Trabalhando com Banco de Dados Utilizando JDBC e JPA

Descrição	Aprenda sobre JDBC e JPA, frameworks da linguagem Java para otimização na hora de trabalhar com banco de dados.
Produzido	AndersonFroes - https://andersonfroes.github.io/Portfolio/

`Formação com esse curso:`

1. Inter Java Developer

`Professor:`

Daniel Karam

`Aulas:`

- **Matéria**
  - **Capitulo 1 - Introdução ao JDBC**

    Material Professor GitHub:

    https://github.com/danielkv7/digital-innovation-one/tree/master/jdbc-basico

    Slide do Professor:

    https://github.com/danielkv7/digital-innovation-one/blob/master/Aula_JDBC_basico/Template padrão de apresentação.pptx

    Link Download MySQL para Windows:

    https://dev.mysql.com/downloads/installer/

    Parte 1: Instruções para Instalar o Banco de Dados - MySQL

    1 - Instalar MySQL no Ubuntu

    - Atualizar repositório: sudo apt update
    - Instalar MySQL: sudo apt install mysql-server
    - Verificar se instalação foi um sucesso: mysql --version
    - (OPCIONAL) Trocar valores defaults para aumentar segurança: sudo mysql_secure_installation

    2 - Configurar Usuário e Senha

    - Acessar banco de dados MySQL: sudo mysql
    - Adicionar senha "password" ao usuário "root" (Rodar no prompt do MySQL)
      ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';
    - Recarregar permissões de acesso ao banco de dados (Rodar no prompt do MySQL)
      FLUSH PRIVILEGES;
    - Sair do prompt do MySQL: quit;
      Obs: A partir de agora, para acessar o mysql deverá ser utilizado o comando abaixo.
      Quando pedir a senha, deverá colocar a senha: password
      mysql -u root -p

    3 - Instalar MySQL Workbench (Opcional)

    - Atualizar repositório: sudo apt update
    - Instalar MySQL Workbench: sudo apt install mysql-workbench
    - Executar MySQL Workbench (também pode executar ao pesquisar por "workbench" em uma GUI do linux): mysql-workbench

    4 - Criar Banco de Dados

    - Acessar banco de dados. Pode ser workbench ou linha de comando: mysql -u root -p (Enter password:) password
    - Criar um banco de dados (rodar no prompt do MySQL OU no MySQL workbench): CREATE database digital_innovation_one;
    - Usar o banco recém criado (digital_innovation_one) (rodar no prompt do MySQL OU no MySQL workbench): USE digital_innovation_one;

    5 - Criar uma Tabela

    - Acessar banco de dados. Pode ser workbench ou linha de comando: mysql -u root -p (Enter password:) password
    - (CASO NÂO ESTEJA NO BANCO DE DADOS) Mudar para o banco digital_innovation_one (rodar no prompt do MySQL OU no MySQL workbench): USE digital_innovation_one;
    - Criar uma tabela no banco de dados (rodar no prompt do MySQL OU no MySQL workbench)

```
CREATE TABLE aluno (
id INTEGER PRIMARY KEY AUTO_INCREMENT,
nome VARCHAR(80) NOT NULL,
idade INTEGER NOT NULL,
estado CHARACTER(2) NOT NULL
);
```

    - Adicionar alguns exemplos (rodar no prompt do MySQL OU no MySQL workbench)

```
INSERT INTO aluno(nome, idade, estado) VALUES ('Pedro', 20, 'RJ');
INSERT INTO aluno(nome, idade, estado) VALUES ('Maria', 35, 'AC');
INSERT INTO aluno(nome, idade, estado) VALUES ('Joao', 10, 'SC');
INSERT INTO aluno(nome, idade, estado) VALUES ('Ana', 51, 'GO');
```

    - No gradle deve-se adicionar no "build.gradle" (na parte de "dependencies") a linha abaixo
      compile group: 'mysql', name: 'mysql-connector-java', version: '8.0.17'

Parte 2: JDBC e Drivers de Conexão

Arquivo: ConnectionJDBC.java

```java
package part2;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionJDBC {

    public static void main(String[] args) {

        // 1 - NÃO ESQUECER DE BAIXAR O DRIVER PARA O BANCO DE DADOS QUE IRÁ UTILIZAR! (Como demonstrado na parte 1 do curso)

        // 2 - Definir parâmetros para se conectar ao banco de dados MySQL.
        String driver = "mysql";
        String dataBaseAddress = "localhost";
        String dataBaseName = "digital_innovation_one";
        String user = "root";
        String password = "password";

        // 3 - Construção da string de conexão.
        StringBuilder sb = new StringBuilder("jdbc:")
                .append(driver).append("://")
                .append(dataBaseAddress).append("/")
                .append(dataBaseName);

        String connectionUrl = sb.toString(); //sb.toString() == "jdbc:mysql://localhost/digital_innovation_one"

        //Carregar a classe específica de implementação do driver na memória da JVM. (A partir da versão JDBC 4 não é mais necessário!!!)
        //Class.forName("com.mysql.jdbc.Driver");

        // 4 - Criar conexão usando o DriverManager, passando como parâmetros a string de conexão, usuário e senha do usuário.
        try (Connection conn = DriverManager.getConnection(connectionUrl, user, password)) {
            System.out.println("SUCESSO ao se conectar ao banco MySQL!");
        } catch (SQLException e) {
            System.out.println("FALHA ao se conectar ao banco MySQL!");
            e.printStackTrace();
        }

    }
}
```

Parte 3: Consultas com JDBC

Arquivo: Aluno.java

```java
package part3;

public class Aluno {

    private int id;
    private String nome;
    private int idade;
    private String estado;

    public Aluno(int id, String nome, int idade, String estado) {
        this.id = id;
        this.nome = nome;
        this.idade = idade;
        this.estado = estado;
    }

    public Aluno(String nome, int idade, String estado) {
        this.nome = nome;
        this.idade = idade;
        this.estado = estado;
    }

    public Aluno() { }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public String getEstado() {
        return estado;
    }

    public void setEstado(String estado) {
        this.estado = estado;
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Aluno{");
        sb.append("id=").append(id);
        sb.append(", nome='").append(nome).append('\'');
        sb.append(", idade=").append(idade);
        sb.append(", estado='").append(estado).append('\'');
        sb.append('}');
        return sb.toString();
    }
}
```

Arquivo: AlunoDAO.java

```java
package part3;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class AlunoDAO {

    // 1 - Consulta
    public List<Aluno> list() {
        //Preparar lista que irá retornar alunos após consultar o banco de dados;
        List<Aluno> alunos = new ArrayList<>();

        try (Connection conn = ConnectionFactory.getConnection()) {
            //Preparar consulta SQL.
            String sql = "SELECT * FROM aluno";

            //Preparar statement com os parâmetros recebidos (nesta função não tem parâmetros, pois irá retornar todos os valores da tabela aluno)
            PreparedStatement stmt = conn.prepareStatement(sql);

            //Executa consulta e armazena o retorno da consulta no objeto "rs".
            ResultSet rs = stmt.executeQuery();

            //Criar um objeto aluno e guardar na lista de alunos.
            while(rs.next()){
                int id = rs.getInt("id");
                String nome = rs.getString("nome");
                int idade = rs.getInt("idade");
                String estado = rs.getString("estado");

                alunos.add(new Aluno(
                        id,
                        nome,
                        idade,
                        estado
                ));
            }
        } catch (SQLException e) {
            System.out.println("Listagem de alunos FALHOU!");
            e.printStackTrace();
        }

        //Retornar todos os alunos encontrados no banco de dados.
        return alunos;
    }

    // 1.1 - Consulta com filtro
    public Aluno getById(int id) {
        //Preparar objeto aluno para receber os valores do banco de dados.
        Aluno aluno = new Aluno();

        try (Connection conn = ConnectionFactory.getConnection()) {
            //Preparar consulta SQL
            String sql = "SELECT * FROM aluno WHERE id = ?";

            //Preparar statement com os parâmetros recebidos
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1, id);

            //Executa consulta e armazena o retorno da consulta no objeto "rs".
            ResultSet rs = stmt.executeQuery();

            //Guardar valores retornados da tabela aluno no objeto aluno
            if (rs.next()){
                aluno.setId(rs.getInt("id"));
                aluno.setNome(rs.getString("nome"));
                aluno.setIdade(rs.getInt("idade"));
                aluno.setEstado(rs.getString("estado"));
            }

        } catch (SQLException e) {
            System.out.println("Listagem de alunos FALHOU!");
            e.printStackTrace();
        }

        //Retorna aluno encontrado no banco de dados.
        return aluno;
    }

    // 2 - Inserção
    public void create(Aluno aluno) {
        try (Connection conn = ConnectionFactory.getConnection()) {

            //Preparar SQL para inserção de dados do aluno.
            String sql = "INSERT INTO aluno(nome, idade, estado) VALUES(?, ?, ?)";

            //Preparar statement com os parâmetros recebidos
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1 , aluno.getNome());
            stmt.setInt(2, aluno.getIdade());
            stmt.setString(3 , aluno.getEstado());

            //Executa inserção e armazena o numero de linhas afetadas
            int rowsAffected = stmt.executeUpdate();

            System.out.println("Inserção BEM SUCEDIDA!. Foi adicionada " + rowsAffected + " linha");
        } catch (SQLException e) {
            System.out.println("Inserção FALHOU!");
            e.printStackTrace();
        }
    }

    // 3 - Delete
    public void delete(int id) {
        try (Connection conn = ConnectionFactory.getConnection()) {

            //Preparar SQL para deletar uma linha.
            String sql = "DELETE FROM aluno WHERE id = ?";

            //Preparar statement com os parâmetros recebidos
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setInt(1 , id);

            //Executa delete e armazena o numero de linhas afetadas
            int rowsAffected = stmt.executeUpdate();
```

```
            System.out.println("Delete BEM SUCEDIDA! Foi deletada " + rowsAffected + " linha");
        } catch (SQLException e) {
            System.out.println("Delete FALHOU!");
            e.printStackTrace();
        }
    }

    // 4 - Atualizar
    public void update(Aluno aluno) {
        try (Connection conn = ConnectionFactory.getConnection()) {

            //Preparar SQL para atualizar linhas.
            String sql = "UPDATE aluno SET nome = ?, idade = ?, estado = ? WHERE id = ?";

            //Preparar statement com os parâmetros recebidos
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, aluno.getNome());
            stmt.setInt(2, aluno.getIdade());
            stmt.setString(3, aluno.getEstado());
            stmt.setInt(4, aluno.getId());

            //Executa atualização e armazena o numero de linhas afetadas
            int rowsAffected = stmt.executeUpdate();

            System.out.println("Atualização BEM SUCEDIDA! Foi atualizada: " + rowsAffected + " linha");
        } catch (SQLException e) {
            System.out.println("Atualização FALHOU!");
            e.printStackTrace();
        }
    }

}
```

Arquivo: ConnectionFactory.java

```
package part3;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class ConnectionFactory {

    //Construtor declarado como privado. Evitando assim criar instâncias da fábrica.
    private ConnectionFactory() {
        throw new UnsupportedOperationException();
    }

    public static Connection getConnection() {

        // OBS: NÃO ESQUECER DE BAIXAR O DRIVER PARA O BANCO DE DADOS QUE IRÁ UTILIZAR! (Como demonstrado na parte 1 do curso)

        // 1 - Declarar objeto conexão (irá receber uma conexão após executar os passos abaixo)
        Connection connection = null;

        // 2 - Carregar arquivo de propriedades para pegar parâmetros necessários para se comunicar com o banco de dados
        try (InputStream input = ConnectionFactory.class.getClassLoader().getResourceAsStream("connection.properties")) {

            // 3 - Definir parâmetros para se conectar ao banco de dados MySQL.
            Properties prop = new Properties();
            prop.load(input);

            String driver = prop.getProperty("jdbc.driver");
            String dataBaseAddress = prop.getProperty("db.address");
            String dataBaseName = prop.getProperty("db.name");
            String user = prop.getProperty("db.user.login");
            String password = prop.getProperty("db.user.password");

            // 4 - Construção da string de conexão.
            StringBuilder sb = new StringBuilder("jdbc:")
                    .append(driver).append("://")
                    .append(dataBaseAddress).append("/")
                    .append(dataBaseName);

            String connectionUrl = sb.toString(); //sb.toString() == "jdbc:mysql://localhost/digital_innovation_one"

            // 5 - Criar conexão usando o DriverManager, passando como parâmetros a string de conexão, usuário e senha do usuário.
            try {
                connection = DriverManager.getConnection(connectionUrl, user, password);
            } catch (SQLException e) {
                System.out.println("FALHA ao tentar criar conexão");
                throw new RuntimeException(e);
            }

        } catch (IOException e) {
            System.out.println("FALHA ao tentar carregar aquivos de propriedades");
            e.printStackTrace();
        }

        return connection;
    }
}
```

Arquivo: QueriesExecution.java

```
package part3;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class QueriesExecution {

    public static void main(String[] args) {

        AlunoDAO alunoDAO = new AlunoDAO();

        // ========================== 1 - Consulta ================================================
        List<Aluno> alunos = alunoDAO.list();
```

```
        alunos.stream().forEach(System.out::println);


        // ====================== 1.1 - Consulta com filtro ======================================
        Aluno alunoParaConsulta = alunoDAO.getById(1);

        //System.out.println(alunoParaConsulta);


        // ========================== 2 - Inserção ================================================
        Aluno alunoParaInsercao = new Aluno(
                "Matheus",
                43,
                "SP"
        );

        //alunoDAO.create(alunoParaInsercao);


        // ========================== 3 - Delete ==================================================
        //alunoDAO.delete(1);


        // ========================== 4 - Atualizar ===============================================
        Aluno alunoParaAtualizar = alunoDAO.getById(3);
        alunoParaAtualizar.setNome("Joaquim");
        alunoParaAtualizar.setIdade(18);
        alunoParaAtualizar.setEstado("RS");

        //alunoDAO.update(alunoParaAtualizar);
    }

}
```

▼ **Capitulo 2 - Introdução ao JPA**

Slide do Professor:

https://github.com/danielkv7/digital-innovation-one/blob/master/Aula_JPA_basico/Aula_JPA_basico_windows.pptx

Parte 1: Entendendo o JPA e Começando o Mapeamento do Banco

Arquivo: persistence.xml

```
<?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_2.xsd"
             version="2.2">

    <!-- Unidade de persistencia da parte 1 do curso (Somente JPA)-->
    <persistence-unit name="part1-DIO">

        <description> Unidade de persistencia da parte 1 do tutorial basico de JPA da Digital Innovation One sem implementacoes (Somente JPA) </description>

        <!-- Classes (entidades) que serao mapeadas -->
        <class>classes.Aluno</class>
        <class>classes.Estado</class>

        <!-- Configuracoes de conexao ao banco de dados -->
        <properties>
            <!-- Configuracoes do banco de dados -->
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/digital_innovation_one" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password" value="password" />
            <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver" />
        </properties>

    </persistence-unit>

    <!-- ================================================================================================================  -->

    <!-- Unidade de persistencia da parte 2 do curso (Com implementacao Hibernate ou EclipseLink) -->
    <persistence-unit name="part2-DIO">

        <description> Unidade de persistencia da parte 2 do tutorial basico de JPA da Digital Innovation One com implementacoes (Hibernate ou EclipseLink) </description>

        <!-- Implementacao do JPA -->
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <!--  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider> -->

        <!-- Classes (entidades) que serao mapeadas -->
        <class>classes.Aluno</class>
        <class>classes.Estado</class>

        <!-- Configuracoes de conexao ao banco de dados e do Hibernate/EclipseLink -->
        <properties>
            <!-- Configuracoes do banco de dados -->
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/digital_innovation_one" />
            <property name="javax.persistence.jdbc.user" value="root" />
            <property name="javax.persistence.jdbc.password" value="password" />
            <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver" />

            <!-- Configuracoes do Hibernate (os parametros so sao reconhecidos se estiver usando a implementacao do Hibernate)-->
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect" />
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.hbm2ddl.auto" value="create" />  <!-- Possible values for hibernate.hbm2ddl.auto are: validate, update, create, create-drop -->

            <!-- Configuracoes do EclipseLink (os parametros so sao reconhecidos se estiver usando a implementacao do EclipseLink) -->
<!--            <property name="eclipselink.target-database" value="MySQL"/>-->
<!--            <property name="eclipselink.logging.level.sql" value="FINE" />-->
<!--            <property name="eclipselink.logging.parameters" value="true" />-->
<!--            <property name="eclipselink.ddl-generation" value="drop-and-create-tables" />-->
        </properties>

    </persistence-unit>

</persistence>
```

Arquivo: Aluno.java

```java
package classes;

import javax.persistence.*;

@Entity
public class Aluno {

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

    @Column(nullable = false)
    private String nome;

    @Column(nullable = false)
    private int idade;

    @ManyToOne(fetch = FetchType.LAZY)
    private Estado estado;

    public Aluno() { }

    public Aluno(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    public Aluno(String nome, int idade, Estado estado) {
        this.nome = nome;
        this.idade = idade;
        this.estado = estado;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    public Estado getEstado() {
        return estado;
    }

    public void setEstado(Estado estado) {
        this.estado = estado;
    }

    @Override
    public String toString() {
        return "Aluno{" +
                "id=" + id +
                ", nome='" + nome + '\'' +
                ", idade=" + idade +
                ", estado=" + estado +
                '}';
    }
}
```

Arquivo: Estado.java

```java
package classes;
import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Entity
public class Estado {

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

    @Column(nullable = false)
    private String nome;

    @Column(nullable = false)
    private String sigla;

    @OneToMany(
            mappedBy = "estado",
            cascade = CascadeType.ALL,
            orphanRemoval = true
    )
    private List<Aluno> alunos = new ArrayList<>();

    public Estado() { }

    public Estado(String nome, String sigla) {
        this.nome = nome;
        this.sigla = sigla;
    }

    public Estado(String nome, String sigla, List<Aluno> alunos) {
        this.nome = nome;
        this.sigla = sigla;
        this.alunos = alunos;
    }
```

```
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

    public List<Aluno> getAlunos() {
        return alunos;
    }

    public void setAlunos(List<Aluno> alunos) {
        this.alunos = alunos;
    }

    @Override
    public String toString() {
        return "Estado{" +
                "id=" + id +
                ", nome='" + nome + '\'' +
                ", sigla='" + sigla + '\'' +
                ", alunos=" + alunos +
                '}';
    }
}
```

Arquivo: ExecutionPart1.java

```
package part1;

import classes.Aluno;
import classes.Estado;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class ExecutionPart1 {
    public static void main(String[] args) {

        // OBS: Esse codigo pode ou nao funcionar de acordo com a biblioteca que foi baixada. Se estiver somente com o JPA baixado, o codigo NAO IRA funcionar.
        // porem se estiver com a biblioteca de algum framework com implementacao JPA (Hibernate ou EclipseLink), o JPA irá automaticamente utiliza-lo.

        // O ideal eh que nessa parte (Parte 1) o codigo EXECUTE COM ERROR (Ao tentar executar irá mostrar um error afirmando que não foi encontradado nenhuma implementação do JPA).
        // pois aqui nao deveria ter nenhuma implementacao JPA sendo utilizada, apenas o JPA puro para demonstrar que através dele é possivel definir a estrutura do codigo e depois escolher
        // a implementacao que ira utilizar. Apenas na parte 2 do curso sera escolhida uma implementacao para o codigo executar sem error

        // 1 - Passos iniciais para criar um gerenciador de entidades com o banco de dados especificado no arquivo "persistence.xml"
        EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("part1-DIO");
        EntityManager entityManager = entityManagerFactory.createEntityManager();

        // 2.1 - Criar instancias para serem adicionadas no banco de dados
        Estado estadoParaAdicionar = new Estado("Rio de Janeiro", "RJ");
        Aluno alunoParaAdicionar = new Aluno("Daniel", 29, estadoParaAdicionar);

        // 2.2 - Iniciar uma trasacao para adiconar as instancias no banco de dados
        entityManager.getTransaction().begin();

        entityManager.persist(estadoParaAdicionar);
        entityManager.persist(alunoParaAdicionar);

        entityManager.getTransaction().commit();

        // 3 - Encerrar o gerenciador de entidades e encerrar a fabrica de gerenciadores de entidade.
        entityManager.close();
        entityManagerFactory.close();

    }
}
```

Parte 2: Implementações do JPA (Hibernate e EclipeLink)

Arquivo: ExecutionPart2.java

```
package part2;

import classes.Aluno;
import classes.Estado;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class ExecutionPart2 {
    public static void main(String[] args) {

        // OBS: Esse codigo deve executar SEM ERROS com a implementacao JPA que foi definida no "persistence.xml".
        // Se estiver somente com o JPA baixado, o codigo NAO IRA funcionar.

        // O ideal é que nessa parte (Parte 2) o codigo EXECUTE SEM ERROS, pois aqui tera uma implementacao JPA sendo utilizada.

        // 1 - Passos iniciais para criar um gerenciadop de entidades com o banco de dados especificado no arquivo  "persistence.xml"
        EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("part2-DIO");
        EntityManager entityManager = entityManagerFactory.createEntityManager();

        // 2.1 - Criar instancias para serem adicionadas no banco de dados
        Estado estadoParaAdicionar = new Estado("Rio de Janeiro", "RJ");
```

```
        Aluno alunoParaAdicionar = new Aluno("Daniel", 29, estadoParaAdicionar);

        // 2.2 - Iniciar uma trasacao para adiconar as instancias no banco de dados
        entityManager.getTransaction().begin();

        entityManager.persist(estadoParaAdicionar);
        entityManager.persist(alunoParaAdicionar);

        entityManager.getTransaction().commit();

        // 3 - Resgatar instâncias no banco de dados
//        Estado estadoEncontrado = entityManager.find(Estado.class, 1);
//        Aluno alunoEncontrado = entityManager.find(Aluno.class, 1);
//
//        System.out.println(estadoEncontrado);
//        System.out.println(alunoEncontrado);

        // 4 - Alterar uma entidade
//        entityManager.getTransaction().begin();
//
//        alunoEncontrado.setNome("Karam");
//        alunoEncontrado.setIdade(20);
//
//        entityManager.getTransaction().commit();

        // 5 - Remover uma entidade
//        entityManager.getTransaction().begin();
//
//        entityManager.remove(alunoEncontrado);
//
//        entityManager.getTransaction().commit();

        // 6 - Encerrar o gerenciador de entidades e encerrar a fabrica de gerenciadores de entidade.
        entityManager.close();
        entityManagerFactory.close();

    }
}
```

Parte 3: Linguagens de Consulta Orientada a Objetos

Arquivo: ExecutionPart3.java

```
package part3;

import classes.Aluno;
import classes.Estado;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import java.util.List;

public class ExecutionPart3 {

    public static void main(String[] args) {

        // 1 - Dados instanciados para serem utilizados como exemplo
        EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("part2-DIO");
        EntityManager entityManager = entityManagerFactory.createEntityManager();
        entityManager.getTransaction().begin();
        Estado estadoParaAdicionar = new Estado("Rio de Janeiro", "RJ");
        entityManager.persist(estadoParaAdicionar);
        entityManager.persist(new Estado("Sao Paulo", "SP"));
        entityManager.persist(new Aluno("Daniel", 29, estadoParaAdicionar));
        entityManager.persist(new Aluno("Joao", 20, estadoParaAdicionar));
        entityManager.persist(new Aluno("Pedro", 30, estadoParaAdicionar));
        entityManager.getTransaction().commit();

        // 2 - Vamos utilizar o método do EntityManager find(), SQL nativo, JPQL e JPA Criteria API para realizar uma
        // consulta que retornarar o mesmo valor equivalente aos seguintes SQL:
        // SELECT * FROM Aluno WHERE id = 1 (Equivalente ao método find do entityManager na parte 2.2)
        // SELECT * FROM Aluno WHERE nome = 'Daniel' (Sera o equivalente para as outras consultas nas partes 2.3 - 2.4 - 2.5)

        // 2.1 O parametro de busca que será utilizado nas proximas consultas
        String nome = "Daniel";

        // ================================================================================================================

        // 2.2 - Utilizando o método find do entityManager
        // Trazendo somente 1 resultado
        Aluno alunoEntityManager = entityManager.find(Aluno.class, 1);

        // Trazendo uma lista como resultado
        // Nao eh possivel!!! Deve utilizar um dos métodos utilizados abaixos nas partes 2.3 - 2.4 - 2.5

        // Resultados das consultas acima
        System.out.println("Consulta alunoEntityManager: " + alunoEntityManager);

        // ================================================================================================================

        // 2.3 - SQL nativo

//        // Trazendo somente 1 resultado
//        String sql = "SELECT * FROM Aluno WHERE nome = :nome ";
//        Aluno alunoSQL = (Aluno) entityManager
//                .createNativeQuery(sql, Aluno.class)
//                .setParameter("nome", nome)
//                .getSingleResult();
//
//        // Trazendo uma lista como resultado
//        String sqlList = "SELECT * FROM Aluno";
//        List<Aluno> alunoSQLList = entityManager
//                .createNativeQuery(sqlList, Aluno.class)
//                .getResultList();
//
//        // Resultados das consultas acima
//        System.out.println("Consulta alunoSQL: " + alunoSQL);
//        alunoSQLList.forEach(Aluno -> System.out.println("Consulta alunoSQLList: " + Aluno));

        // ================================================================================================================

        // 2.4 - JPQL

//        // Trazendo somente 1 resultado
//        String jpql = "select a from Aluno a where a.nome = :nome";
```

```
//        Aluno alunoJPQL = entityManager
//                .createQuery(jpql, Aluno.class)
//                .setParameter("nome", nome)
//                .getSingleResult();
//
//        // Trazendo uma lista como resultado
//        String jpqlList = "select a from Aluno a where a.estado = :estado";
//        List<Aluno> alunoJPQLList = entityManager
//                .createQuery(jpqlList, Aluno.class)
//                .setParameter("estado", estadoParaAdicionar)
//                .getResultList();
//
//        // Resultados das consultas acima
//        System.out.println("Consulta alunoJPQL: " + alunoJPQL);
//        alunoJPQLList.forEach(Aluno -> System.out.println("Consulta alunoJPQLList: " + Aluno));

        // ============================================================================================================

        // 2.5 - JPA Criteria API + JPA Metamodel

//        // Trazendo somente 1 resultado
//        CriteriaQuery<Aluno> criteriaQuery = entityManager.getCriteriaBuilder().createQuery(Aluno.class);
//        Root<Aluno> alunoRoot = criteriaQuery.from(Aluno.class);
//        CriteriaBuilder.In<String> inClause = entityManager.getCriteriaBuilder().in(alunoRoot.get(Aluno_.nome));
//        inClause.value(nome);
//        criteriaQuery.select(alunoRoot).where(inClause);
//        Aluno alunoAPICriteria = entityManager.createQuery(criteriaQuery).getSingleResult();
//
//        // Trazendo uma lista como resultado
//        CriteriaQuery<Aluno> criteriaQueryList = entityManager.getCriteriaBuilder().createQuery(Aluno.class);
//        Root<Aluno> alunoRootList = criteriaQueryList.from(Aluno.class);
//        List<Aluno> alunoAPICriteriaList = entityManager.createQuery(criteriaQueryList).getResultList();
//
//        // Resultados das consultas acima
//        System.out.println("Consulta alunoAPICriteria: " + alunoAPICriteria);
//        alunoAPICriteriaList.forEach(Aluno -> System.out.println("Consulta alunoAPICriteriaList: " + Aluno));

    }
}
```