Instalando o MySQL

Servidor de banco de dados.

<https://dev.mysql.com/downloads/windows/installer/8.0.html>

* Inicializando o serviço:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Serviço em execução:**

(MYSQL80)

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Abrindo a ferramenta de administração da base de dados:**

MYSQL WORKBENCH

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Criando uma base de dados:**

(LINGUAGEM SQL)

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

**Visualizando as bases de dados:**

SCHEMAS

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Criando uma tabela para cadastro de clientes, composta de:

|  |  |  |  |
| --- | --- | --- | --- |
| **Campo** | **Tipo de dado** | **Obrigatório** | **Extras** |
| IDCLIENTE | Numérico inteiro | Sim | Chave primária |
| NOME | Texto | Sim | Máximo 150 carac. |
| EMAIL | Texto | Sim | Máximo 100 carac. |
| CPF | Texto | Sim | Máximo 15 carac. |

**Diagrama de Entidade / Relacionamento (DER)**

Modelo do banco de dados

Interface gráfica do usuário

Descrição gerada automaticamente

#criando a base de dados

**CREATE DATABASE AULA05JAVA;**

#acessando a base de dados

**USE AULA05JAVA;**

#criando a tabela de clientes

**CREATE TABLE CLIENTE(**

**IDCLIENTE INTEGER AUTO\_INCREMENT,**

**NOME VARCHAR(150) NOT NULL,**

**EMAIL VARCHAR(100) NOT NULL,**

**CPF VARCHAR(15) NOT NULL,**

**PRIMARY KEY(IDCLIENTE));**

**Tabela criada:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Criando um projeto Java:**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Criando os pacotes do projeto:**

Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente

/entities/Cliente.java

Classe de entidade similar a tabela criada no banco de dados.

Para criarmos este modelo de dados iremos seguir o padrão   
**JAVABEAN**, ou seja, a classe irá conter:

* Atributos privados
* Construtor sem argumentos
* Construtor com argumentos
* Métodos de encapsulamento
  + Setters
  + Getters
* Sobrescrita dos métodos da classe Object
  + toString
  + equals
  + hashCode

**package** entities;

**public** **class** Cliente {

// atributos privados

**private** Integer idCliente;

**private** String nome;

**private** String email;

**private** String cpf;

// construtor sem argumentos

**public** Cliente() {

// **TODO** Auto-generated constructor stub

}

// construtor com entrada de argumentos (sobrecarga de método)

**public** Cliente(Integer idCliente, String nome,

String email, String cpf) {

**super**();

**this**.idCliente = idCliente;

**this**.nome = nome;

**this**.email = email;

**this**.cpf = cpf;

}

// métodos setters (atribuição) e getters (saída)

**public** Integer getIdCliente() {

**return** idCliente;

}

**public** **void** setIdCliente(Integer idCliente) {

**this**.idCliente = idCliente;

}

**public** String getNome() {

**return** nome;

}

**public** **void** setNome(String nome) {

**this**.nome = nome;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getCpf() {

**return** cpf;

}

**public** **void** setCpf(String cpf) {

**this**.cpf = cpf;

}

// sobrescrita do método toString da classe Object

@Override

**public** String toString() {

**return** "Cliente [idCliente=" + idCliente

+ ", nome=" + nome + ", email="

+ email + ", cpf=" + cpf + "]";

}

}

**Próximo passo:**

Criar uma classe no pacote **/inputs** composta de métodos **estáticos** para fazer a leitura de cada campo da classe Cliente.

/inputs/ClienteInput.java

**package** inputs;

**import** java.util.Scanner;

**public** **class** ClienteInput {

// método para ler o id do cliente

**public static** Integer lerIdCliente() {

@SuppressWarnings("resource")

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Informe o id do cliente......: ");

**return** Integer.*parseInt*(scanner.nextLine());

}

// método para ler o nome do cliente

**public static** String lerNome() {

@SuppressWarnings("resource")

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Informe o nome do cliente....: ");

**return** scanner.nextLine();

}

// método para ler o email do cliente

**public static** String lerEmail() {

@SuppressWarnings("resource")

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Informe o email do cliente...: ");

**return** scanner.nextLine();

}

// método para ler o cpf do cliente

**public static** String lerCpf() {

@SuppressWarnings("resource")

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Informe o cpf do cliente.....: ");

**return** scanner.nextLine();

}

}

Criando a classe para executarmos o projeto:

/principal/Program.java

**package** principal;

**import** entities.Cliente;

**import** inputs.ClienteInput;

**public** **class** Program {

**public** **static** **void** main(String[] args) {

**try** {

System.***out***.println("\nCADASTRO DE CLIENTE:\n");

Cliente cliente = **new** Cliente();

cliente.setNome(ClienteInput.*lerNome*());

cliente.setEmail(ClienteInput.*lerEmail*());

cliente.setCpf(ClienteInput.*lerCpf*());

//imprimindo

System.***out***.println(cliente.toString());

}

**catch** (Exception e) {

// imprimir mensagem de erro

System.***out***.println("\nErro: " + e.getMessage());

}

}

}

**Testando:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

CADASTRO DE CLIENTE:

Informe o nome do cliente....: Sergio Mendes

Informe o email do cliente...: sergio.coti@gmail.com

Informe o cpf do cliente.....: 123.456.789-00

Cliente [idCliente=null, nome=Sergio Mendes, email=sergio.coti@gmail.com, cpf=123.456.789-00]

**JDBC** – JAVA DATABASE CONNECTIVITY

Biblioteca Java responsável por conectar as aplicações

desenvolvidas em Java em uma base de dados.

Para tipo de banco de dados que pretendemos utilizar nós precisamos incluir no projeto o **DRIVER JDBC** correspondente do banco de dados que estamos trabalhando. Exemplo: MYSQL, ORACLE, POSTGRE etc.

Precisamos, no nosso caso, baixar o **DRIVER JDBC para o MYSQL**.

Como estamos ainda trabalhando em POO (programação orientada a objetos), iremos baixar o DRIVER manualmente e depois adicioná-lo também de forma manual no projeto.

Quando estivermos desenvolvendo aplicações web nas próximas aulas iremos utilizar um framework chamado **MAVEN** que irá instalar essas bibliotecas de forma automatizada.

<https://mvnrepository.com/>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Baixando a versão **8.0.27**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

<https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.27>

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Adicionando referência no projeto para esta biblioteca:**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

ADD EXTERNAL JARS

Adicionando bibliotecas externas.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Selecione o conector do MySQL:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente

Para que possamos conectar na base de dados, iremos criar uma classe Java somente para esta finalidade (Abrir uma conexão com o MySQL)

Esta classe será uma “Fábrica de conexões” pois ela terá um método que irá retornar conexões com uma base de dados do MySQL, neste caso o banco criado com o nome: **AULA05JAVA**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Criando a classe:**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** factories;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**public** **class** ConnectionFactory {

// atributos de valores já definidos (constantes)

**private** **static** **final** String ***HOST*** = "jdbc:mysql://localhost:3306/

AULA05JAVA?useTimezone=true

&serverTimezone=UTC&useSSL=false";

**private** **static** **final** String ***USER*** = "root";

**private** **static** **final** String ***PASSWORD*** = "coti";

// método para criar e retornar uma conexão com o banco de dados

**public** **static** Connection getConnection() **throws** Exception {

// criando e retornando uma conexão com o banco de dados

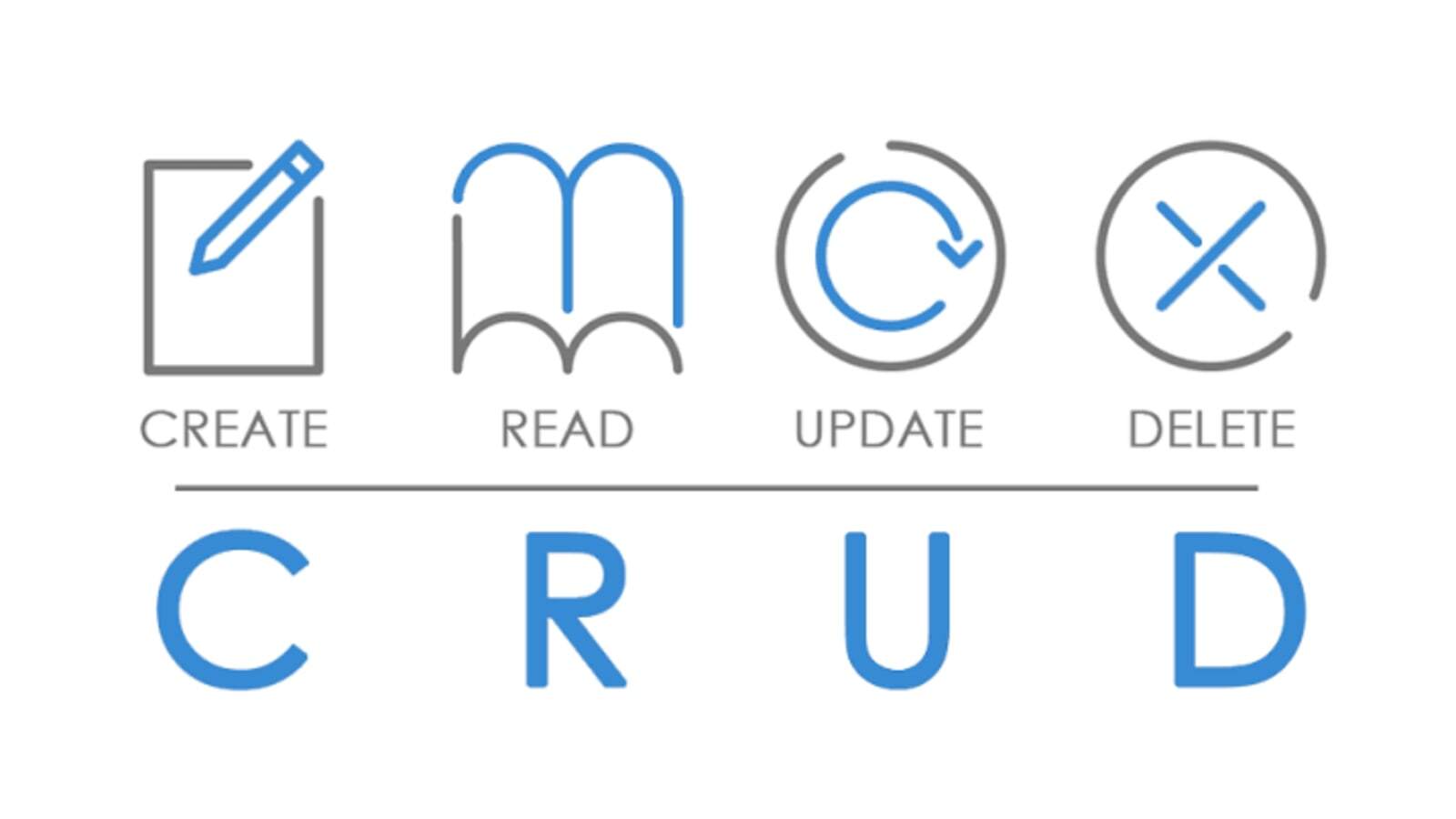
**return** DriverManager.*getConnection*(***HOST***, ***USER***, ***PASSWORD***);

}

}

**Padrão Repository**

Iremos criar classes para persistir as informações de clientes em uma base de dados (GRAVAR, ALTERAR, EXCLUIR e CONSULTAR). Este tipo de rotina é chamada de CRUD (CREATE, READ, UPDATE e DELETE).



Primeiro, iremos criar uma interface para definir quais métodos deverão ser implementados para construirmos o repositório de clientes.

Uma interface é composta de métodos abstratos, que depois alguma classe deverá implementar (fornecer corpo para os métodos).

**Primeiro, iremos criar a interface e a abstração dos métodos**

Depois iremos criar a classe para implementar os métodos.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

/interfaces/IClienteRepository.java

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** interfaces;

**import** java.util.List;

**import** entities.Cliente;

**public interface IClienteRepository {**

**// métodos abstratos (somente assinatura)**

**void create(Cliente cliente) throws Exception;**

**void update(Cliente cliente) throws Exception;**

**void delete(Cliente cliente) throws Exception;**

**List<Cliente> findAll() throws Exception;**

**}**

Próximo passo:

Implementar a interface

Iremos criar uma classe que forneça implementação   
para todos os métodos abstratos da interface.

Interface gráfica do usuário

Descrição gerada automaticamente

/repositories/ClienteRepository.java

Interface gráfica do usuário

Descrição gerada automaticamente

**package** repositories;

**import** java.util.List;

**import** entities.Cliente;

**import** interfaces.IClienteRepository;

**public** **class** ClienteRepository **implements** IClienteRepository {

@Override

**public** **void** create(Cliente cliente) **throws** Exception {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** update(Cliente cliente) **throws** Exception {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** delete(Cliente cliente) **throws** Exception {

// **TODO** Auto-generated method stub

}

@Override

**public** List<Cliente> findAll() **throws** Exception {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

**java.sql**

Pacote JAVA onde estão a maioria das classes para

manipulação de banco de dados, as principais são:

Connection

Interface para armazenar a conexão aberta com o banco de dados.

PreparedStatement

Utilizado para que possamos executar comandos SQL no banco de dados, tais como INSERT, UPDATE, DELETE e SELECT.

CallableStatement

Utilizado para executar STORED PROCEDURES no banco de dados.

ResultSet

Utilizado para que possamos executar e ler resultados obtidos de consultas (SELECT) feitas no banco de dados.

Implementando o método **create** para   
gravar um cliente na base de dados:

**package** repositories;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.util.List;

**import** entities.Cliente;

**import** factories.ConnectionFactory;

**import** interfaces.IClienteRepository;

**public** **class** ClienteRepository **implements** IClienteRepository {

**@Override**

**public void create(Cliente cliente) throws Exception {**

**//abrindo uma conexão com o banco de dados**

**Connection connection = ConnectionFactory.*getConnection*();**

**//gravar um cliente na base de dados**

**PreparedStatement statement = connection.prepareStatement**

**("INSERT INTO CLIENTE(NOME, CPF, EMAIL) VALUES(?, ?, ?)");**

**statement.setString(1, cliente.getNome());**

**statement.setString(2, cliente.getCpf());**

**statement.setString(3, cliente.getEmail());**

**statement.execute(); //executando o comando**

**statement.close();**

**//fechando a conexão**

**connection.close();**

**}**

@Override

**public** **void** update(Cliente cliente) **throws** Exception {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** delete(Cliente cliente) **throws** Exception {

// **TODO** Auto-generated method stub

}

@Override

**public** List<Cliente> findAll() **throws** Exception {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

**Testando o cadastro do cliente na classe Program.java**

package principal;

import entities.Cliente;

import inputs.ClienteInput;

import repositories.ClienteRepository;

public class Program {

public static void main(String[] args) {

try {

System.out.println("\nCADASTRO DE CLIENTE:\n");

Cliente cliente = new Cliente();

cliente.setNome(ClienteInput.lerNome());

cliente.setEmail(ClienteInput.lerEmail());

cliente.setCpf(ClienteInput.lerCpf());

**//cadastrando no banco de dados**

**ClienteRepository clienteRepository**

**= new ClienteRepository();**

**clienteRepository.create(cliente);**

**System.out.println**

**("\nCLIENTE CADASTRADO COM SUCESSO!");**

}

catch (Exception e) {

// imprimir mensagem de erro

System.out.println("\nErro: " + e.getMessage());

}

}

}

**Testando:**

Texto, Aplicativo

Descrição gerada automaticamente

CADASTRO DE CLIENTE:

Informe o nome do cliente....: Sergio da Silva Mendes

Informe o email do cliente...: sergio.coti@gmail.com

Informe o cpf do cliente.....: 123.456.789-00

CLIENTE CADASTRADO COM SUCESSO!

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Desenvolvendo os demais métodos de UPDATE e DELETE:

/repositories/ClienteRepository.java

**package** repositories;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.util.List;

**import** entities.Cliente;

**import** factories.ConnectionFactory;

**import** interfaces.IClienteRepository;

**public** **class** ClienteRepository **implements** IClienteRepository {

@Override

**public** **void** create(Cliente cliente) **throws** Exception {

//abrindo uma conexão com o banco de dados

Connection connection = ConnectionFactory.*getConnection*();

//gravar um cliente na base de dados

PreparedStatement statement = connection.prepareStatement

("INSERT INTO CLIENTE(NOME, CPF, EMAIL) VALUES(?, ?, ?)");

statement.setString(1, cliente.getNome());

statement.setString(2, cliente.getCpf());

statement.setString(3, cliente.getEmail());

statement.execute(); //executando o comando

statement.close();

//fechando a conexão

connection.close();

}

**@Override**

**public void update(Cliente cliente) throws Exception {**

**Connection connection = ConnectionFactory.*getConnection*();**

**PreparedStatement statement = connection.prepareStatement**

**("UPDATE CLIENTE SET NOME = ?, EMAIL = ?,**

**CPF = ? WHERE IDCLIENTE = ?");**

**statement.setString(1, cliente.getNome());**

**statement.setString(2, cliente.getEmail());**

**statement.setString(3, cliente.getCpf());**

**statement.setInt(4, cliente.getIdCliente());**

**statement.execute();**

**statement.close();**

**connection.close();**

**}**

**@Override**

**public void delete(Cliente cliente) throws Exception {**

**Connection connection = ConnectionFactory.*getConnection*();**

**PreparedStatement statement = connection.prepareStatement**

**("DELETE FROM CLIENTE WHERE IDCLIENTE = ?");**

**statement.setInt(1, cliente.getIdCliente());**

**statement.execute();**

**statement.close();**

**connection.close();**

**}**

@Override

**public** List<Cliente> findAll() **throws** Exception {

// **TODO** Auto-generated method stub

**return** **null**;

}

}

Continua...