

ANHANGUERA – CAMPUS GUAJAJARAS
THIAGO CONEGUNDES MORAIS - 3510640001
SISTEMA DE INFORMAÇÃO

PORTFÓLIO – RELATÓRIO DE AULA PRÁTICA:
PROGRAMAÇÃO WEB II
PARTE II

BELO HORIZONTE - MINAS GERAIS
MARÇO/2026

THIAGO CONEGUNDES MORAIS - 3510640001
PORTFÓLIO – RELATÓRIO DE AULA PRÁTICA

PROGRAMAÇÃO WEB II
PARTE II

Trabalho acadêmico apresentado ao Curso de Bacharelado em Sistema de Informação da Faculdade Anhanguera como requisito para obtenção de pontuação semestral.

Mediator Pedagógico: Rafael R. Fassula

BELO HORIZONTE - MINAS GERAIS
MARÇO/2026

SUMÁRIO

INTRODUÇÃO	1
OBJETIVO	1
DESENVOLVIMENTO.....	2
1.1) INSERÇÃO DOS ARQUIVOS E DEPENDÊNCIAS DO HIBERNATE	2
1.2) INSERÇÃO DAS JARS NO JAVA BUILD PATH PASSO 6	3
1.3) CONFIGURAÇÃO DO COMPILADOR PASSO 7	3
1.4) CONFIGURAÇÃO DO PROJECT FACETS PASSO	4
1.5) CONFIGURAÇÃO DO PERSISTENCE.XML PASSO 8.5.....	4
RESULTADO FINAL.....	19
CONSIDERAÇÕES FINAIS.....	21
REFERÊNCIAS.....	22

INTRODUÇÃO

O conceito de ORM (Object-Relational Mapping) é fundamental para o entendimento desse projeto. De acordo com a Dev.Media (2026), ORM é uma técnica que faz a “ponte” entre o mundo orientado a objetos que contempla as classes, objetos, herança, encapsulamento, conectando ao mundo relacional que contém tabelas, linhas, colunas, chaves primárias/estrangeiras.

Sem ORM é necessário escrever muitos scripts SQL, converter resultados de tabelas em objetos Java e controlar conexões, transações e mapeamentos, todos esses processos são executados manualmente, com ORM, isso é automatizado.

Hibernate é um framework de persistência de dados que atua como intermediário entre a aplicação Java e o banco de dados. A persistência de dados é a capacidade de guardar informações de forma permanente para que possam ser acessadas e utilizadas posteriormente e o Hibernate atua nesse processo. (DEV.MEDIA, 2026).

OBJETIVO

1) Construir uma aplicação Web utilizando as seguintes tecnologias:

- Linguagem de programação Java;
- Framework JSF(Java Server Faces);
- Framework Hibernate;
- IDE Eclipse; Servidor WildFlay;
- Banco de dados Oracle MySQL;

2) Construir um sistema Web Agenda de contatos com o objetivo de realizar cadastros enviados os dados para o banco de dados realizando manobras como inserção e busca de dados.

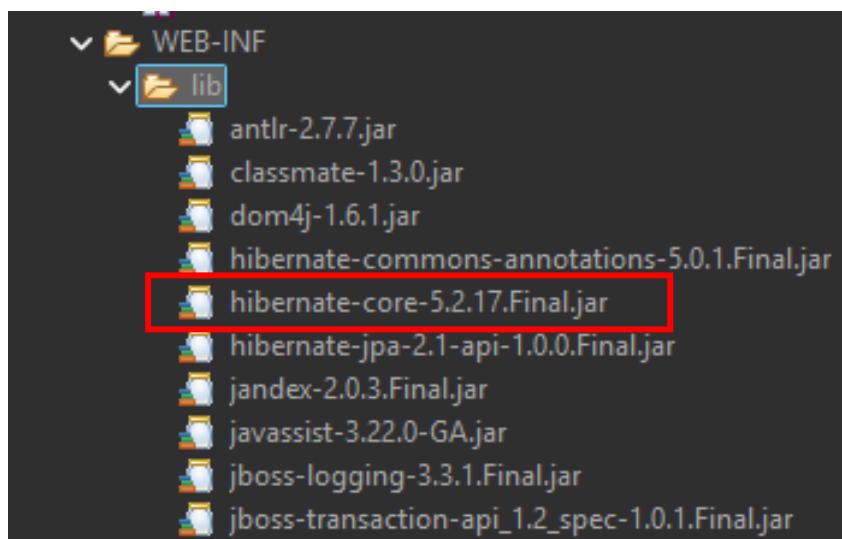
DESENVOLVIMENTO

Para iniciar o trabalho é necessário verificar se todas as versões estão convergentes, para isso temos:

- Compilador Java na versão 1.8.0_482;
- JDK na versão 1.8.0_482;
- Wild Fly na versão 15;
- Hibernate na versão 5.2.17 (Versão compatível com os tópicos acima);

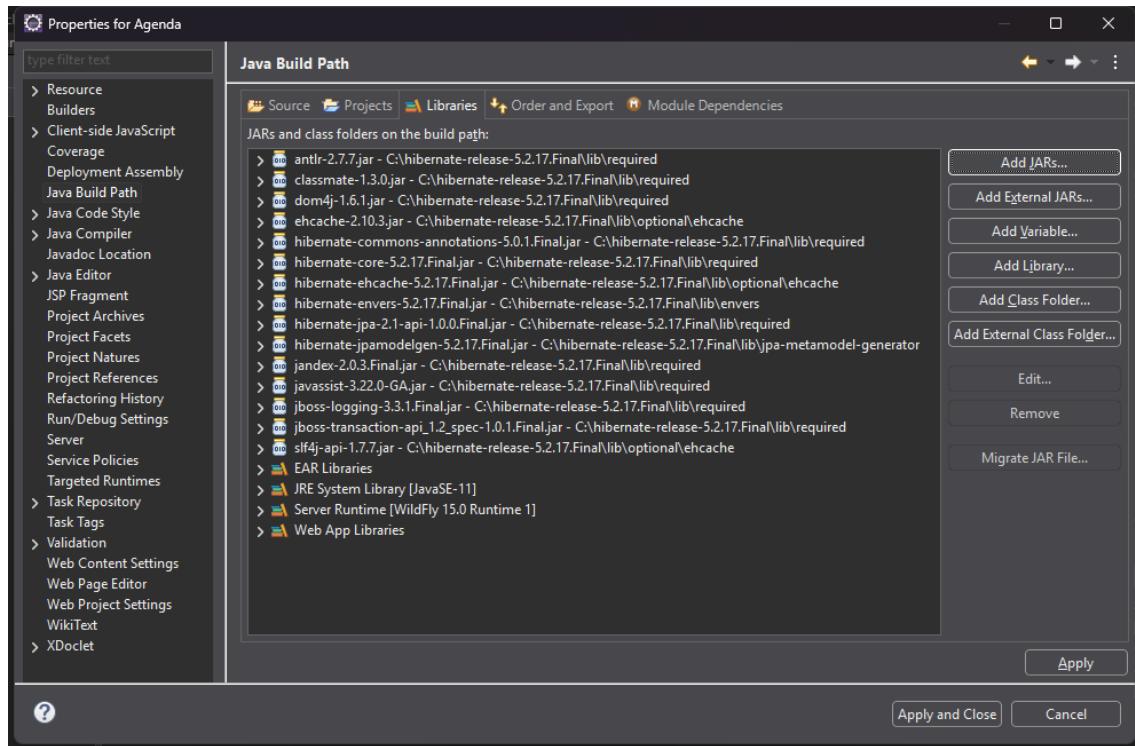
1) CONFIGURAÇÃO DO HIBERNATE

1.1) INSERÇÃO DOS ARQUIVOS E DEPENDÊNCIAS DO HIBERNATE

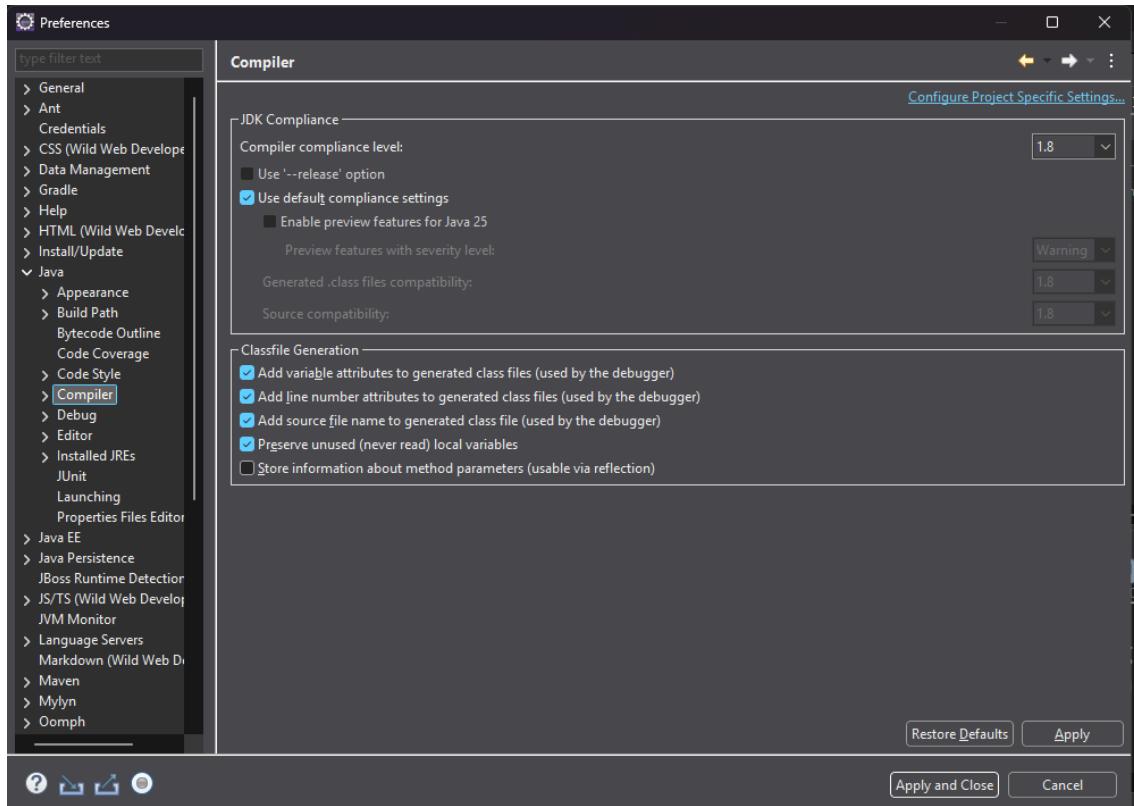


Arquivos .jar (Java Archive) do Hibernate são bibliotecas comprimidas que contêm o código pré-compilado (classes Java), e recursos necessários para integrar o framework Hibernate em um projeto Java. O arquivo hibernate-core.jar atua como núcleo funcional do Hibernate.

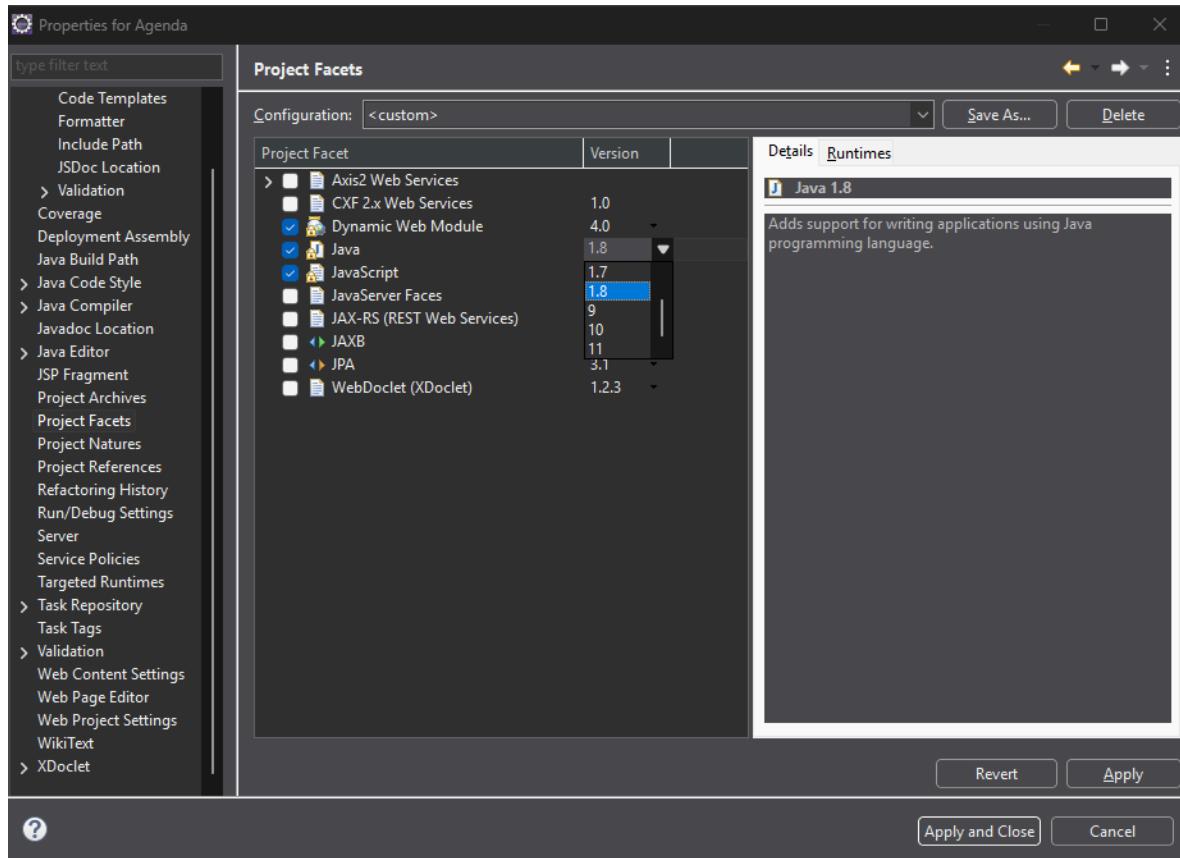
1.2) INSERÇÃO DAS JARS NO JAVA BUILD PATH PASSO 6



1.3) CONFIGURAÇÃO DO COMPILADOR PASSO 7



1.4) CONFIGURAÇÃO DO PROJECT FACETS PASSO 8D

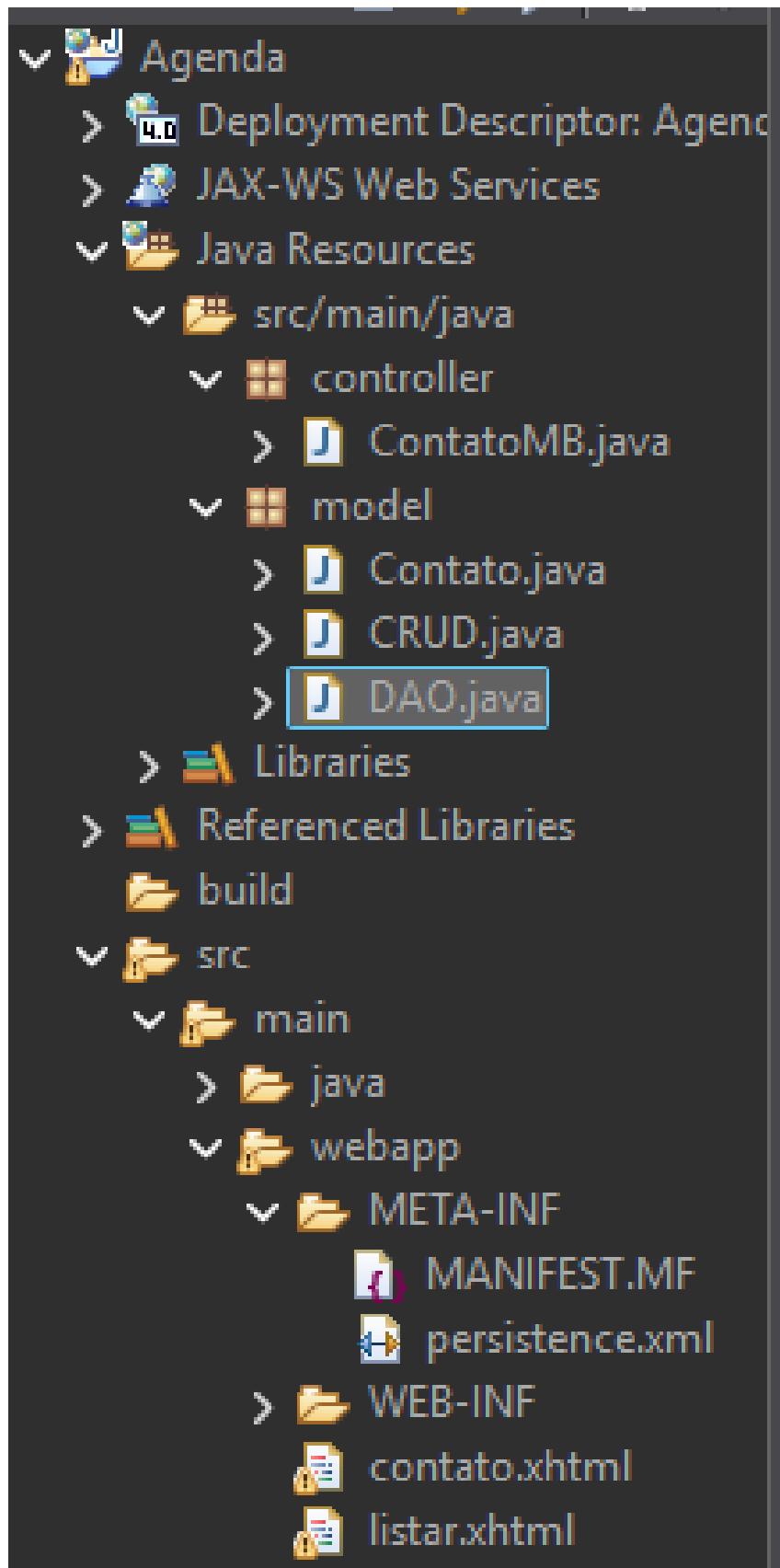


1.5) CONFIGURAÇÃO DO PERSISTENCE.XML PASSO 8.5

The screenshot shows the 'persistence.xml' file in the Eclipse IDE. The file is located in the 'META-INF' folder of the 'webapp' project. The code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
  http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
  version="2.1">
  <persistence-unit name="agendaPersist" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class>model.Contato</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost:3306/agenda?serverTimezone=UTC"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value="" />
      <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
      <property name="hibernate.hbm2ddl.auto" value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

2) ESTRUTURA DE PASTAS DO PROJETO AGENDA



3) APRESENTAÇÃO DO CÓDIGO FONTE - CONTATOMB.JAVA

```
J ContatoMB.java ×
1 package controller;
2
3+ import javax.inject.Named;...
10
11 @Named
12 @RequestScoped
13 public class ContatoMB {
14
15     private Contato c = new Contato();
16
17     private List<Contato> listaContatos;
18
19
20     public ContatoMB() {
21         }
22
23
24+     public String getId() {
25         return String.valueOf(c.getId());
26     }
27
28+     public void setId(String id) {
29         c.setId(Integer.parseInt(id));
30     }
31
32+     public String getNome() {
33         return c.getNome();
34     }
35
36+     public void setNome(String nome) {
37         c.setNome(nome);
38     }
39
40+     public String getSobrenome() {
41         return c.getSobrenome();
42     }
43
```

```

44•     public void setSobrenome(String sobrenome) {
45         c.setSobrenome(sobrenome);
46     }
47
48•     public String getTelefone() {
49         return c.getTelefone();
50     }
51
52•     public void setTelefone(String telefone) {
53         c.setTelefone(telefone);
54     }
55
56•     public String getEmail() {
57         return c.getEmail();
58     }
59

60•     public void setEmail(String email) {
61         c.setEmail(email);
62     }
63
64•     public void salvar() {
65         try {
66             System.out.println("== TENTANDO SALVAR ==");
67             System.out.println("Nome: " + c.getNome());
68             System.out.println("Sobrenome: " + c.getSobrenome());
69             System.out.println("Telefone: " + c.getTelefone());
70             System.out.println("Email: " + c.getEmail());
71
72             CRUD.inserir(c);
73
74             System.out.println("== SALVO COM SUCESSO ==");
75
76             // Limpar o formulário após salvar
77             c = new Contato();
78             // Recarregar a lista
79             listaContatos = CRUD.listarTodos();
80•         } catch(Exception ex) {
81             ex.printStackTrace();
82         }
83     }
84

```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.1

A classe “**ContatoMB.java**” está dentro do pacote controller, ela faz a conexão entre o que está no front end, “**arquivo xhtml**”, e o “esqueleto”, “**contato.java**”.

Essa classe tem um objeto chamado “C” que tem relação com a classe contato. Esse objeto aciona os métodos get e set da classe contato, como explicado na primeira parte desse trabalho.

Na linha 17 foi utilizado um Array List com o nome: “listaContatos” essa lista é do tipo “contato”, ou seja, ela armazena todos os atributos declarados na classe contato.

Na linha 64, no método salvar, na linha 64 a 70, foi implementado o System.out.println para averiguar se os dados estão de fato sendo resgatados método get.

3.1) APRESENTAÇÃO DO CÓDIGO FONTE - CONTATO.JAVA

```
1 package model;
2
3 import javax.persistence.*;
4 @Entity
5 @Table(name = "Contato")
6 |
7 public class Contato {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.IDENTITY)
11    @Column
12    private int id;
13    @Column
14    private String nome;
15    @Column
16    private String sobrenome;
17    @Column
18    private String telefone;
19    @Column
20    private String email;
21
22    public int getId() {
23        return id;
24    }
25}
```

```
26•    public void setId(int id) {  
27        this.id = id;  
28    }  
29  
30•    public String getNome() {  
31        return nome;  
32    }  
33  
34•    public void setNome(String nome) {  
35        this.nome = nome;  
36    }  
37  
38•    public String getSobrenome() {  
39        return sobrenome;  
40    }  
41
```

```
42•    public void setSobrenome(String sobrenome) {  
43        this.sobrenome = sobrenome;  
44    }  
45  
46•    public String getTelefone() {  
47        return telefone;  
48    }  
49  
50•    public void setTelefone(String telefone) {  
51        this.telefone = telefone;  
52    }  
53  
54•    public String getEmail() {  
55        return email;  
56    }  
57  
58•    public void setEmail(String email) {  
59        this.email = email;  
60    }  
61  
62 }
```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.1

A classe “**Contato.java**” funciona como arcabouço, esqueleto ela traz os atributos principais como nome telefone como variáveis privadas sendo necessário os métodos get e set para acessá-las.

A anotação @Entity, na linha 4, marca a classe como entidade JPA (Java Persistence API) (API = Application Program Interface). Indica que os objetos dessa classe serão persistidos no banco de dados.

A anotação @Table, linha 5, especifica qual tabela do banco de dados os dados serão direcionados.

O “@Id” Marca o campo id como chave primária da tabela, Significa que este campo é o identificador único de cada registro

@GeneratedValue(strategy = GenerationType.IDENTITY) Define como o valor do ID será gerado automaticamente IDENTITY = Delega ao banco de dados a responsabilidade de gerar o ID sendo equivalente ao AUTO_INCREMENT do MySQL

3.2) APRESENTAÇÃO DO CÓDIGO DO ARQUIVO CRUD.JAVA

```
1 package model;
2
3+import java.util.List;[]
4
5
6
7
8
9 public class CRUD {
10     public static void inserir(Contato c1) {
11
12         try {
13             EntityManager entityManager = DAO.getEntityManager();
14
15             entityManager.getTransaction().begin();
16             entityManager.persist(c1);
17             entityManager.getTransaction().commit();
18             entityManager.close();
19             System.out.println("conectado Salvo!");
20         } catch(Exception ex) {
21             ex.printStackTrace();
22         }
23     }
24 }
```

```

25     // NOVO MÉTODO PARA LISTAR TODOS
26●  public static List<Contato> listarTodos() {
27     List<Contato> contatos = null;
28●  try {
29     EntityManager entityManager = DAO.getEntityManager();
30     entityManager.getTransaction().begin();
31
32     // Query JPQL para buscar todos os contatos
33     contatos = entityManager.createQuery("FROM Contato", Contato.class).getResultList();
34
35     entityManager.getTransaction().commit();
36     entityManager.close();
37● } catch(Exception ex) {
38     ex.printStackTrace();
39 }
40     return contatos;
41 }
42

```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.2

O arquivo **crud.java** tem a função de enviar dados para o banco, realizando os processos de inserção - “creat”; leitura - “read”; alteração “update”; deleção –“delete”.

Na linha 10 temos o método inserir, ele recebe como parâmetro o objeto C1 do tipo Contato. O **EntityManager** é o objeto que gerencia as operações com banco, já o **DAO.getEntityManager()** é método que cria/devolve uma conexão com o banco

O método **getTransaction()** acessa o controle de transação. O método **begin()**, Inicia uma transação e o método **persist(c1)** manda salvar o objeto c1 no banco.

Na linha 26, temos o método **listarTodos()**, na linha 33 o arrayList **contatos** que chama o método “**entityManager.createQuery**” criando uma query que traz todos os contatos contidos na tabela contatos no banco de dados.

3.3) APRESENTAÇÃO DO CÓDIGO DO ARQUIVO DAO.JAVA

```
1 package model;
2
3 import javax.persistence.*;
4
5 public class DAO {
6
7     private static final EntityManagerFactory emFactory;
8
9     static {
10         emFactory = Persistence.createEntityManagerFactory("agendaPersist");
11     }
12
13     public static EntityManager getEntityManager() {
14         return emFactory.createEntityManager();
15     }
16
17     public static void fecharFactory() {
18         emFactory.close();
19     }
20
21 }
```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.3

O arquivo “DAO” significa Data Access Object, ele gerencia a fábrica de conexões com o banco de dados usando JPA (Java Persistence Application).

Na linha 7, temos o EntityManagerFactory emFactoy que representa Fábrica que cria conexões com o banco.

Na linha 10, o método “Persistence.createEntityManagerFactory()” cria a fábrica, já o parâmetro "agendaPersist" representa o nome da unidade de persistência definida no arquivo persistence.xml.

Na linha 13, o emFactory pede para criar uma nova entidade de conexão com o banco de dados.

3.4) APRESENTAÇÃO DO CÓDIGO DO ARQUIVO PERSISTENCE.XML

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7 <persistence-unit name="agendaPersist" transaction-type="RESOURCE_LOCAL">
8 <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
9 <class>model.Contato</class>
```

```

persistence.xml X
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd (xsi:schemaLocation with catalog)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6 version="2.1">
7 <persistence-unit name="agendaPersist" transaction-type="RESOURCE_LOCAL">
8 <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
9 <class>model.Contato</class>
10 <properties>
11 <property name="javax.persistence.jdbc.url"
12 value="jdbc:mysql://localhost:3306/agenda?serverTimezone=UTC"/>
13 <property name="javax.persistence.jdbc.user" value="root"/>
14 <property name="javax.persistence.jdbc.password" value="150589"/>
15 <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
16 <property name="hibernate.show_sql" value="true"/>
17 <property name="hibernate.format_sql" value="true"/>
18 <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
19 <property name="hibernate.hbm2ddl.auto" value="update"/>
20 </properties>
21 </persistence-unit>
22 </persistence>

```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.4

O arquivo persistence.xml é o arquivo mais importante do JPA, nele é dito com se conectar com o banco de dados especificando: o banco, senha, propriedades da conexão, porta, etc...

```

13 <property name="javax.persistence.jdbc.user" value="root"/>
14 <property name="javax.persistence.jdbc.password" value="150589"/>
15 <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
16 <property name="hibernate.show_sql" value="true"/>

```

Na linha 13 temos o value “**root**” que é o nome da instancia do banco de dados, como também, a senha de acesso na linha 14, o conector jdbc driver na linha 15.

Na linha 16 a propriedade: **name="hibernate.show_sql" value="true"** mostra o sql no console do eclipse, muito utilizado para entender o código no processo de inspeção e “TheBug”.

Na linha 18 na propriedade **value="org.hibernate.dialect.MySQL5Dialect"** mostra o dialeto SQL usado no MySQL 5 usado no workbench.

3.5) APRESENTAÇÃO DO CÓDIGO DO ARQUIVO CONTATO.XHTML

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3<html xmlns="http://www.w3.org/1999/xhtml"
4      xmlns:h="http://xmlns.jcp.org/jsf/html">
5<h:head>
6    <meta charset="UTF-8" />
7    <title>Agenda Virtual</title>
8    <meta name="viewport" content="width=device-width, initial-scale=1" />
9    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" />
10   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"></script>
11 </h:head>
12<h:body>
13<div class="container mt-4 border rounded" >
14    <h1 class="mb-4">Agenda</h1>
15
16<h:form>
17
18<div class="mb-3" >
19    <label class="form-label" >Nome:</label>
20    <h:inputText value="#{contatoMB.nome}" styleClass="form-control w-50" />
21 </div>
22
23<div class="mb-3" >
24    <label class="form-label" >Sobrenome:</label>
25    <h:inputText value="#{contatoMB.sobrenome}" styleClass="form-control w-50" />
26 </div>
27
28<div class="mb-3" >
29    <label class="form-label" >Telefone:</label>
30    <h:inputText value="#{contatoMB.telefone}" styleClass="form-control w-50" />
31 </div>
32
33<div class="mb-3" >
34    <label class="form-label" >E-mail:</label>
35    <h:inputText value="#{contatoMB.email}" styleClass="form-control w-50" />
36 </div>
37
38<div class="mt-4" >
39    <h:commandButton value="Salvar" action="#{contatoMB.salvar}" styleClass="btn btn-primary" />
40 </div>
41 <div class="mt-4" >
42    <h:link outcome="Listar" value="Listar Todos" styleClass="btn btn-primary" />
43 </div>
44 </h:form>
45
46<div class="mt-4 p-3 border rounded bg-light" >
47    <h5 class="text-primary" >Contato Cadastrado:</h5>
48    <p class="mb-1" >Código Id: #{contatoMB.id}</p>
49    <p class="mb-1" >Nome: #{contatoMB.nome} #{contatoMB.sobrenome}</p>
50    <p class="mb-1" >Telefone: #{contatoMB.telefone}</p>
51    <p class="mb-1" >E-mail: #{contatoMB.email}</p>
52
53 </div>
54 </div>
55 </div>
56 </h:body>
57 </html>
```

BREVE EXPLICAÇÃO DO CÓDIGO ACIMA TÓPICO 3.5

O arquivo CONTATO.XHTML é o arquivo front end sendo a interface visual de interação com o usuário. Para essa interface, ao invés de usar um arquivo estilo.css foi utilizado o boot strap para trazer um designer mais profissional de forma mais simples.

```
7      <title>Agenda Virtual</title>
8      <meta name="viewport" content="width=device-width, initial-scale=1" />
9      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" />
10     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"></script>
11  </head>
```

Na linha 9 e 10 trazemos a conexão com o boot strap. O boot strap é uma framework com uma estrutura de código que facilita no processo de estilização da interface gráfica.

```
16•      <h:form>
17
18•          <div class="mb-3">
19              <label class="form-label">Nome:</label>
20              <h:inputText value="#{contatoMB.nome}" styleClass="form-control w-50" />
21          </div>
22
```

Na linha 20 temos a tag **<h:inputText>** representado um componente JSF para campo de texto. O styleClass = "form-control w-50" significa que o a caixa de entrada terá 50% da largura da tela.

O valor digitado pelo usuário sera encaminhado para o arquivo contatoMB.nome que ativará o objeto C com os métodos get e set enviado os dados para o contato.java,

```
38
39•          <div class="mt-4">
40              <h:commandButton value="Salvar" action="#{contatoMB.salvar}" styleClass="btn . . .
41          </div>
42•          <div class="mt-4">
43              <h:link outcome="Listar" value="Listar Todos" styleClass="btn btn-primary" />
44          </div>
45      </h:form>
46
```

Na linha 40, ao clicar no botão salvar, o método salvar da classe contatoMB. é ativado.

Ao acionar o método salvar, os dados percorrem o arquivo contato.xhtml, passando pelo contatoMB.java e por fim ele chega ao contato.java; porém, na classe contatoMB, quando o método salvar é ativado, é invocado também o método inserir da classe CRUD, fazendo que todos os dados sejam armazenados no banco de dados.

```
public void salvar() {
    try {
        System.out.println("== TENTANDO SALVAR ==");
        System.out.println("Nome: " + c.getNome());
        System.out.println("Sobrenome: " + c.getSobrenome());
        System.out.println("Telefone: " + c.getTelefone());
        System.out.println("Email: " + c.getEmail());

        CRUD.inserir(c);
    }
}
```

```
9 public class CRUD {
10     public static void inserir(Contato c1) {
11
12         try {
13             EntityManager entityManager = DAO.getEntityManager();
14
15             entityManager.getTransaction().begin();
16             entityManager.persist(c1);
17             entityManager.getTransaction().commit();
18             entityManager.close();
19             System.out.println("conectado Salvo!");
20         } catch(Exception ex) {
21             ex.printStackTrace();
22         }
23     }
24 }
```

E assim ocorre o processo de inserção de dados, processamento e armazenamento.

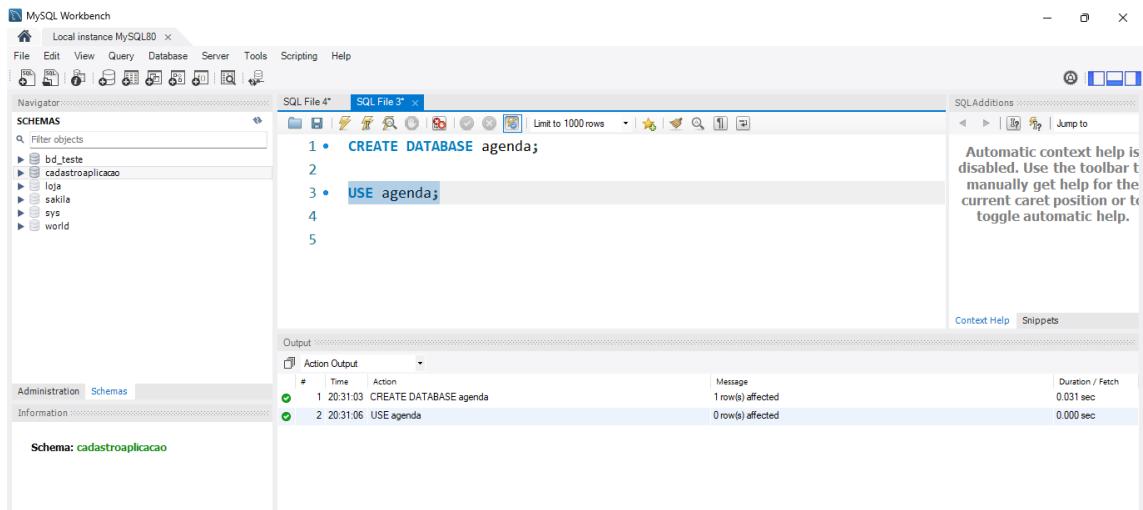
3.5) APRESENTAÇÃO DO CÓDIGO LISTAR.XHTML

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3<html xmlns="http://www.w3.org/1999/xhtml"
4      xmlns:h="http://xmlns.jcp.org/jsf/html"
5      xmlns:f="http://xmlns.jcp.org/jsf/core">
6<h:head>
7     <meta charset="UTF-8" />
8     <title>Lista de Contatos</title>
9     <meta name="viewport" content="width=device-width, initial-scale=1" />
10    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css" rel="stylesheet" />
11    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"></script>
12 </h:head>

13<h:body>
14    <div class="container mt-4">
15        <h1 class="mb-4">Lista de Contatos</h1>
16        |
17        <h:link outcome="contato" value="Novo Contato" styleClass="btn btn-success mb-3" />
18
19<h: dataTable value="#{contatoMB.listaContatos}" var="contato"
20      styleClass="table table-striped table-bordered"
21      headerClass="table-dark">
22
23    <h:column>
24        <f:facet name="header">ID</f:facet>
25        #{contato.id}
26    </h:column>
27
28    <h:column>
29        <f:facet name="header">Nome</f:facet>
30        #{contato.nome} #{contato.sobrenome}
31    </h:column>
32
33    <h:column>
34        <f:facet name="header">Telefone</f:facet>
35        #{contato.telefone}
36    </h:column>
37
38    <h:column>
39        <f:facet name="header">Email</f:facet>
40        #{contato.email}
41    </h:column>
42
43    </h: dataTable>
44
45    <h:link outcome="contato" value="Voltar" styleClass="btn btn-secondary" />
46  </div>
47 </h:body>
48 </html>
```

O arquivo **listar.xhtml** apresenta na tela de resultados que foram cadastrados e estão contidos no banco de dados.

3.5) APRESENTAÇÃO DO BANCO DE DADOS



The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'Local instance MySQL80' is selected. The 'Navigator' pane on the left shows the 'SCHEMAS' section with databases: bd_teste, cadastroaplicacao, loja, sakila, sys, and world. The 'SQL File 3*' tab in the center contains the following SQL code:

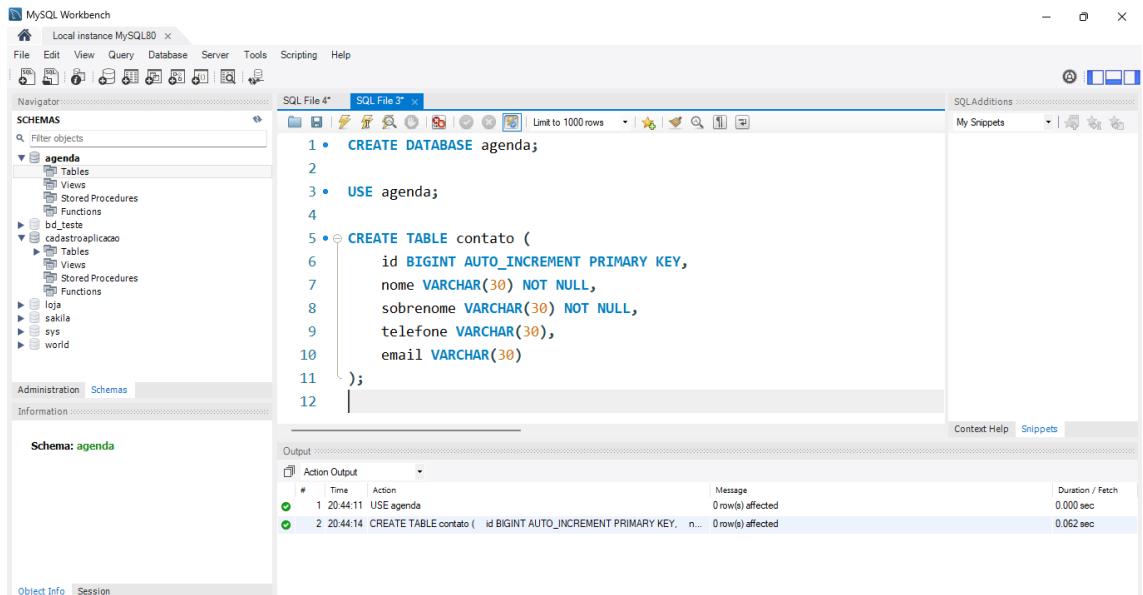
```
1 • CREATE DATABASE agenda;
2
3 • USE agenda;
```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	20:31:03	CREATE DATABASE agenda	1 row(s) affected	0.031 sec
2	20:31:06	USE agenda	0 row(s) affected	0.000 sec

The status bar at the bottom indicates the schema is 'cadastroaplicacao'.

Criação do banco de dados agenda, criação da coluna contato e seus respectivos atributos.



The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'Local instance MySQL80' is selected. The 'Navigator' pane on the left shows the 'SCHEMAS' section with the 'agenda' database selected. The 'SQL File 4*' tab in the center contains the following SQL code:

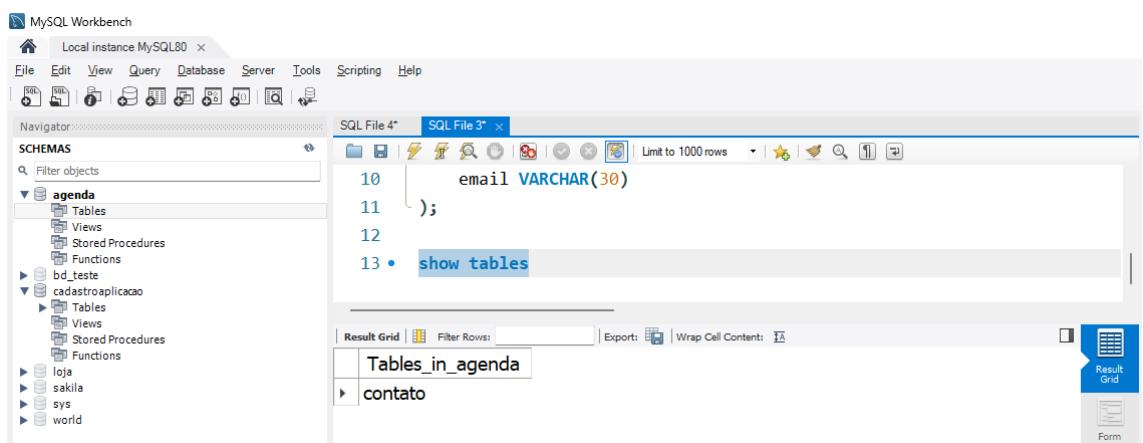
```
1 • CREATE DATABASE agenda;
2
3 • USE agenda;
4
5 • CREATE TABLE contato (
6     id BIGINT AUTO_INCREMENT PRIMARY KEY,
7     nome VARCHAR(30) NOT NULL,
8     sobrenome VARCHAR(30) NOT NULL,
9     telefone VARCHAR(30),
10    email VARCHAR(30)
11 );
12
```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	20:44:11	USE agenda	0 row(s) affected	0.000 sec
2	20:44:14	CREATE TABLE contato (id BIGINT AUTO_INCREMENT PRIMARY KEY, n...)	0 row(s) affected	0.062 sec

The status bar at the bottom indicates the schema is 'agenda'.

RESULTADO DA CRIAÇÃO DAS TABELAS



The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'Local instance MySQL80' is selected. The 'Navigator' pane on the left shows the 'SCHEMAS' section with the 'agenda' database selected. The 'SQL File 4*' tab in the center contains the following SQL code:

```
10     email VARCHAR(30)
11 );
12
13 • show tables
```

The 'Result Grid' pane at the bottom shows the results of the 'show tables' command:

Tables_in_agenda
contato

The status bar at the bottom indicates the schema is 'agenda'.

RESULTADO DO SELECT NA TABELA

Observe que o banco de dados está vazio nesse momento.

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the database schema with the 'agenda' database selected. The 'Tables' section under 'agenda' is empty. The 'SQL File 4*' tab contains the following SQL code:

```
12
13 • show tables
14
15 ✘ select * from contato;
```

The 'Result Grid' pane below shows a table with columns: id, nome, sobrenome, telefone, email. There is one row with all columns set to 'NULL'.

RESULTADO FINAL

The screenshot shows a web browser window at the URL `localhost:8080/Agenda/contato.xhtml`. The page title is 'Agenda'. It contains a form with the following fields:

- Nome:
- Sobrenome:
- Telefone:
- E-mail:

Below the form are two buttons: 'Salvar' (Save) and 'Listar Todos' (List All). The browser's address bar and various icons are visible at the top.

Após “Salvar” e clicar em “Listar Todos”, temos:

The screenshot shows a web application window titled "Lista de Contatos". At the top left is a green button labeled "Novo Contato". Below it is a table with three columns: "ID", "Nome", "Telefone", and "Email". A single row is displayed, containing the values: ID 1, Nome "THIAGO TESTE UM MORAIS", Telefone "31900000000", and Email "teste_morais@gmail.com". At the bottom left of the table is a grey button labeled "Voltar". The browser's address bar shows the URL "localhost:8080/Agenda/listar.xhtml". The status bar at the bottom of the browser indicates "Todos os favoritos".

Ao retornar ao banco de dados, temos:

The screenshot shows the MySQL Workbench interface. On the left, the "Navigator" pane displays database schemas: "agenda", "bd_teste", "cadastroaplicacao", "loja", "sakila", "sys", and "world". The "agenda" schema is expanded, showing "Tables", "Views", "Stored Procedures", and "Functions". The "SQL File 3" tab contains the SQL query: "select * from contato;". The "Result Grid" pane shows the results of the query, displaying one row of data: id 1, nome "THIAGO TESTE UM", sobrenome "MORAIS", telefone "31900000000", and email "teste_morais@gmail.com". Red arrows point from the query line in the editor to the result grid. The "Output" pane at the bottom shows the log entry: "13 20:53:43 select * from contato LIMIT 0, 1000" and "1 row(s) returned".

CONSIDERAÇÕES FINAIS

Para executar esse trabalho foi possível superar muitos desafios passando pelos principais tópicos:

- Entendimento dos conceitos do Hibernate e ORM na prática;
- Junção de todos os conhecimentos: HTML, Boot Strap, JAVA Front End, Back End, Banco de dados.
- Assimilação dos conceitos do DAO e do arquivo persisntece.xml

Nessa jornada concluí que o hibernate facilita e sintetiza o processo de escrita de código otimizando o processo de desenvolvimento de sistema.

REFERÊNCIAS

- 1) DEV.MEDIA. Guia Completo de Hibernate: Aprenda Hibernate do Básico ao Avançado, 2026 Disponível em m: <https://www.devmedia.com.br/guia/hibernate>. Acesso em 18 fev. 2026.
- 2) DEV.MEDIA. Guia completo de JSF (JavaServer Faces): aprenda JSF agora! Disponível em: <https://www.devmedia.com.br/guia/jsf-javaserver>. Acesso em: [Março 2026]
- 3) CONFIGURANDO O SERVIDOR WILDFLY COM A IDE ECLIPSE. Disponível em: <https://www.youtube.com/watch?v=pKqQXSUypow>. Acesso em: 13 fev. 2026.