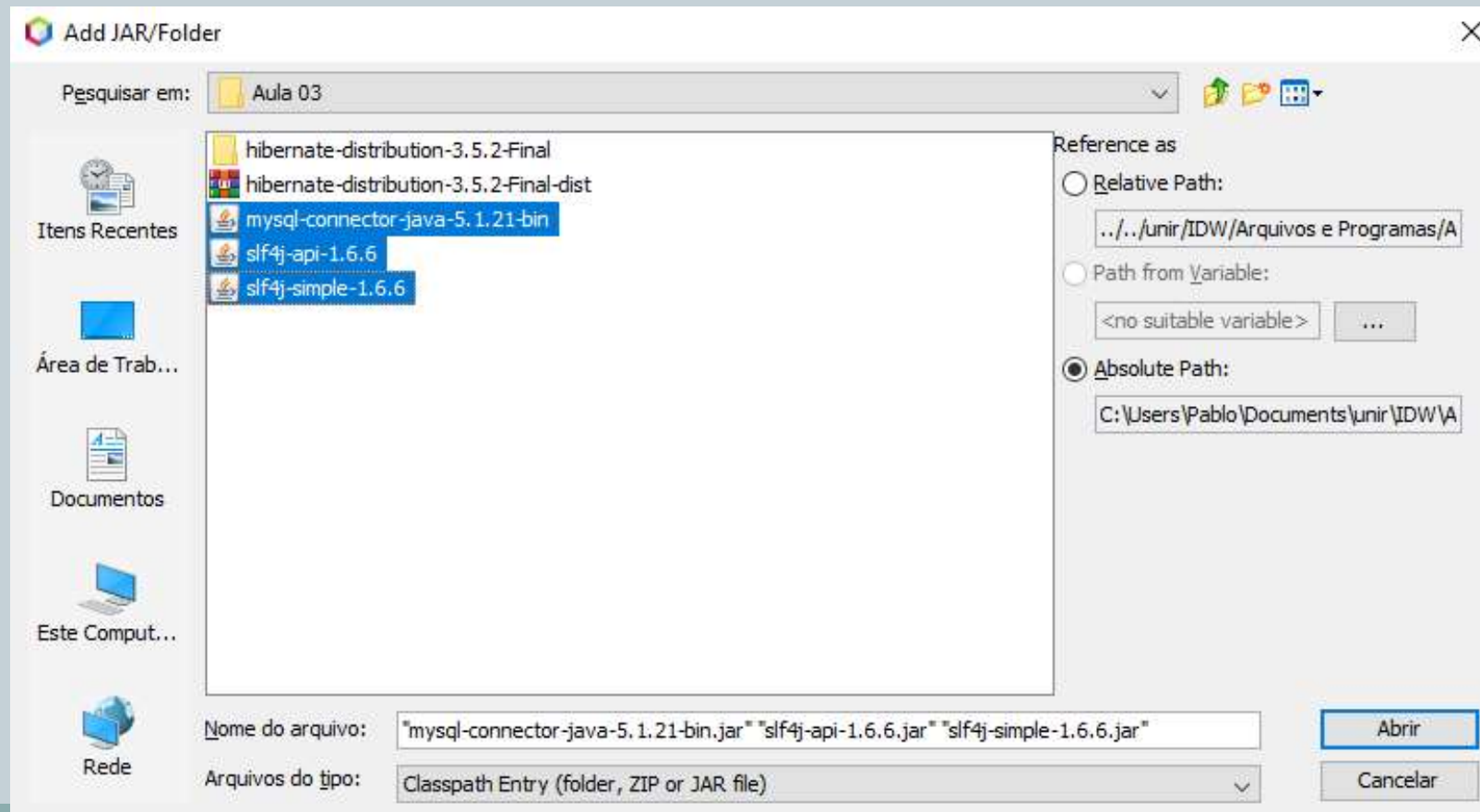




# JDBC



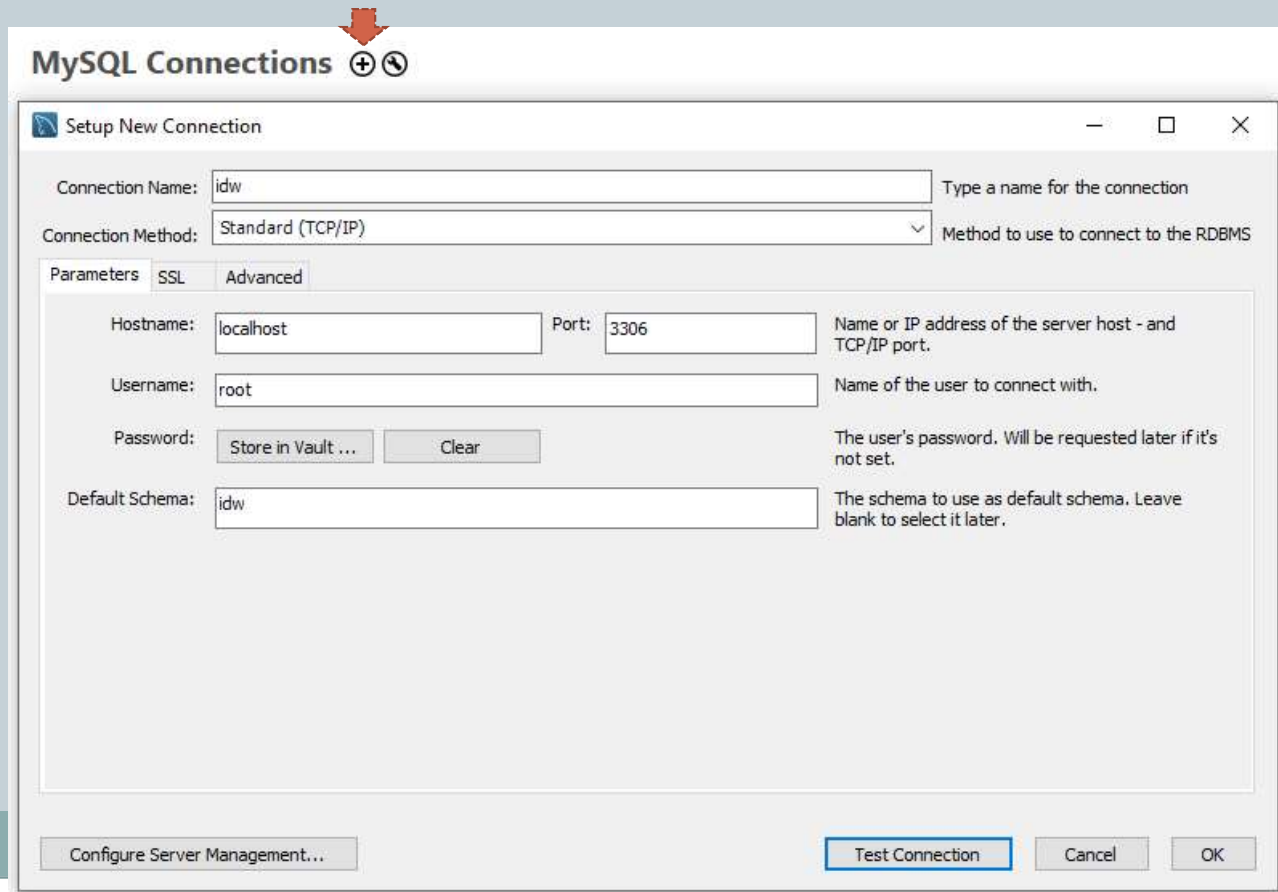
- Adicione o arquivo JAR do JDBC ao projeto:



# JDBC



- Abra o MySQL Workbench.
- Crie uma nova conexão MySQL e abra ela:



# JDBC



- Crie o BD “agenda” no MySQL :

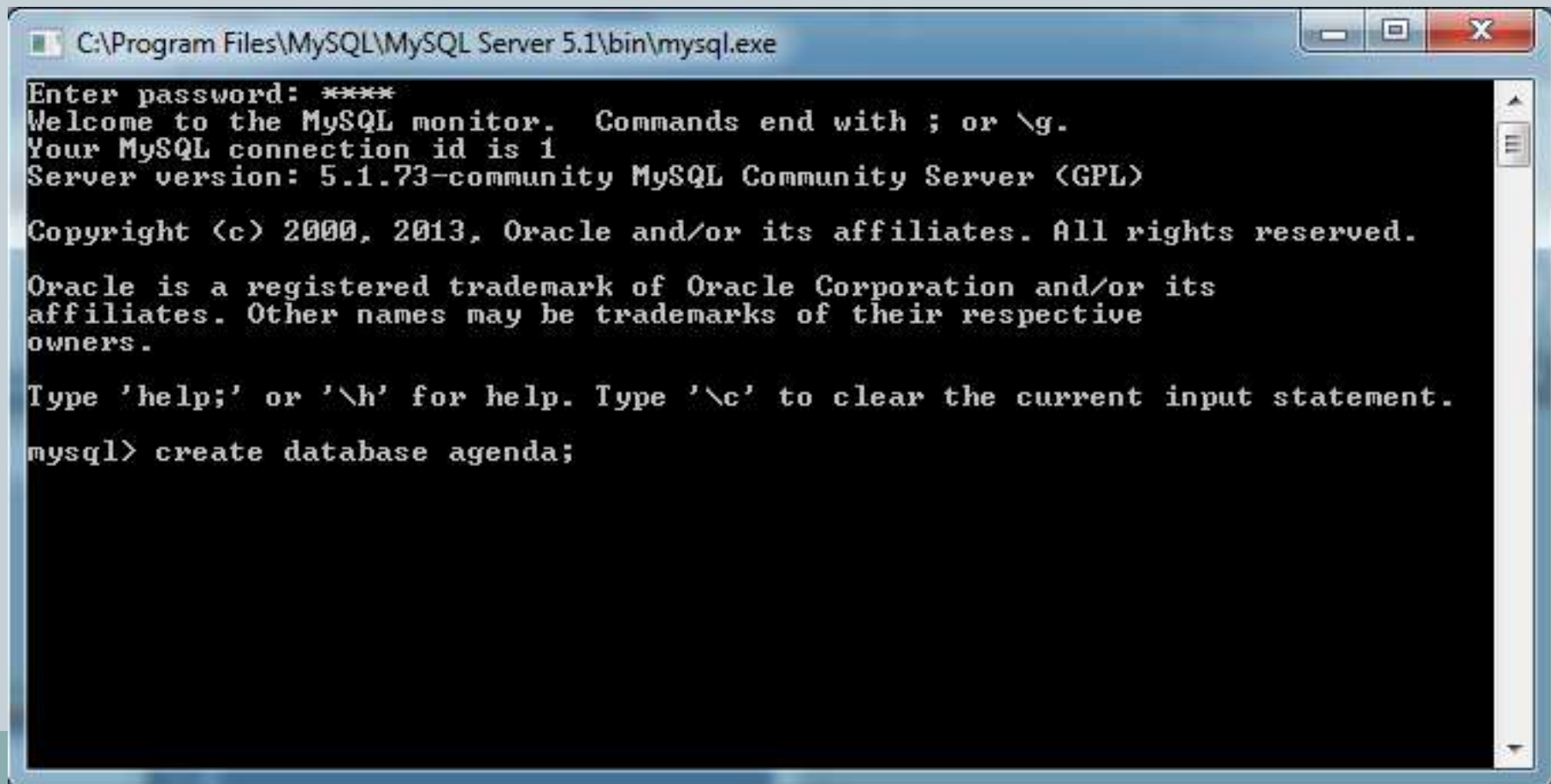
Clique aqui para executar



# JDBC



- Crie o BD Agenda no MySQL via linha de comando:

A screenshot of a Windows command prompt window titled 'C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe'. The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The text inside the window is as follows:

```
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.73-community MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

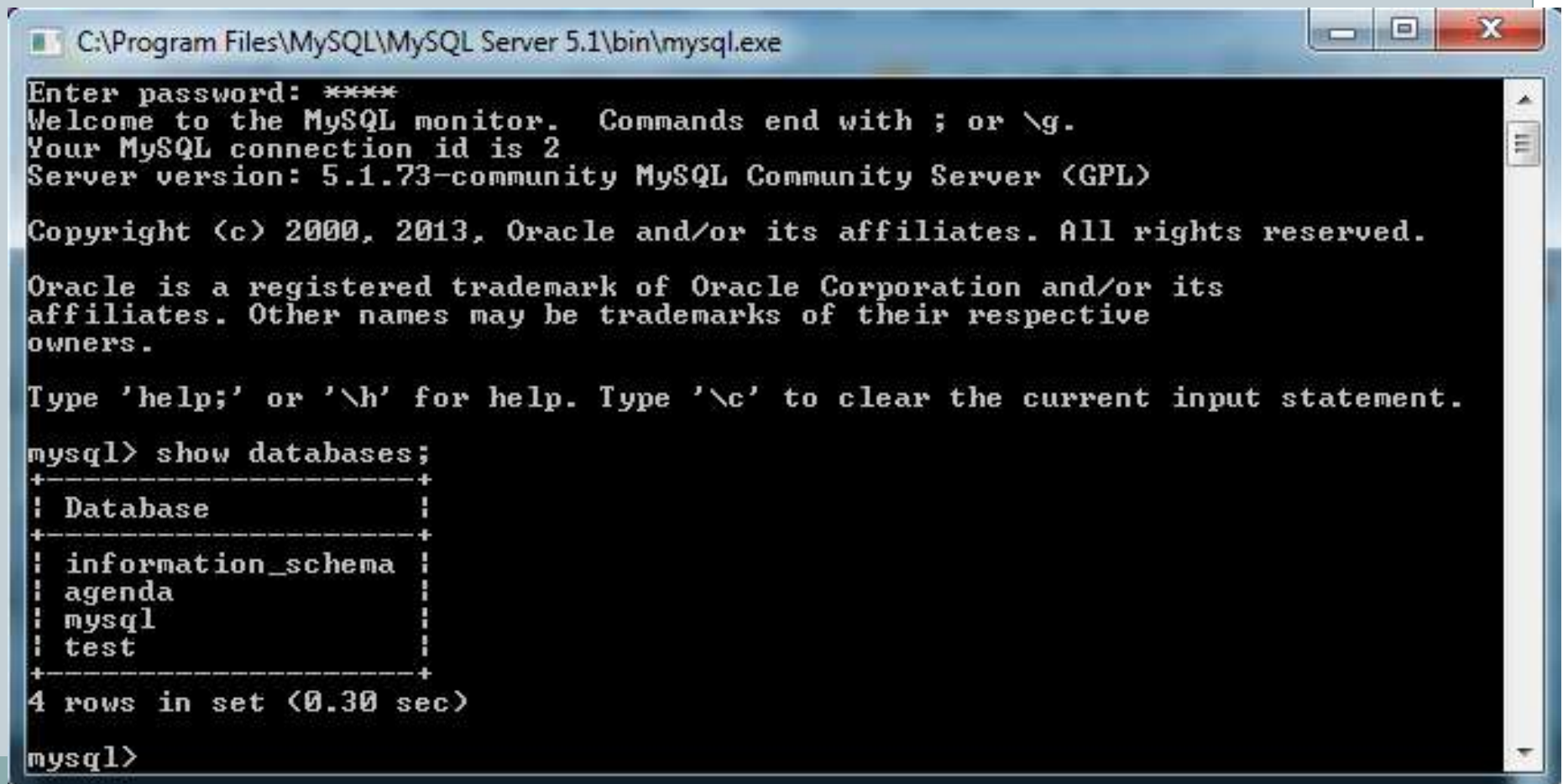
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database agenda;
```

# JDBC



- Verifique se o BD Agenda foi criado no MySQL:



```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.73-community MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

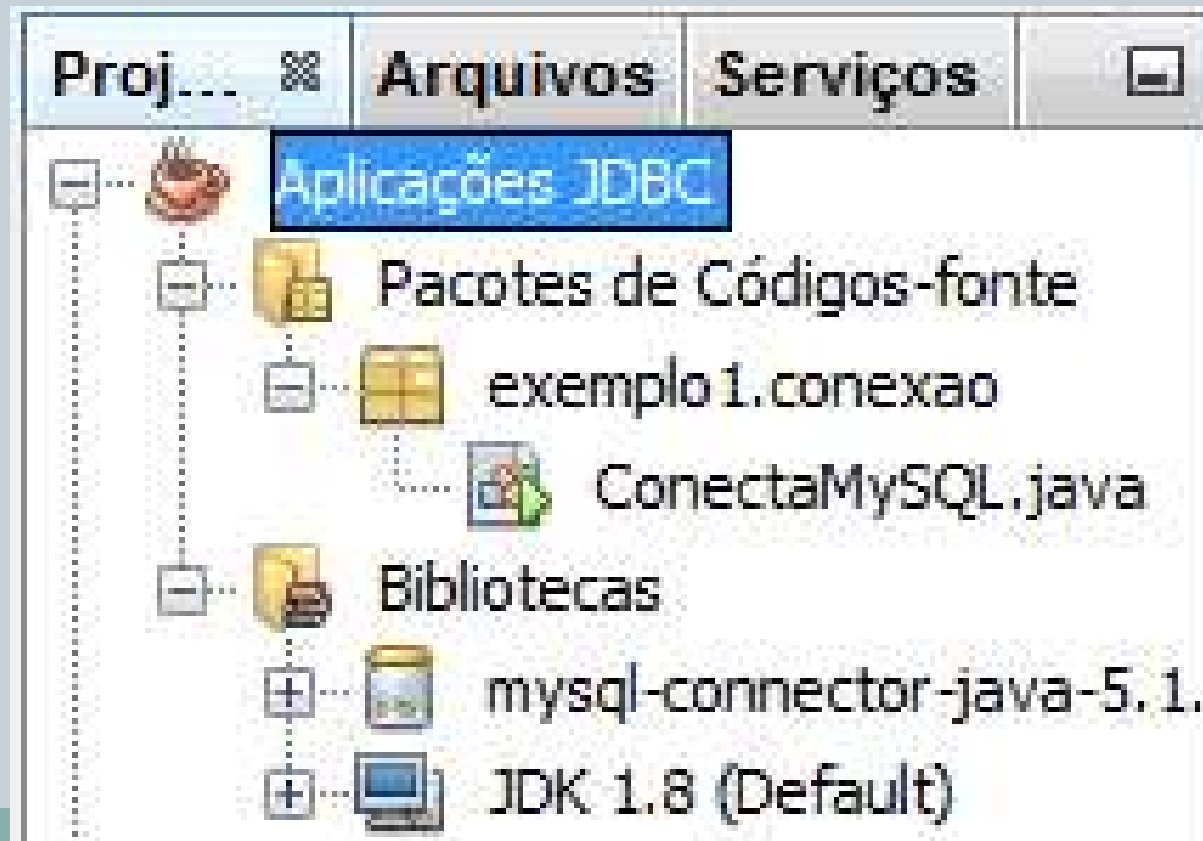
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| agenda    |
| mysql     |
| test      |
+-----+
4 rows in set (0.30 sec)

mysql>
```

# JDBC



- Adicione a classe “ConectaMySQL” no pacote “exemplo1.conexao”



# JDBC



- Insira o código abaixo na classe e rode a aplicação :

```
public class ConectaMySQL {  
    public static void main(String[] args) throws SQLException {  
        Connection conexao = null;  
  
        try {  
            // Registrando a classe JDBC no sistema em tempo de execução  
            Class.forName("com.mysql.jdbc.Driver");  
            String url = "jdbc:mysql://localhost/agenda";  
            String usuario = "root";  
            String senha = "root";  
  
            conexao = DriverManager.getConnection(url, usuario, senha);  
            System.out.println("Conectou!");  
        } catch (ClassNotFoundException e) {  
            System.out.println("Classe não encontrada. Erro: " + e.getMessage());  
        } catch (SQLException e) {  
            System.out.println("Ocorreu um erro de SQL. Erro: " + e.getMessage());  
        } finally {  
            try {  
                conexao.close();  
            } catch (SQLException e) {  
                System.out.println("Erro ao fechar a conexão. Erro: " + e.getMessage());  
            }  
        }  
    }  
}
```

# JDBC



- Se tudo ocorre bem, a mensagem “Conectou!” deve aparecer na tela.
- Todo o processo passa pela classe “DriverManager”.
- O método “getConnection()”, através dos parâmetros passados, é responsável pela conexão com o BD.



# JDBC



- **Erros que ocorrem com o JDBC:**
  - `ClassNotFoundException`: projeto não encontrou o arquivo jar do JDBC.
  - `SQLException` – No suitable driver found for: Pode ser que o parâmetro da URL esteja errado ou o driver não está no classpath do Projeto.
  - Nome do banco incorreto.
  - A porta do BD.

# JDBC



- Operações com JDBC:
  - Defina o BD que está sendo utilizado para as operações.

```
C:\Program Files\MySQL\MySQL Server 5.1\bin\mysql.exe
Your MySQL connection id is 2
Server version: 5.1.73-community MySQL Community Server (GPL)
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

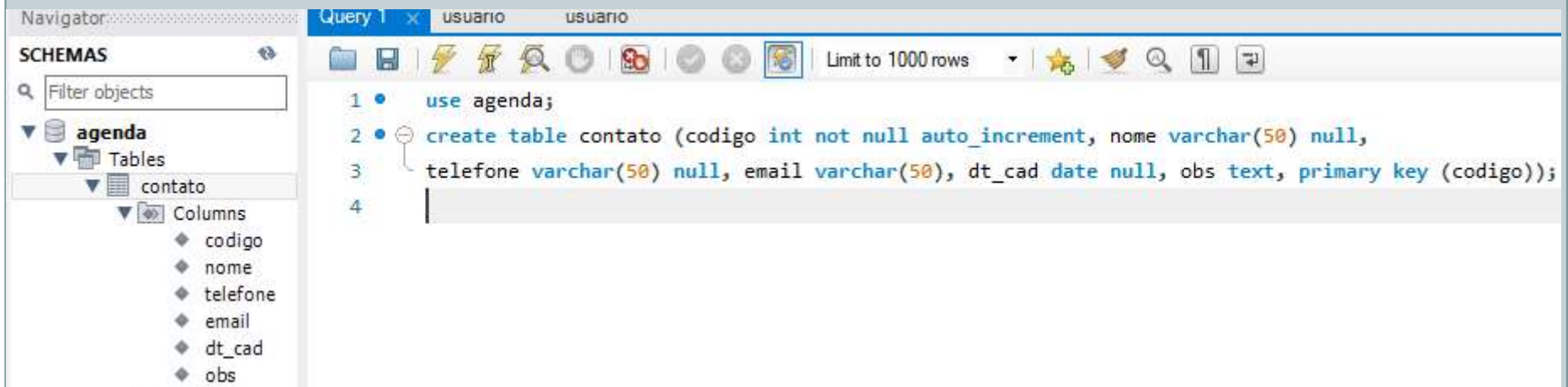
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| agenda    |
| mysql     |
| test      |
+-----+
4 rows in set (0.30 sec)

mysql> USE agenda
Database changed
mysql>
```

# JDBC



- Crie a tabela contato no BD “agenda”.



# JDBC



- Crie a classe “Contato” no pacote “exemplo1.crudjdbc”.

New Java Class

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name: Contato

Project: Aplicações JDBC

Location: Source Packages

Package: exemplo1.crudjdbc

Created File: pablo\Documents\NetBeansProjects\Aplicações JDBC\src\exemplo1\crudjdbc\Contato.java

< Back Next > **Finish** Cancel Help

# JDBC



- Insira o código abaixo na classe Contato.
  - Obs: Não esqueça de gerar os getters e setters.

```
public class Contato {  
  
    private Integer codigo;  
    private String nome, telefone, email, observacao;  
    private Date dataCadastro;  
  
    //Aperte alt+insert e gere os getters e setters  
  
}
```

# JDBC



- Crie a classe “ContatoCrudJDBC” no pacote “exemplo1.crudjdbc”.

**New Java Class**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name: ContatoCRUD JDBC

Project: Aplicações JDBC

Location: Source Packages

Package: exemplo1.crudjdbc

Created File: ...\\src\\exemplo1\\crudjdbc\\ContatoCRUD\_JDBC.java

< Back Next > Finish Cancel Help

# JDBC



- Insira o código do método “salvar” dentro da nova classe ContatoCRUD\_JDBC.java.
- Obs: Aceite as importações recomendadas.

```
public void salvar(Contato contato) {  
    Connection conexao = this.geraConexao();  
    PreparedStatement insereSt = null;  
  
    String sql = "insert into contato(nome, telefone, email, dt_cad, obs) values(?, ?, ?, ?, ?)";  
  
    try {  
        insereSt = conexao.prepareStatement(sql);  
        insereSt.setString(1, contato.getNome());  
        insereSt.setString(2, contato.getTelefone());  
        insereSt.setString(3, contato.getEmail());  
        insereSt.setDate(4, contato.getDataCadastro());  
        insereSt.setString(5, contato.getObservacao());  
        insereSt.executeUpdate();  
    } catch (SQLException e) {  
        System.out.println("Erro ao incluir contato. Mensagem: " + e.getMessage());  
    } finally {  
        try {  
            insereSt.close();  
            conexao.close();  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operações de inserção. Mensagem: " + e.getMessage());  
        }  
    }  
}
```

# JDBC



- Insira o código do método “atualizar”.

```
public void atualizar(Contato contato) {
    Connection conexao = this.geraConexao();
    PreparedStatement atualizaSt = null;

    //Aqui não atualizamos o campo data de cadastro
    String sql = "update contato set nome=?, telefone=?, email=?, obs=? where codigo=?";

    try {
        atualizaSt = conexao.prepareStatement(sql);
        atualizaSt.setString(1, contato.getNome());
        atualizaSt.setString(2, contato.getTelefone());
        atualizaSt.setString(3, contato.getEmail());
        atualizaSt.setString(4, contato.getObservacao());
        atualizaSt.setInt(5, contato.getCodigo().intValue());
        atualizaSt.executeUpdate();
    } catch (SQLException e) {
        System.out.println("Erro ao atualizar contato. Mensagem: " + e.getMessage());
    } finally {
        try {
            atualizaSt.close();
            conexao.close();
        } catch (Throwable e) {
            System.out.println("Erro ao fechar operações de atualização. Mensagem: " + e.getMessage());
        }
    }
}
```



# JDBC



- Insira o código do método “excluir”.

```
public void excluir(Contato contato) {  
    Connection conexao = this.geraConexao();  
    PreparedStatement excluiSt = null;  
  
    String sql = "delete from contato where codigo = ?";  
  
    try {  
        excluiSt = conexao.prepareStatement(sql);  
        excluiSt.setInt(1, contato.getCodigo().intValue());  
        excluiSt.executeUpdate();  
    } catch (SQLException e) {  
        System.out.println("Erro ao excluir contato. Mensagem: " + e.getMessage());  
    } finally {  
        try {  
            excluiSt.close();  
            conexao.close();  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operações de exclusão. Mensagem: " + e.getMessage());  
        }  
    }  
}
```

# JDBC



- Abaixo é mostrado o código do método “listar”.

```
public List<Contato> listar() {  
    Connection conexao = this.geraConexao();  
    List<Contato> contatos = new ArrayList<Contato>();  
    Statement consulta = null;  
    ResultSet resultado = null;  
    Contato contato = null;  
    String sql = "select * from contato";  
    try {  
        consulta = conexao.createStatement();  
        resultado = consulta.executeQuery(sql);  
        while (resultado.next()) {  
            contato = new Contato();  
            contato.setCodigo(new Integer(resultado.getInt("codigo")));  
            contato.setNome(resultado.getString("nome"));  
            contato.setTelefone(resultado.getString("telefone"));  
            contato.setEmail(resultado.getString("email"));  
            contato.setDataCadastro(resultado.getDate("dt_cad"));  
            contato.setObservacao(resultado.getString("obs"));  
            contatos.add(contato);  
        }  
    } catch (SQLException e) {  
        System.out.println("Erro ao buscar código do contato. Mensagem: " + e.getMessage());  
    }  
}
```

# JDBC



```
    } finally {  
        try {  
            consulta.close();  
            resultado.close();  
            conexao.close();  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operações de consulta. Mensagem: " + e.getMessage());  
        }  
    }  
    return contatos;  
}
```

# JDBC



- Insira o código abaixo para buscar um contato :

```
public Contato buscaContato(int valor) {
    Connection conexao = this.geraConexao();
    PreparedStatement consulta = null;
    ResultSet resultado = null;
    Contato contato = null;

    String sql = "select * from contato where codigo = ?";

    try {
        consulta = conexao.prepareStatement(sql);
        consulta.setInt(1, valor);
        resultado = consulta.executeQuery();

        if (resultado.next()) {
            contato = new Contato();
            contato.setCodigo(new Integer(resultado.getInt("codigo")));
            contato.setNome(resultado.getString("nome"));
            contato.setTelefone(resultado.getString("telefone"));
            contato.setEmail(resultado.getString("email"));
            contato.setDataCadastro(resultado.getDate("dt_cad"));
            contato.setObservacao(resultado.getString("obs"));
        }
    } catch (SQLException e) {
        System.out.println("Erro ao buscar código do contato. Mensagem: " + e.getMessage());
    }
}
```

# JDBC



```
} finally {  
    try {  
        consulta.close();  
        resultado.close();  
        conexao.close();  
    } catch (Throwable e) {  
        System.out.println("Erro ao fechar operações de consulta. Mensagem: " + e.getMessage());  
    }  
}  
return contato;
```

# JDBC



- Crie a função “geraConexao()”.

```
public Connection geraConexao() {  
    Connection conexao = null;  
  
    try {  
        // Registrando a classe JDBC no sistema em tempo de execução  
        Class.forName("com.mysql.jdbc.Driver");  
        String url = "jdbc:mysql://localhost/agenda";  
        String usuario = "root";  
        String senha = "root";  
        conexao = DriverManager.getConnection(url, usuario, senha);  
    } catch (ClassNotFoundException e) {  
        System.out.println("Classe não encontrada. Erro: " + e.getMessage());  
    } catch (SQLException e) {  
        System.out.println("Ocorreu um erro de SQL. Erro: " + e.getMessage());  
    }  
    return conexao;  
}
```



# JDBC



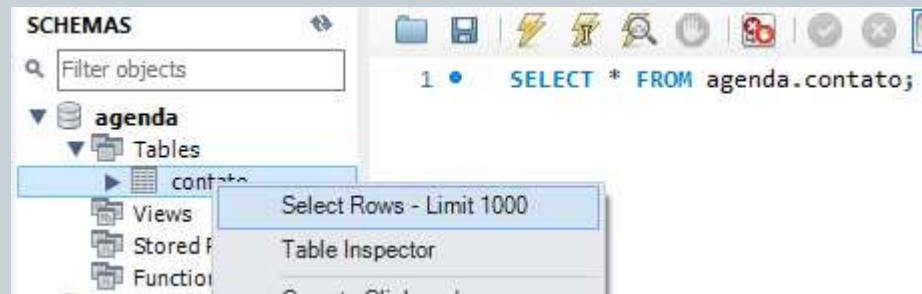
- Crie a função principal do código.

```
public static void main(String[] args) {  
    ContatoCRUD_JDBC contatoCRUD_JDBC = new ContatoCRUD_JDBC();  
    Contato beltrano = new Contato();  
    beltrano.setNome("Pablo Nunes Vargas");  
    beltrano.setDataCadastro(new Date(System.currentTimeMillis()));  
    beltrano.setEmail("eng.pnv@gmail.com");  
    beltrano.setTelefone("(69) 99999-9999");  
    beltrano.setObservacao("novo cliente");  
    contatoCRUD_JDBC.salvar(beltrano);  
  
    Contato fulano = new Contato();  
    fulano.setNome("João da Silva");  
    fulano.setDataCadastro(new Date(System.currentTimeMillis()));  
    fulano.setEmail("joao@gmail.com");  
    fulano.setTelefone("(69) 99999-9999");  
    fulano.setObservacao("cliente antigo");  
    contatoCRUD_JDBC.salvar(fulano);  
  
    System.out.println("Contatos cadastrados: " + contatoCRUD_JDBC.listar().size());  
}
```

# JDBC



- Execute a aplicação e verifique se os contatos foram cadastrados no MySQL.



	codigo	nome	telefone	email	dt_cad	obs
▶	1	Pablo Nunes Vargas	(69) 99999-9999	eng.pnv@gmail.com	2021-03-04	novo cliente
	2	João da Silva	(69) 99999-9999	joao@gmail.com	2021-03-04	cliente antigo
•	NULL	NULL	NULL	NULL	NULL	NULL



# Atividades



- Apague todos os contatos cadastrados.
- Crie o contato com seus dados.
- Atualize um dado qualquer do seu contato.
- Crie um novo atributo(CPF) na classe Contato.java e atualize seu contato com esse novo atributo.

# DAO - DATA ACCESS OBJECT



- Colocar código SQL dentro de suas classes de lógica é algo nem um pouco elegante e muito menos viável quando você precisa manter o seu código.
- É capaz de isolar todo o acesso a banco em classes bem simples, cuja instância é um **objeto** responsável por **acessar** os **dados**.

# ORM



- ORM – Object Relational Mapping(Mapeamento Objeto-Relacional).
- “...uma forma automatizada e transparente de persistir objetos que pertencem a uma aplicação nas respectivas tabelas em um banco relacional...” (Luckow)
- Transforma dados de uma forma para outra de maneira reversível.
- Uma solução ORM, deve conter os seguintes elementos:
  - Uma API para realizar as operações CRUD.
  - Uma linguagem ou API para especificar consultas às classes ou suas propriedades.
  - Facilidade de especificar o metadado de mapeamento.
  - Verificações de leitura suja ou de carregamento sob demanda.
- <https://www.youtube.com/watch?v=2jod22kRohE>

# Hibernate



- O Hibernate é uma das soluções ORM.
- Ferramenta utilizada para realizar o mapeamento de objeto/relacional de forma completa.
- Segue especificação da JPA (Java Persistence API).
  - Uma parte da especificação EJB (Enterprise Java Beans).
  - Essa especificação trata de entidades, mapeamentos, interfaces para gerenciar a persistência e linguagem de consulta.
- Outras soluções ORM: TopLink (Oracle) e OpenJPA (Apache).
- <https://www.youtube.com/watch?v=HdxQErYWaJc>

# Hibernate



- Instalando o Hibernate:
  - Crie um novo projeto do tipo Java Application com o nome “Aplicações Hibernate”.

New Java Application

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name: Aplicações Hibernate

Project Location: C:\Users\Pablo\Documents\NetBeansProjects Browse...

Project Folder: :Users\Pablo\Documents\NetBeansProjects\Aplicações Hibernate

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder: Browse...

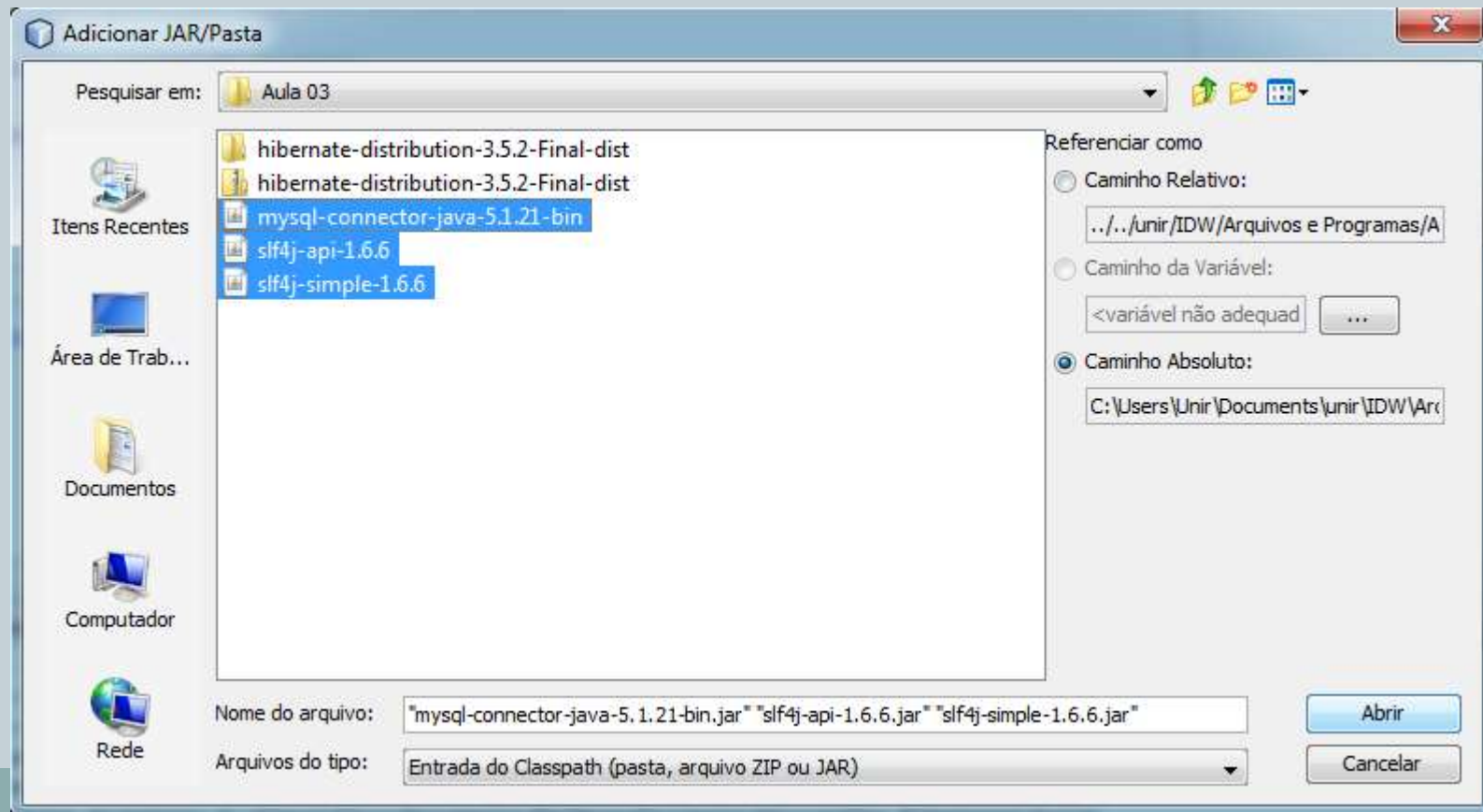
Different users and projects can share the same compilation libraries (see Help for details).

☒ Create Main Class: aplicações.hibernate.AplicaçõesHibernate

< Back Next > Finish Cancel Help

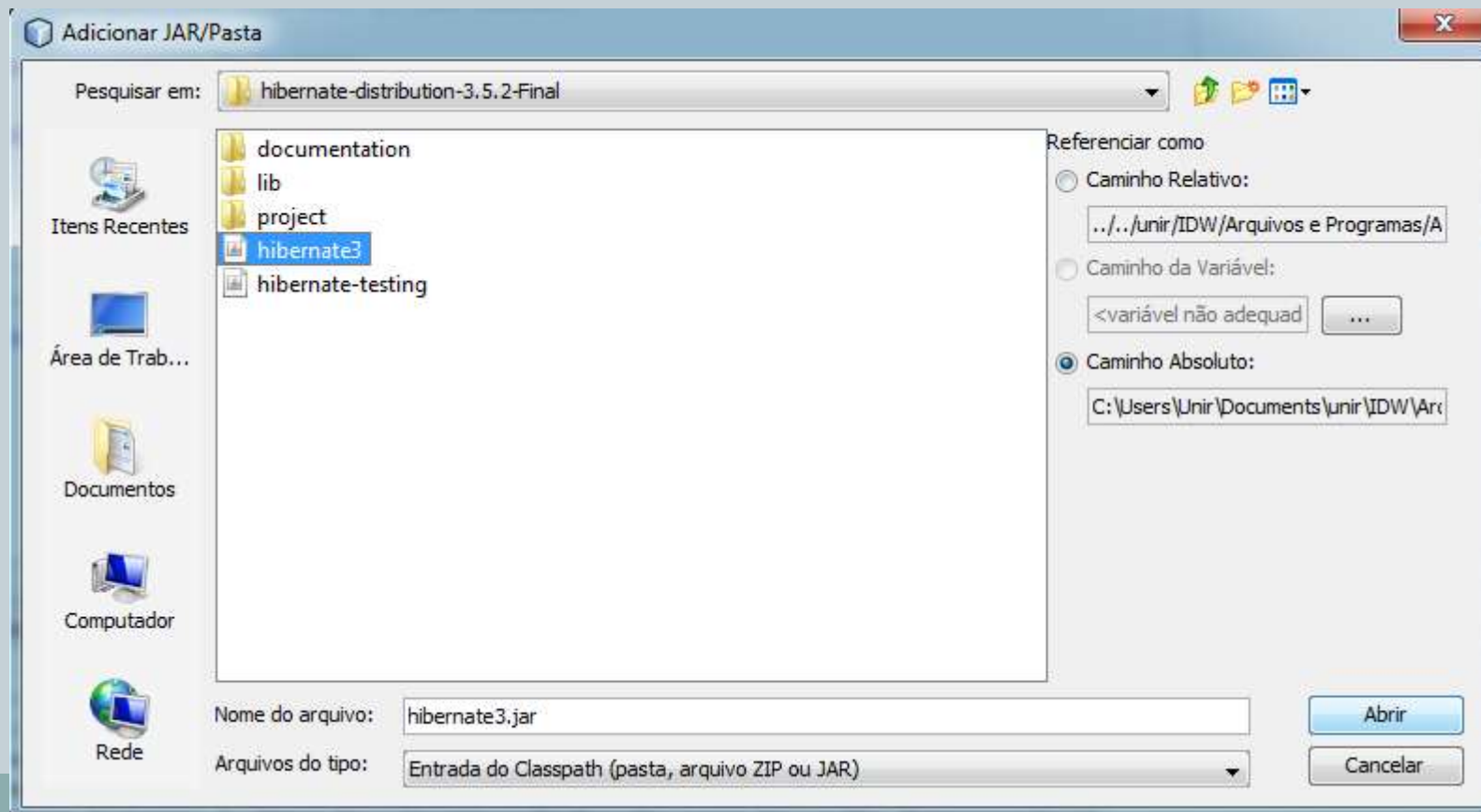
# Hibernate

- Instalando o Hibernate:
  - Adicione as jars abaixo localizadas na pasta “Aula 03”.



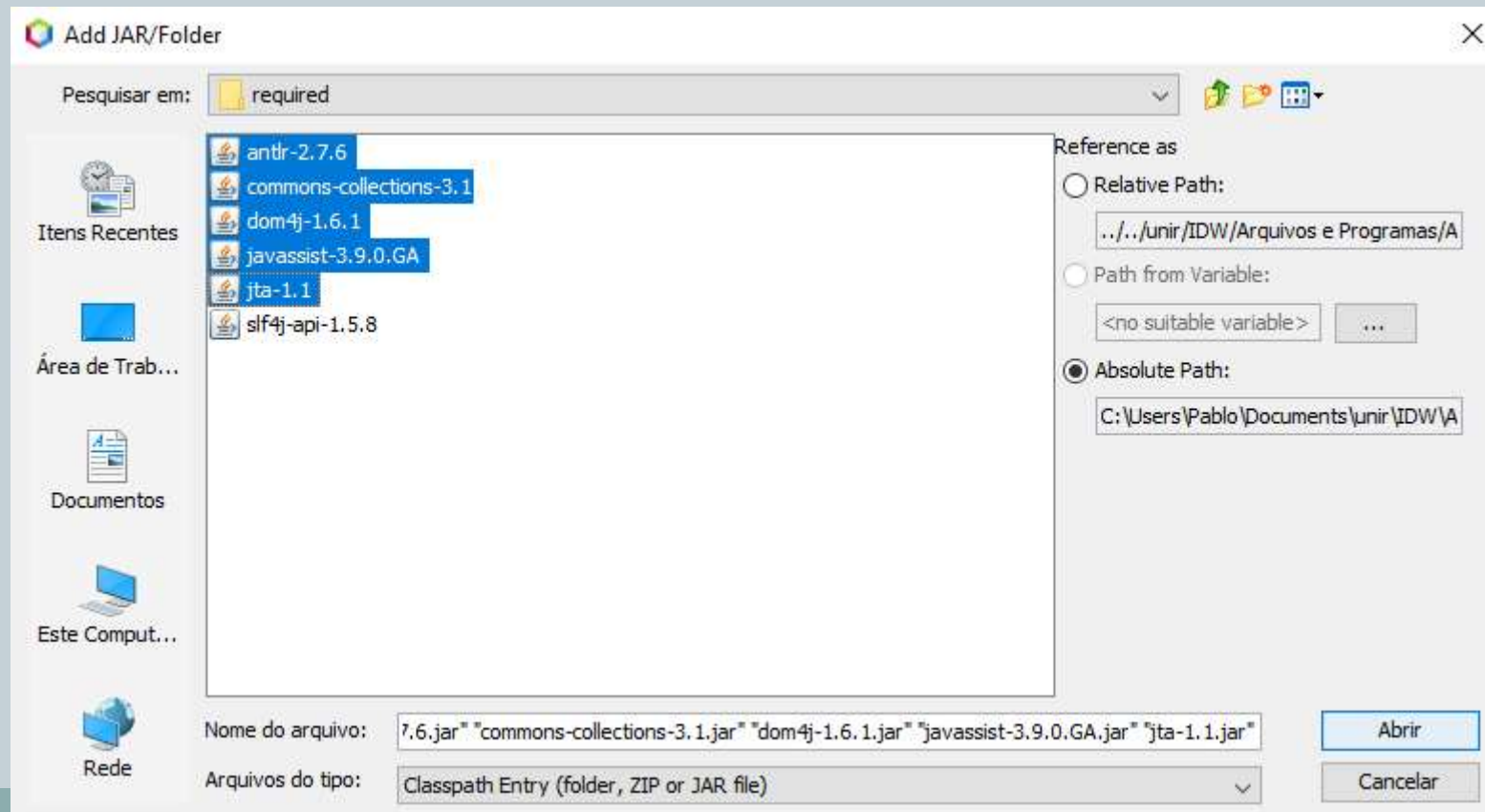
# Hibernate

- Instalando o Hibernate:
  - Extraí o arquivo zip do Hibernate e adicione a jar abaixo.



# Hibernate

- Instalando o Hibernate:
  - Adicione também as jars da pasta “required” do Hibernate.

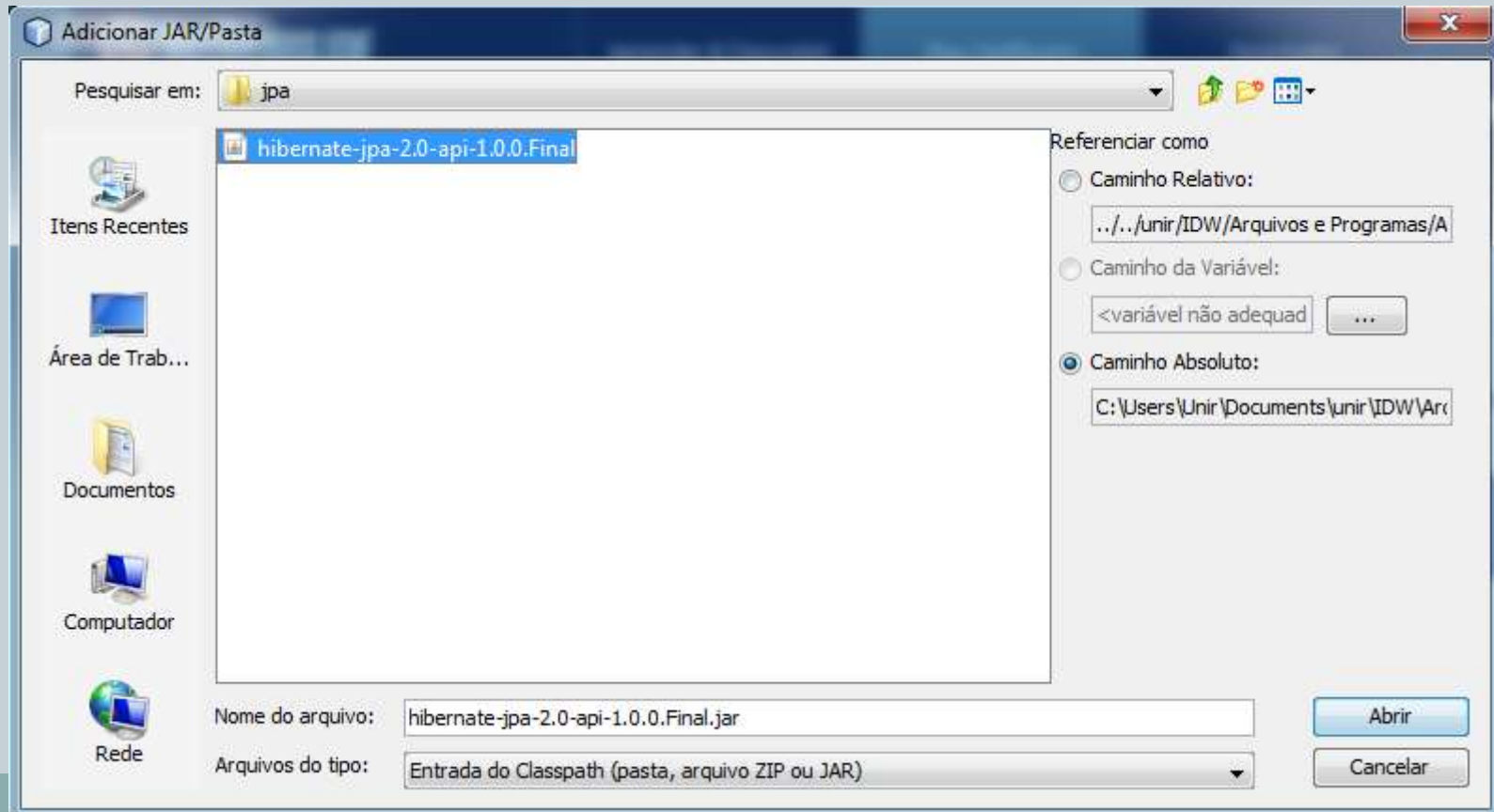




# Hibernate



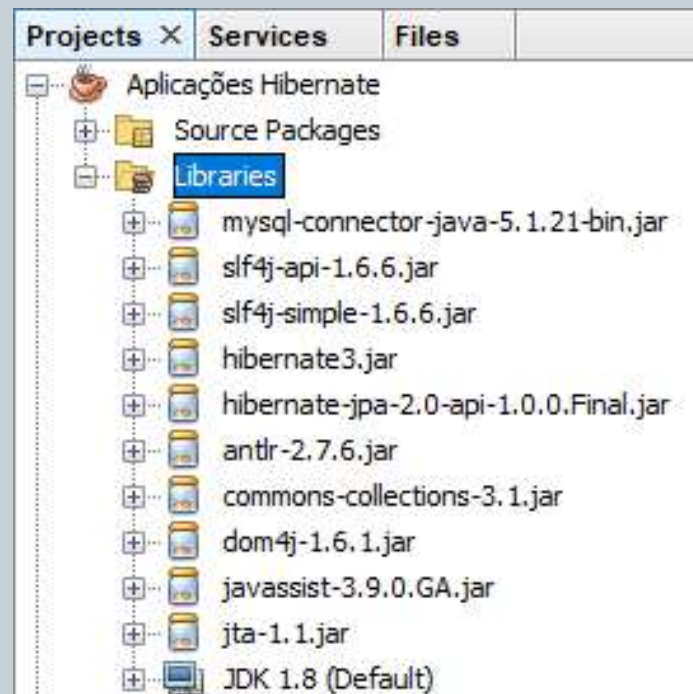
- Instalando o Hibernate:
  - Adicione também a jar da pasta “jpa” do Hibernate.



# Hibernate



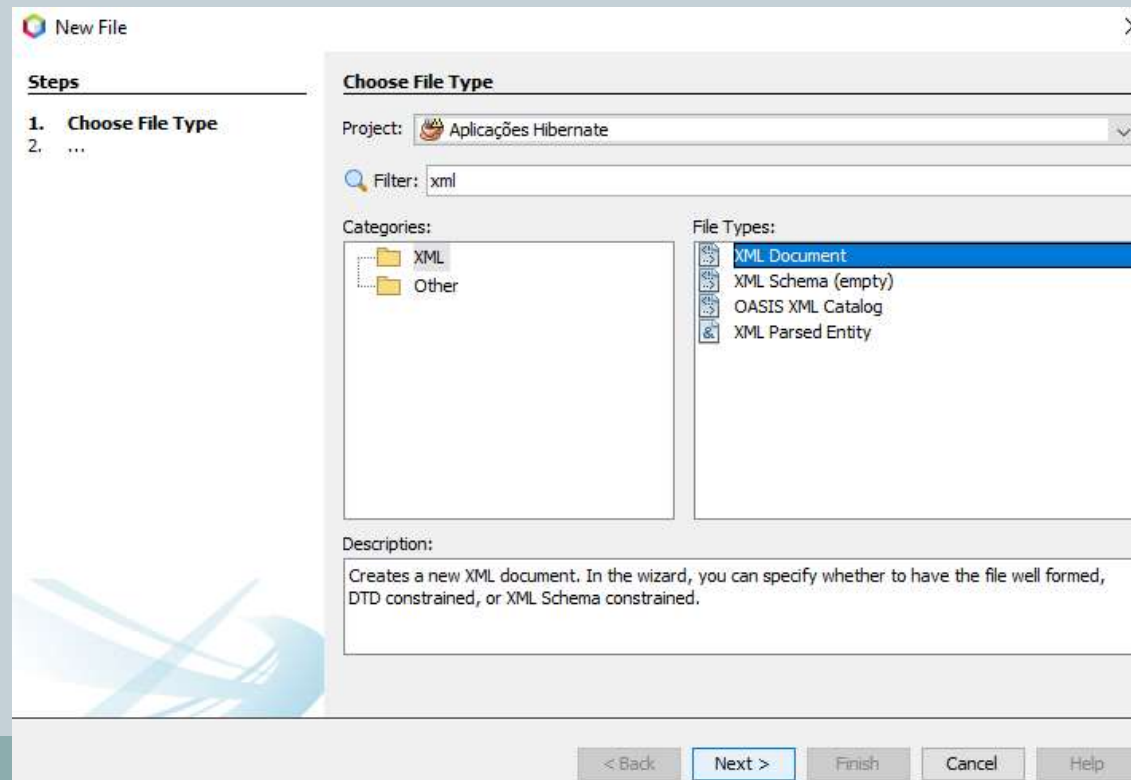
- Instalando o Hibernate:
  - No final a pasta “Bibliotecas” deve ter os seguintes jars.



# Hibernate



- Arquivo de configuração do Hibernate:
  - Clique com o botão direito em cima da aplicação, vá em Novo>Outros e selecione a opção abaixo.



# Hibernate



- Arquivo de configuração do Hibernate:
  - Na próxima tela verifique se está com os mesmos dados da imagem abaixo.

The screenshot shows the 'New XML Document' dialog box in NetBeans IDE. The 'Steps' panel on the left indicates the current step is '2. Name and Location'. The 'Name and Location' panel on the right contains the following fields:

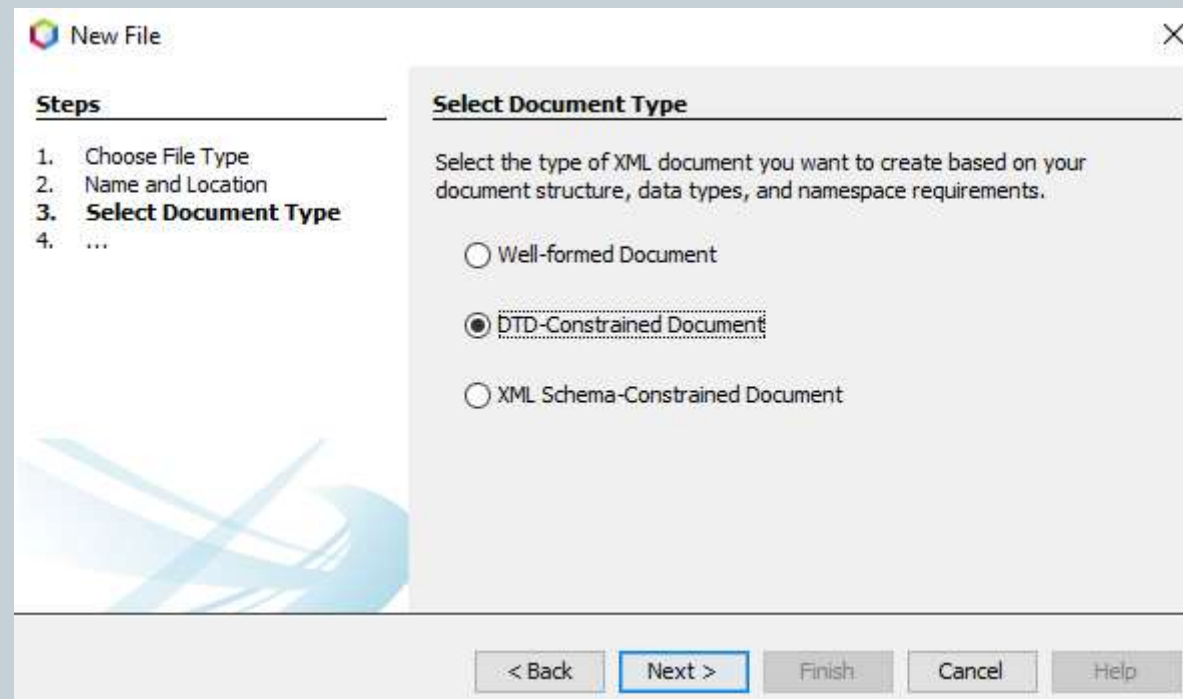
- File Name:** hibernate.cfg
- Project:** Aplicações Hibernate
- Folder:** src (with a 'Browse...' button next to it)
- Created File:** ::\Users\Pablo\Documents\NetBeansProjects\Aplicações Hibernate\src\hibernate.cfg.xml

At the bottom of the dialog, there are five buttons: '< Back', 'Next >' (which is highlighted with a blue border), 'Finish', 'Cancel', and 'Help'.

# Hibernate



- Arquivo de configuração do Hibernate:
  - Na próxima tela verifique se está com os mesmos dados da imagem abaixo e clique em Next.



# Hibernate



- Arquivo de configuração do Hibernate:
  - Na próxima tela verifique se está com os mesmos dados da imagem abaixo e clique em Finish.

The screenshot shows a 'New File' dialog box with a 'Steps' panel on the left and a 'DTD Options' panel on the right. The 'Steps' panel lists four steps: 1. Choose File Type, 2. Name and Location, 3. Select Document Type, and 4. DTD Options (which is currently selected). The 'DTD Options' panel contains the text 'Specify the following DTD options for your document:' followed by three dropdown menus labeled 'DTD Public ID:', 'DTD System ID:', and 'Document Root:'. At the bottom of the dialog are five buttons: '< Back', 'Next >', 'Finish' (highlighted with a blue border), 'Cancel', and 'Help'.

New File

**Steps**

1. Choose File Type
2. Name and Location
3. Select Document Type
4. **DTD Options**

**DTD Options**

Specify the following DTD options for your document:

DTD Public ID:

DTD System ID:

Document Root:

< Back Next > Finish Cancel Help

# Hibernate



- Arquivo de configuração do Hibernate:
  - Apague o código gerado e adicione o abaixo.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- Configuração da conexão com o banco MySQL e dialeto -->
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost/agenda</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <!-- Usando as configurações do C3PO para pool de conexões -->
    <property name="c3po.min_size">5</property>
    <property name="c3po.max_size">20</property>
    <property name="c3po.timeout">300</property>
    <property name="c3po.max_statements">50</property>
    <property name="c3po.idle_test_period">3000</property>
    <!-- Configurações de debug -->
    <!--
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="generate_statistics">true</property>
    <property name="use_sql_comments">true</property>
    -->
  </session-factory>
</hibernate-configuration>
```

# Hibernate



- Crie a classe HibernateUtil de acordo com a figura.

The image shows a screenshot of the 'New Classe Java' (New Java Class) dialog box in the NetBeans IDE. The dialog is titled 'New Classe Java' and has a close button (X) in the top right corner. It is divided into two main sections: 'Etapas' (Steps) on the left and 'Nome e Localização' (Name and Location) on the right. The 'Etapas' section shows a list of steps: '1. Escolher Tipo de Arquivo' and '2. Nome e Localização', with the second step being the active one. The 'Nome e Localização' section contains several fields: 'Nome da Classe:' with the text 'HibernateUtil'; 'Projeto:' with the text 'Aplicações Hibernate'; 'Localização:' with a dropdown menu showing 'Pacotes de Códigos-fonte'; 'Pacote:' with a dropdown menu showing 'exemplo2.conexao'; and 'Arquivo Criado:' with the text 'nts\NetBeansProjects\Aplicações Hibernate\src\exemplo2\conexao\HibernateUtil.java'. At the bottom of the dialog, there are five buttons: '< Voltar', 'Próximo >', 'Finalizar', 'Cancelar', and 'Ajuda'. The 'Finalizar' button is highlighted in blue.



# Hibernate



- Insira o código da classe HibernateUtil.

```
private static final SessionFactory sessionFactory = buildSessionFactory();

private static SessionFactory buildSessionFactory() {
    try {
        AnnotationConfiguration cfg = new AnnotationConfiguration();
        cfg.configure("hibernate.cfg.xml");
        return cfg.buildSessionFactory();
    } catch (Throwable e) {
        System.out.println("Criação inicial do objeto SessionFactory falhou. Erro: " + e);
        throw new ExceptionInInitializerError(e);
    }
}

public static SessionFactory getSessionFactory() {
    return sessionFactory;
}
```

# Hibernate



- Crie a classe “ConectaHibernateMySQL” para testar a conexão.

**New Classe Java**

**Etapas**

1. Escolher Tipo de Arquivo
2. **Nome e Localização**

**Nome e Localização**

Nome da Classe:

Projeto:

Localização:

Pacote:

Arquivo Criado:

< Voltar   Próximo >   Finalizar   Cancelar   Ajuda

# Hibernate



- Insira o código da classe “ConectaHibernateMySQL”.

```
public class ConectaHibernateMySQL {  
    public static void main(String[] args) {  
        Session sessao = null;  
        try {  
            sessao = HibernateUtil.getSessionFactory().openSession();  
            System.out.println("Conectou!");  
        } finally {  
            sessao.close();  
        }  
    }  
}
```

# Hibernate



- Execute a aplicação e veja se no terminal de saída do NetBeans a mensagem “Conectou!” apareceu.

# Hibernate



- Mapeamento de objetos com annotations
  - Crie a classe “ContatoAnnotations” no pacote “exemplo2.crudannotations”.

```
@Entity
@Table(name = "contato")
public class ContatoAnnotations {

    @Id
    @GeneratedValue
    @Column(name = "codigo")
    private Integer codigo;

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

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

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

    @Column(name = "dt_cad", nullable = true)
    private Date dataCadastro;

    @Column(name = "obs", nullable = true)
    private String observacao;

    //gerar getters e setters.
}
```

# Hibernate



- Mapeamento de objetos com annotations
  - Vá no arquivo hibernate.cfg.xml e acrescente a linha abaixo dentro das tags <session-factory> </session-factory>.

```
<mapping class="exemplo2.crudannotations.ContatoAnnotations"/>
```

# Hibernate



- **Mapeamento de objetos com annotations**
  - Crie a classe “ContatoCRUD\_Annotations.java” no pacote “exemplos2.crudannotations”.
  - Dentro da classe adicione os seguintes métodos.

# Hibernate



- Mapeamento de objetos com annotations

```
public void salvar(ContatoAnnotations contato) {  
    Session sessao = null;  
    Transaction transacao = null;  
  
    try {  
        sessao = HibernateUtil.getSessionFactory().openSession();  
        transacao = sessao.beginTransaction();  
        sessao.save(contato);  
        transacao.commit();  
    } catch (HibernateException e) {  
        System.out.println("Não foi possível inserir o contato. Erro: " + e.getMessage());  
    } finally {  
        try {  
            sessao.close();  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operação de inserção. Mensagem: " + e.getMessage());  
        }  
    }  
}
```



# Hibernate



- Mapeamento de objetos com annotations

```
public void atualizar(ContatoAnnotations contato) {  
    Session sessao = null;  
    Transaction transacao = null;  
  
    try {  
        sessao = HibernateUtil.getSessionFactory().openSession();  
        transacao = sessao.beginTransaction();  
        sessao.update(contato);  
        transacao.commit();  
    } catch (HibernateException e) {  
        System.out.println("Não foi possível alterar o contato. Erro: " + e.getMessage());  
    } finally {  
        try {  
            sessao.close();  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operação de atualização. Mensagem: " + e.getMessage());  
        }  
    }  
}
```

# Hibernate



- Mapeamento de objetos com annotations

```
public void excluir(ContatoAnnotations contato) {  
    Session sessao = null;  
    Transaction transacao = null;  
  
    try {  
        sessao = HibernateUtil.getSessionFactory().openSession();  
        transacao = sessao.beginTransaction();  
        sessao.delete(contato);  
        transacao.commit();  
    } catch (HibernateException e) {  
        System.out.println("Não foi possível excluir o contato. Erro: " + e.getMessage());  
    } finally {  
        try {  
            sessao.close();  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operação de exclusão. Mensagem: " + e.getMessage());  
        }  
    }  
}
```

# Hibernate



- Mapeamento de objetos com annotations

```
public List<ContatoAnnotations> listar() {  
    Session sessao = null;  
    Transaction transacao = null;  
    Query consulta = null;  
    List<ContatoAnnotations> resultado = null;  
  
    try {  
        sessao = HibernateUtil.getSessionFactory().openSession();  
        transacao = sessao.beginTransaction();  
        consulta = sessao.createQuery("SELECT * FROM CONTATO");  
        resultado = consulta.list();  
        transacao.commit();  
        return resultado;  
    } catch (HibernateException e) {  
        System.out.println("Não foi possível selecionar contatos. Erro: " + e.getMessage());  
        throw new HibernateException(e);  
    } finally {  
        try {  
            sessao.close();  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operação de consulta. Mensagem: " + e.getMessage());  
        }  
    }  
}
```

# Hibernate



- Mapeamento de objetos com annotations

```
public ContatoAnnotations buscaContato(int valor) {
    ContatoAnnotations contato = null;
    Session sessao = null;
    Transaction transacao = null;
    Query consulta = null;

    try {
        sessao = HibernateUtil.getSessionFactory().openSession();
        transacao = sessao.beginTransaction();
        consulta = sessao.createQuery("from Contato where codigo = :parametro");
        consulta.setInteger("parametro", valor);
        contato = (ContatoAnnotations) consulta.uniqueResult();
        transacao.commit();
        return contato;
    } catch (HibernateException e) {
        System.out.println("Não foi possível buscar contato. Erro: " + e.getMessage());
    } finally {
        try {
            sessao.close();
        } catch (Throwable e) {
            System.out.println("Erro ao fechar operação de buscar. Mensagem: " + e.getMessage());
        }
    }
    return contato;
}
```

# Hibernate



- Mapeamento de objetos com annotations

```
public static void main(String[] args) {
    ContatoCRUD_Annotations contatoCrudXML = new ContatoCRUD_Annotations();
    String[] nomes = {"Fulano", "Beltrano", "Ciclano"};
    String[] fones = {"(47) 2222-1111", "(47) 7777-5555", "(47) 9090-2525"};
    String[] emails = {"fulano@teste.com.br", "beltrano@teste.com.br", "ciclano@teste.com.br"};
    String[] observacoes = {"Novo cliente", "Cliente em dia", "Ligar na quinta"};
    ContatoAnnotations contato = null;

    for (int i = 0; i < nomes.length; i++) {
        contato = new ContatoAnnotations();
        contato.setNome(nomes[i]);
        contato.setTelefone(fones[i]);
        contato.setEmail(emails[i]);
        contato.setDataCadastro(new Date(System.currentTimeMillis()));
        contato.setObservacao(observacoes[i]);
        contatoCrudXML.salvar(contato);
    }
    System.out.println("Total de registros cadastrados: " + contatoCrudXML.listar().size());
}
```

# Hibernate



- **Mapeamento de objetos com annotations**
  - Execute essa Classe e verifique no terminal se tudo ocorreu correto.
  - Abra o MySQL e verifique se foi salvo os 3 contatos.

# Hibernate



- O arquivo `hibernate.cfg.xml`:
  - Arquivo de configuração do Hibernate com o BD.
  - A tag `<mapping resource/>` serve para que o hibernate reconheça o mapeamento.
- `HibernateUtil.java`:
  - Serve para fazer a “ligação” entre o arquivo de configuração e a conexão com o BD.

# Hibernate



- Os métodos “equals()” e “hashCode()”:
  - ✦ Servem para agilizar as buscas em collections.
  - ✦ O hibernate recomenda o uso desses métodos em classes persistentes.
  - ✦ “hashCode” serve para dividir os objetos salvos por algum critério.
  - ✦ “equals” serve para comparar o objeto em questão.
  - ✦ O NetBeans cria automaticamente esses métodos.
- A interface Serializable:
  - ✦ Salva, grava, captura o estado de um objeto e, em um momento oportuno, recuperá-lo.
  - ✦ Transforma o objeto em uma stream de bytes.

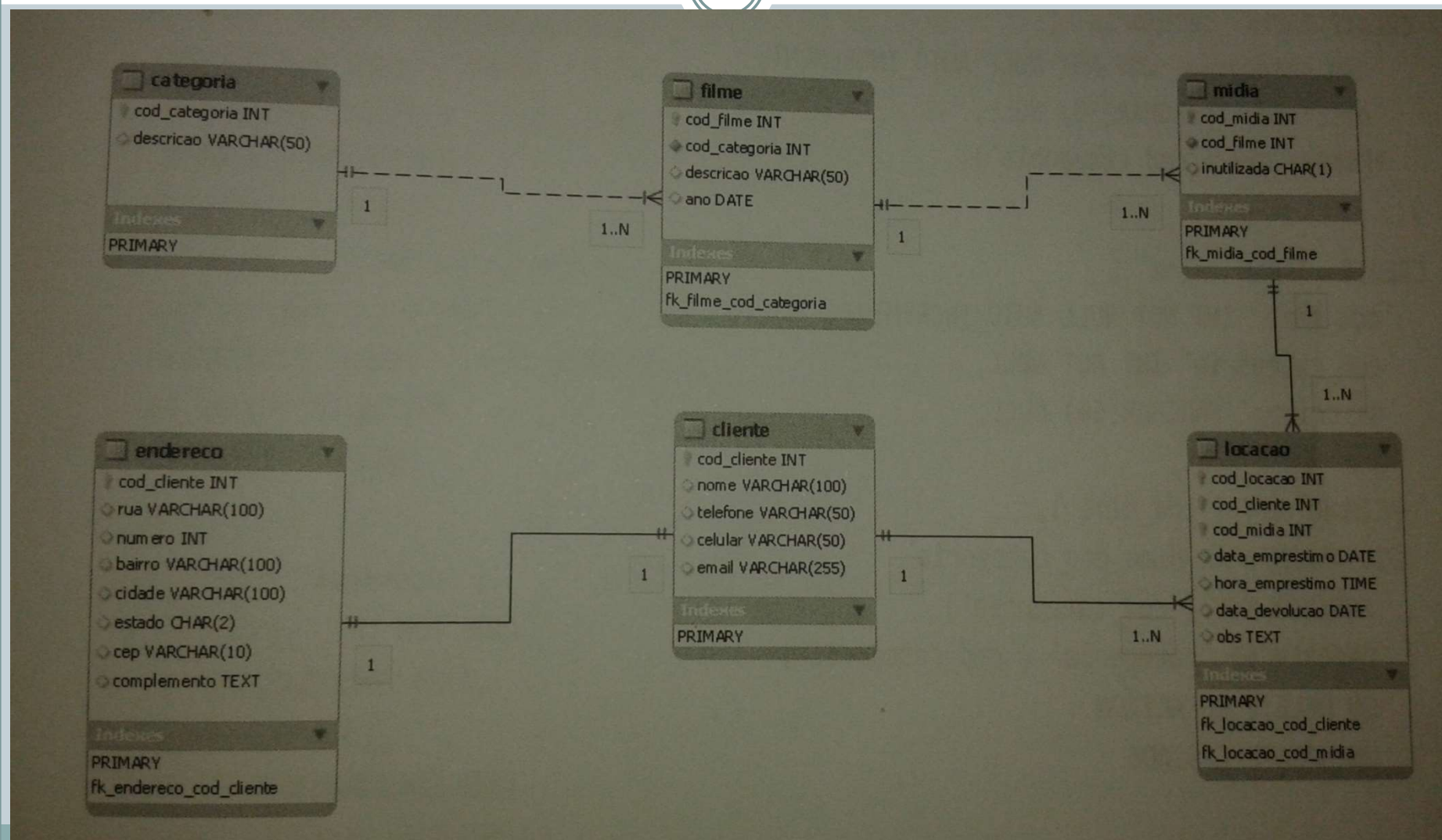


# Hibernate



- Crie um novo projeto do tipo “Aplicação Java” com o nome de “Locadora”.
- Adicione as .jar do exemplo anterior com Hibernate.
- Crie a Classe “HibernateUtil” no pacote “exemplo3.util” com o mesmo código do exemplo anterior.
- Crie utilizando o MySQL Workbench o banco de dados “locadora”.

# Hibernate



# Hibernate



- Crie a classe “Categoria” no pacote “exemplo3.categoria” com o seguinte código.

```
@Entity
@Table(name = "categoria")
public class Categoria implements Serializable{

    @Id
    @GeneratedValue
    @Column(name = "cod_categoria")
    private Integer categoria;
    private String descricao;

    //Gerar gets e sets
    //Gerar equals() e hashCode()

}
```

# Hibernate



- Crie a classe “CategoriaDAO” no pacote “exemplo3.categoria” com o seguinte código.

```
public class CategoriaDAO {  
    private Session sessao;  
    private Transaction transacao;
```

# Hibernate



- Na classe “CategoriaDAO” crie o método salvar.

```
public void salvar(Categoria categoria) {  
    try {  
        this.sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
        this.transacao = this.sessao.beginTransaction();  
        this.sessao.save(categoria);  
        this.transacao.commit();  
    } catch (HibernateException e) {  
        System.out.println("Não foi possível inserir a categoria. "  
            + "Erro: " + e.getMessage());  
    } finally {  
        try {  
            if (this.sessao.isOpen()) {  
                this.sessao.close();  
            }  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operação de inserção. "  
                + "Mensagem: " + e.getMessage());  
        }  
    }  
}
```

# Hibernate



- Na classe “CategoriaDAO” crie o método atualizar.

```
public void atualizar(Categoria categoria) {  
    try {  
        this.sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
        this.transacao = this.sessao.beginTransaction();  
        this.sessao.update(categoria);  
        this.transacao.commit();  
    } catch (HibernateException e) {  
        System.out.println("Não foi possível alterar a categoria. Erro: " + e.getMessage());  
    } finally {  
        try {  
            if (this.sessao.isOpen()) {  
                this.sessao.close();  
            }  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operação de atualização. Mensagem: " + e.getMessage());  
        }  
    }  
}
```

# Hibernate



- Na classe “CategoriaDAO” crie o método excluir.

```
public void excluir(Categoria categoria) {  
    try {  
        this.sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
        this.transacao = this.sessao.beginTransaction();  
        this.sessao.delete(categoria);  
        this.transacao.commit();  
    } catch (HibernateException e) {  
        System.out.println("Não foi possível excluir a categoria. Erro: " + e.getMessage());  
    } finally {  
        try {  
            if (this.sessao.isOpen()) {  
                this.sessao.close();  
            }  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operação de exclusão. Mensagem: " + e.getMessage());  
        }  
    }  
}
```



# Hibernate



- Na classe “CategoriaDAO” crie a função buscaCategoria.

```
public Categoria buscaCategoria(Integer codigo) {
    Categoria categoria = null;

    try {
        this.sessao = HibernateUtil.getSessionFactory().getCurrentSession();
        this.transacao = this.sessao.beginTransaction();
        Criteria filtro = this.sessao.createCriteria(Categoria.class);
        filtro.add(Restrictions.eq("categoria", codigo));
        categoria = (Categoria) filtro.uniqueResult();
        this.transacao.commit();
    } catch (Throwable e) {
        if (this.transacao.isActive()) {
            this.transacao.rollback();
        }
    } finally {
        try {
            if (this.sessao.isOpen()) {
                this.sessao.close();
            }
        } catch (Throwable e) {
            System.out.println("Erro ao fechar operação de busca. Mensagem: " + e.getMessage());
        }
    }

    return categoria;
}
```



# Hibernate



- Na classe “CategoriaDAO” crie a função listar.

```
public List<Categoria> listar() {  
    List<Categoria> categorias = null;  
  
    try {  
        this.sessao = HibernateUtil.getSessionFactory().getCurrentSession();  
        this.transacao = this.sessao.beginTransaction();  
        Criteria filtro = this.sessao.createCriteria(Categoria.class);  
        categorias = filtro.list();  
        this.transacao.commit();  
    } catch (Throwable e) {  
        if (this.transacao.isActive()) {  
            this.transacao.rollback();  
        }  
    } finally {  
        try {  
            if (this.sessao.isOpen()) {  
                this.sessao.close();  
            }  
        } catch (Throwable e) {  
            System.out.println("Erro ao fechar operação de listagem. Mensagem: " + e.getMessage());  
        }  
    }  
    return categorias;  
}
```

# Hibernate



- Crie a classe “Filme” no pacote “exemplo3.filme” com seguinte código.

```
@Entity
@Table(name = "filme")
public class Filme implements Serializable {

    @Id
    @GeneratedValue
    @Column(name = "cod_filme")
    private Integer filme;

    @ManyToOne
    @JoinColumn(name = "cod_categoria")
    private Categoria categoria;

    private String descricao;

    private Date ano;

    //Gerar gets e sets
    //Gerar equals() e hashCode()
}
```

# Hibernate



- Crie a classe “FilmeDAO” no pacote “exemplo3.filme” com o código parecido com de “CategoriaDAO”.
  - Obs: Mude os parâmetros para o da Classe Filme.

# Hibernate



- Crie a classe “Midia” no pacote “exemplo3.midia” com o seguinte código.

```
@Entity
@Table(name = "midia")
public class Midia implements Serializable {

    @Id
    @GeneratedValue
    @Column(name = "cod_midia")
    private Integer midia;

    @ManyToOne
    @JoinColumn(name = "cod_filme")
    private Filme filme;

    private String inutilizada;

    //Gerar gets e sets
    //Gerar equals() e hashCode()
}
```

# Hibernate



- Crie a classe “MidiaDAO” no pacote “exemplo3.midia” com o código parecido com de “CategoriaDAO”.
  - Obs: Mude os parâmetros para o da Classe Midia.

# Hibernate



- Crie a classe “Locacao” no pacote “exemplo3.locacao” com o seguinte código.

```
@Entity
@Table(name = "locacao")
public class Locacao implements Serializable {

    @Id
    @GeneratedValue
    @Column(name = "cod_locacao")
    private Integer locacao;

    @ManyToOne
    @JoinColumn(name = "cod_cliente")
    private Cliente cliente;

    @ManyToOne
    @JoinColumn(name = "cod_midia")
    private Midia midia;

    @Column(name = "data_emprestimo", updatable = false)
    //A tag updatable informa se o valor pode ser alterado
    private Date dataEmprestimo;
```

# Hibernate



```
@Column(name = "hora_emprestimo", updatable = false)
private Time horaEmprestimo;

@Column(name = "data_devolucao")
private Date dataDevolucao;

@Column(name = "obs")
private String observacao;
//Gerar gets e sets
//Gerar equals() e hashCode()
}
```

# Hibernate



- Crie a classe “LocacaoDAO” no pacote “exemplo3.locacao” com o código parecido com de “CategoriaDAO”.
  - Obs: Mude os parâmetros para o da Classe Locacao.



# Hibernate



- Crie a classe “Cliente” no pacote “exemplo3.cliente” com o seguinte código.

```
@Entity
@Table(name = "cliente")
public class Cliente implements Serializable {

    @Id
    @GeneratedValue
    @Column(name = "cod_cliente")
    private Integer cliente;

    @OneToOne
    @PrimaryKeyJoinColumn(name = "cod_cliente")
    private Endereco endereco;

    @OneToMany(mappedBy="cliente")
    private List<Locacao> locacoes;
```

# Hibernate



```
private String nome;  
  
private String telefone;  
  
private String celular;  
  
private String email;  
  
//Gerar gets e sets  
//Gerar equals() e hashCode()  
}
```

# Hibernate



- Crie a classe “ClienteDAO” no pacote “exemplo3.cliente” com o código parecido com de “CategoriaDAO”.
  - Obs: Mude os parâmetros para o da Classe Cliente.

# Hibernate



- Crie a classe “Endereco” no pacote “exemplo3.endereco” com o seguinte código.

```
@Entity
@Table(name = "endereco")
public class Endereco implements Serializable {

    @Id
    @GeneratedValue(generator = "fk_endereco_cod_cliente")
    @org.hibernate.annotations.GenericGenerator(name = "fk_endereco_cod_cliente",
        strategy = "foreign", parameters = @Parameter(name = "property", value = "cliente"))
    @Column(name = "cod_cliente")
    private Integer endereco;

    @OneToOne(mappedBy="endereco")
    private Cliente cliente;

    private String rua;

    private Integer numero;

    private String bairro;

    private String cidade;
```

# Hibernate



```
@Column(name = "estado")
private String uf;

private String cep;

private String complemento;

//Gerar gets e sets
//Gerar equals() e hashCode()
}
```

# Hibernate



- Crie a classe “EnderecoDAO” no pacote “exemplo3.endereco” com o código parecido com de “CategoriaDAO”.
  - Obs: Mude os parâmetros para o da Classe Endereco.

# Hibernate



- Crie o arquivo de configuração hibernate com o seguinte código.

```
<hibernate-configuration>
  <session-factory>
    <!-- Configuração da conexão com o banco MySQL e dialeto -->
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost/locadora</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <property name="current_session_context_class">thread</property>
    <property name="hibernate.hbm2ddl.auto">create</property>

    <!-- Usando as configurações do C3PO para pool de conexões -->
    <property name="c3po.min_size">5</property>
    <property name="c3po.max_size">20</property>
    <property name="c3po.timeout">300</property>
    <property name="c3po.max_statements">50</property>
    <property name="c3po.idle_test_period">3000</property>
```

# Hibernate



```
<!-- Configurações de debug -->
<property name="show_sql">true</property>
<property name="format_sql">true</property>
<property name="generate_statistics">true</property>
<property name="use_sql_comments">true</property>

<mapping class="exemplo3.categoria.Categoria"/>
<mapping class="exemplo3.filme.Filme"/>
<mapping class="exemplo3.midia.Midia"/>
<mapping class="exemplo3.cliente.Cliente"/>
<mapping class="exemplo3.endereco.Endereco"/>
<mapping class="exemplo3.locacao.Locacao"/>

</session-factory>
</hibernate-configuration>
```



# Hibernate



- Crie a classe “Locadora” no pacote “exemplo3.locadora” com o seguinte código.

```
public class Locadora {  
    public static void main(String[] args) {  
        HibernateUtil.getSessionFactory().openSession();  
        Locadora locadora = new Locadora();  
        locadora.cadastraCategorias();  
        locadora.cadastraFilmes();  
        locadora.cadastraMidias();  
  
        EnderecoDAO enderecoDAO = new EnderecoDAO();  
        Endereco endereco = new Endereco();  
        Cliente cliente = new Cliente();  
        ClienteDAO clienteDAO = new ClienteDAO();  
        cliente.setCelular("(47) 1111-2222");  
        cliente.setEmail("solaris@javapro.com.br");  
        cliente.setNome("Fulano Solaris");  
        cliente.setTelefone("(47) 3333-7777");  
        cliente.setEndereco(endereco);  
        endereco.setBairro("Centro");  
        endereco.setCep("89000-000");  
        endereco.setCidade("Joinville");  
        endereco.setComplemento("casa");  
        endereco.setNumero(new Integer(1));  
        endereco.setRua("Av. Principal");  
        endereco.setUf("SC");  
        endereco.setCliente(cliente);  
        clienteDAO.salvar(cliente);  
        enderecoDAO.salvar(endereco);  
    }  
}
```

# Hibernate



```
LocacaoDAO locacaoDAO = new LocacaoDAO();
Locacao locacao = new Locacao();
locacao.setDataDevolucao(new Date(System.currentTimeMillis()));
locacao.setDataEmprestimo(new Date(System.currentTimeMillis()));
locacao.setObservacao("Devolução final de semana");
locacao.setHoraEmprestimo(new Time(System.currentTimeMillis()));

locacao.setCliente(cliente);

MidiaDAO midiaDAO = new MidiaDAO();
Midia midia = (Midia) midiaDAO.buscaMidia(new Integer(1));
locacao.setMidia(midia);

locacaoDAO.salvar(locacao);
System.out.println("Cadastros gerados com sucesso!");
}
```

# Hibernate



- Crie o método cadastraCategoria() dentro da classe “Locadora”.

```
public void cadastraCategorias() {  
    //Criando as categorias dos filmes  
    String categorias[] = {"Aventura", "Ação", "Comédia"};  
    Categoria categoria = null;  
    CategoriaDAO categoriaDAO = new CategoriaDAO();  
  
    for (int i = 0; i < 3; i++) {  
        categoria = new Categoria();  
        categoria.setDescricao(categorias[i]);  
        categoriaDAO.salvar(categoria);  
    }  
}
```

# Hibernate



- Crie o método cadastraFilmes() dentro da classe “Locadora”.

```
public void cadastraFilmes() {
    CategoriaDAO categoriaDAO = new CategoriaDAO();
    String[] filmesDescricao = {"Senhor dos Anéis", "Transformers", "Ghostbusters"};
    //Aqui subtraímos o ano de lançamento do filme de 1900, para gravarmos o ano correto no banco.
    Date[] filmesAnoProducao = {new Date(2001-1900, 11, 19), new Date(2007-1900, 6, 20), new Date(1985-1900, 1, 1)};
    FilmeDAO filmeDAO = new FilmeDAO();
    Filme filme = null;

    for (int i = 0; i < 3; i++) {
        filme = new Filme();
        filme.setDescricao(filmesDescricao[i]);
        filme.setAno(filmesAnoProducao[i]);
        filme.setCategoria(categoriaDAO.buscaCategoria(i + 1));
        filmeDAO.salvar(filme);
    }
}
```

# Hibernate



- Crie o método cadastraMidias() dentro da classe “Locadora”.

```
public void cadastraMidias() {  
    Midia midia = null;  
    Filme filme = null;  
    MidiaDAO midiaDAO = new MidiaDAO();  
    FilmeDAO filmeDAO = new FilmeDAO();  
    List<Filme> resultado = filmeDAO.listar();  
  
    for (int i = 0; i < 3; i++) {  
        midia = new Midia();  
        filme = (Filme) resultado.get(i);  
        midia.setFilme(filme);  
        midia.setInutilizada("N");  
        midiaDAO.salvar(midia);  
    }  
}
```

# Hibernate



- Execute o projeto e verifique se ocorreu tudo certo na saída do terminal.
- Vá ao MySQL Workbench e verifique se as tabelas foram criadas corretamente.

# Hibernate



- `@Entity`: especifica a classe que está sendo mapeada.
- `@Table`: especifica a tabela no banco de dados.
- `@Id`: indica qual é a chave primaria da classe.
- `@GeneratedValue`: valor gerado automaticamente pelo banco.
- `@Column`: definir as configurações da coluna no BD.
- `@OneToOne`: configura um relacionamento um-para-um.
- `@OneToMany`: configura um relacionamento um-para-muitos.
  - O “`mappedBy`” indica um mapeamento bidirecional.

# Hibernate



- @ManyToOne: configura um relacionamento muitos-para-um.
- @ManyToMany: configura um relacionamento muitos-para-muitos.
- @JoinColumn: indica explicitamente o nome da coluna que guarda a chave estrangeira.
- @PrimaryKeyJoinColumn: usado em relacionamentos um-para-um quando ambas as tabelas compartilham a mesma chave primária.
- @GenericGenerator: indica como a chave será gerada.
  - No exemplo da locadora, quando um cliente for salvo, o código de sua chave primaria da classe Cliente também será utilizado pela classe Endereco.
- <property name="hibernate.hbm2ddl.auto">create</property>: cria as tabelas no BD de acordo com os annotations.



# Hibernate



- Crie um novo projeto do tipo aplicação Java chamado “Comercio”.
- Adicione as jars do exemplo 3.
- Copie a classe “HibernateUtil” do exemplo 3 no pacote “exemplo4.util”.
- Crie o banco de dados “comercio” no MySQL Workbench.

# Hibernate



- Crie o arquivo de configuração do hibernate com o seguinte código.

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- Configuração da conexão com o banco MySQL e dialeto -->
    <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost/comercio</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <property name="current_session_context_class">thread</property>
    <property name="hibernate.hbm2ddl.auto">update</property>

    <!-- Usando as configurações do C3PO para pool de conexões -->
    <property name="c3po.min_size">5</property>
    <property name="c3po.max_size">20</property>
    <property name="c3po.timeout">300</property>
    <property name="c3po.max_statements">50</property>
    <property name="c3po.idle_test_period">3000</property>

    <!-- Mapeando classes -->
    <mapping class="exemplo4.categoria.Categoria"/>
    <mapping class="exemplo4.produto.Produto"/>
  </session-factory>
</hibernate-configuration>
```

# Hibernate



- Crie a classe “Produto” dentro do pacote “exemplo4.produto” com o seguinte código.

```
@Entity
@Table(name = "produto")
public class Produto implements Serializable {

    @Id
    @GeneratedValue
    @Column(name = "cod_produto")
    private Integer produto;

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

    @Column(name = "preco")
    private Double preco;

    //gerar gets e sets
    //gerar equals e hashCode

}
```

# Hibernate



- Crie a classe “ProdutoDAO” dentro do pacote “exemplo4.produto” nos mesmos moldes do já mostrado no exemplo 3.

# Hibernate



- Crie a classe “Categoria” dentro do pacote “exemplo4.categoria” com o seguinte código.

```
@Entity
@Table(name = "categoria")
public class Categoria implements Serializable {

    @Id
    @GeneratedValue
    @Column(name = "cod_categoria")
    protected Integer categoria;

    @Column(name = "descricao")
    protected String descricao;

    @ManyToMany
    @JoinTable(name = "categoria_produto",
        joinColumns = {@JoinColumn(name = "cod_categoria")},
        inverseJoinColumns = {@JoinColumn(name = "cod_produto")})
    private Set<Produto> produtos = new HashSet<Produto>();

    //gerar gets e sets
    //gerar equals e hashCode
}
```

# Hibernate



- Crie a classe “CategoriaDAO” dentro do pacote “exemplo4.categoria” nos mesmos moldes do já mostrado no exemplo 3.

# Hibernate



- Crie a classe “Cadastro” no pacote “exemplo4.cadastro” e adicione o método main.

```
public static void main(String[] args) {  
    Cadastro cadastro = new Cadastro();  
  
    cadastro.cadastraProdutos();  
    cadastro.cadastraCategorias();  
    System.out.println("Cadastros gerados com sucesso!");  
}
```

# Hibernate



- Dentro da classe Cadastro adicione o método “cadastraProdutos()”.

```
public void cadastraProdutos() {  
    String descricao[] = {"Bicicleta", "Televisão", "DVD"};  
    Double preco[] = {356.83, 19.99, 195.60};  
    ProdutoDAO produtoDAO = new ProdutoDAO();  
    Produto produto = null;  
  
    for (int i = 0; i < 3; i++) {  
        produto = new Produto();  
        produto.setDescricao(descricao[i]);  
        produto.setPreco(preco[i]);  
        produtoDAO.salvar(produto);  
    }  
}
```



# Hibernate



- Dentro da classe Cadastro adicione o método “cadastraCategorias()”.

```
public void cadastraCategorias() {  
    String descricao[] = {"Utilidades", "Geral"};  
    CategoriaDAO categoriaDAO = new CategoriaDAO();  
    Categoria categoria = null;  
    ProdutoDAO produtoDAO = new ProdutoDAO();  
    Set<Produto> produtos = new HashSet<Produto>();  
    List<Produto> produtosListagem = produtoDAO.listar();  
  
    for (int i = 0; i < produtosListagem.size(); i++) {  
        produtos.add(produtosListagem.get(i));  
    }  
  
    for (int i = 0; i < 2; i++) {  
        categoria = new Categoria();  
        categoria.setDescricao(descricao[i]);  
        categoria.setProdutos(produtos);  
        categoriaDAO.salvar(categoria);  
    }  
}
```

# Hibernate



- Execute a aplicação e veja se ocorreu tudo ok.
- O Projeto Comercio mostrou o mapeamento muitos-para-muitos unidirecional.
- Nesse exemplo é cadastrado todos os produtos em todas as categorias.
- O hibernate pode utilizar de 2 classes para realizar consultas, são elas: Query e Criteria.
- Verifique que a tag “<property name="hibernate.hbm2ddl.auto">update</property>” no arquivo do hibernate agora mudou para update.
  - Update - faz as alterações no banco do tipo, cria novas colunas ou tabelas. Também altera as propriedades das colunas. Isso acontece quando você modifica o mapeamento, no caso os annotations. Se as tabelas ainda não existem no banco ele cria no primeiro acesso.
  - Create - é arriscado usar porque ele exclui tudo e depois cria de novo. Então até para testes, você pode perder toda a sua base usando ele. Use no máximo na primeira vez que rodar o hibernate, depois modifique a configuração.

# Referências



- Hibernate Community Documentation.  
HIBERNATE – Relational Persistence for Idiomatic Java.
- Java Persistence API Documentation.
- LUCKOW, Décio. Programação Java para a WEB.  
São Paulo: Novatec, 2010.

# Extras



- <https://www.youtube.com/watch?v=AD-gNDoQkeo>
- <https://www.youtube.com/watch?v=fmTbTgW19Do&list=PL8R29l8KkRsJQDEjRm6laLT2oa8YtalUx>
- [https://www.youtube.com/watch?v=mZ4UJD4JyHs&list=PL8R29l8KkRsJfQq\\_mfi\\_289-PQjVx4Jln](https://www.youtube.com/watch?v=mZ4UJD4JyHs&list=PL8R29l8KkRsJfQq_mfi_289-PQjVx4Jln)
- <https://www.youtube.com/c/CanalGeekDev/playlists>
- <https://www.youtube.com/c/AlgaWorksCursosOnline/playlists>
- <https://www.youtube.com/c/marknit1/playlists>

# Cursos



- <https://www.jdevtreinamento.com.br/formacao-java-web-profissional/index.html>

# Cursos



- <https://www.youtube.com/user/mjailton/playlists>
- <https://www.youtube.com/c/CursoemV%C3%ADdeo/playlists>
- <https://www.youtube.com/c/Javaavancado/playlists>
- <https://www.youtube.com/c/DevDojoBrasil/playlists>
- <https://www.youtube.com/c/SergioCordeirodaSilva/playlists>