**Novo projeto:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Criando uma classe de entidade:

/entities/Cliente.java

**JavaBeans**

Segundo o padrão JavaBeans, toda classe de entidade (modelo de dados) deverá seguir as seguintes especificações:]

* Atributos privados
* Métodos de encapsulamento
  + Setters
  + Getters
* Método construtor sem argumentos
* Sobrecarga de construtores
  + Construtores com entrada de argumentos
* Sobrescrever os métodos da classe Object

**package** entities;

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

**private** Integer idCliente;

**private** String nome;

**private** String email;

**public** Cliente() {

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

}

**public** Cliente(Integer idCliente, String nome, String email) {

**super**();

**this**.idCliente = idCliente;

**this**.nome = nome;

**this**.email = email;

}

**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;

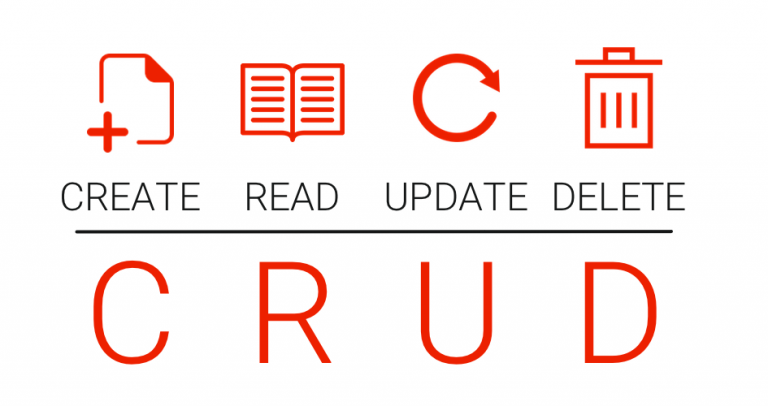
}

}

Desenvolvendo um projeto Java capaz de acessar um banco de dados criado no POSTGRESQL e gravar, alterar, excluir e consultar clientes neste banco de dados.

**CRUD** – CREATE, READ, UPDATE e DELETE

Gravar, ler, alterar e excluir.



Instalando o **PostGreSQL**

<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**PgAdmin**

Ferramenta pata administração dos bancos   
de dados criados no PostGreSQL.

Logotipo, nome da empresa

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Entre com a senha do administrador do banco de dados:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Primeiro passo:

Criar um banco de dados no PostGreSQL para

o desenvolvimento do projeto da aula.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Criando uma tabela no banco**

**de dados para Clientes:**

Linguagem SQL.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Criando um arquivo SQL no projeto Java

/sql/script.sql

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**create** **table** cliente(

idcliente serial **primary** **key**,

nome **varchar**(150) **not** **null**,

email **varchar**(100) **not** **null**

);

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Criando uma classe para que possamos executar as operações no banco de dados relacionadas a tabela de cliente (CRUD).

/repositories/ClienteRepository.java

Interface gráfica do usuário, Tabela, Excel

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**package** repositories;

**import** java.util.List;

**import** entities.Cliente;

**public** **class** ClienteRepository {

// método para gravar um cliente no banco de dados

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

// **TODO**

}

// método para atualizar os dados

// de um cliente no banco de dados

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

// **TODO**

}

// método para excluir um cliente no banco de dados

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

// **TODO**

}

// método para retornar todos os clientes

// cadastrados no banco de dados

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

// **TODO**

**return** **null**;

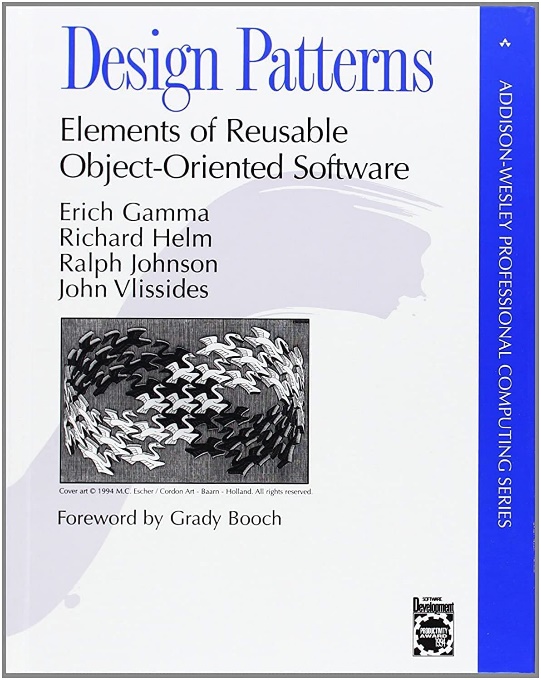
}

}

**Design Patterns Gof**

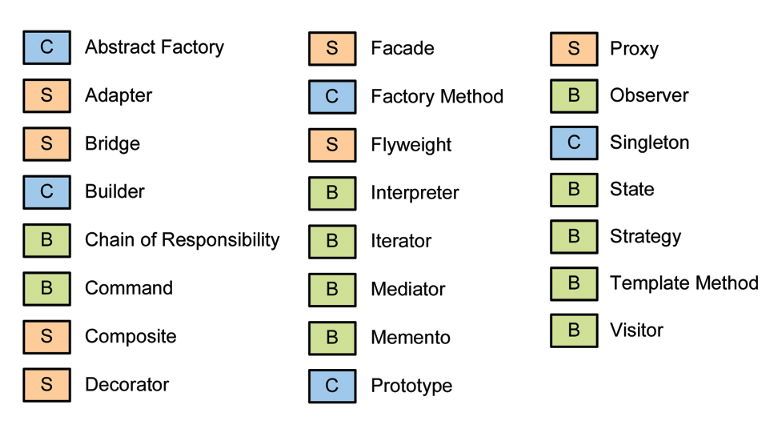
São 23 padrões de projeto propostos por 4 autores chamados

de GoF (Gang of Four) no seguinte livro:



Este livro define 23 padrões de soluções para projetos orientados a objetos, divididos em 3 categorias:

* **Padrões criacionais**
* **Padrões estruturais**
* **Padrões comportamentais**



**Factory Method**

Padrão de projeto (Design Pattern Gof) do tipo Criacional, pois tem a ver como criação e inicialização de objetos. Tem como objetivo prover um único ponto de acesso para criação de um determinado objeto.

Vamos usar este padrão no projeto para criarmos uma classe através do qual possamos “fabricar” conexões com banco de dados.

Exemplo:

Interface gráfica do usuário, Diagrama, Tabela, Excel

Descrição gerada automaticamente

/factories/ConnectionFactory.java

Classe para criar e retornar conexões com o banco de dados.

**java.sql**

Biblioteca que vamos utilizar para conectar a nossa aplicação em qualquer tipo de banco de dados relacional.

**package** factories;

**import** java.sql.Connection;

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

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

**public** Connection createConnection() **throws** Exception {

// **TODO**

**return** **null**;

}

}

**Métodos estáticos**

São métodos declarados em classes Java com a palavra reservada **static**

Tem como característica principal o fato de que para serem executados eles não precisam de uma variável de instancia, ou seja, são chamados diretamente a partir do nome da classe.

Exemplo de método não estático:

**class** A {

**public void print() {**

**System.*out*.println("Hello, A!");**

**}**

}

**class** Program {

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

**A a = new A();**

**a.print();**

}

}

Exemplo de método estático:

**class** A {

**public static void print() {**

**System.*out*.println("Hello, A!");**

**}**

}

**class** Program {

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

**A.*print*();**

}

}

**Desenvolvendo a classe para**

**conexão com bancos de dados:**

**package** factories;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

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

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

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

// parâmetros para conexão com o banco de dados

String driver = "org.postgresql.Driver";

String server = "jdbc:postgresql:

//localhost:5432/bd\_projeto05";

String user = "postgres";

String password = "coti";

// Carregando o driver JDBC para conexão com o PostGreSQL

Class.*forName*(driver);

// fazendo a conexão com o banco de dados, utilizando

// o caminho do banco, usuário e senha de administrador

**return** DriverManager.*getConnection*

(server, user, password);

}

}

**Drivers JDBC (**Java Database Connectivity**)**

São bibliotecas desenvolvidas para Java que permitem conectar os projetos com diversos tipos de bancos de dados. Para cada tipo de banco de dados existe um driver JDBC necessário para que possamos estabelecer conexão com o respectivo SGBD.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Baixando o Driver JDBC para conexão com o PostGreSQL:**

<https://mvnrepository.com/>

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

<https://mvnrepository.com/search?q=postgresql>

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Em Java, as bibliotecas são arquivos de extensão **.jar**

<https://mvnrepository.com/artifact/org.postgresql/postgresql>

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

<https://mvnrepository.com/artifact/org.postgresql/postgresql/42.5.1>

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Adicionando esta biblioteca no projeto:**

BUILD PATH / CONFIGURE BUILD PATH

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

LIBRARIES / ADD EXTERNAL JARS

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente

Desenvolvendo a classe Repository:

/Repositories/ClienteRepository.java

**package** repositories;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.util.List;

**import** entities.Cliente;

**import** factories.ConnectionFactory;

**public** **class** ClienteRepository {

// método para gravar um cliente no banco de dados

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

//abrindo conexão com o banco de dados

Connection connection = ConnectionFactory

.*createConnection*();

//executando um comando SQL no banco

//de dados para cadastrar o cliente

PreparedStatement preparedStatement

= connection.prepareStatement

("insert into cliente(nome, email) values(?, ?)");

//passando os parametros do comando SQL

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

preparedStatement.setString(2, cliente.getEmail());

//executar o comando SQL e fechar

//a conexão com o banco de dados

preparedStatement.execute();

connection.close();

}

// método para atualizar os dados

//de um cliente no banco de dados

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

// **TODO**

}

// método para excluir um cliente no banco de dados

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

// **TODO**

}

// método para retornar todos os clientes

//cadastrados no banco de dados

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

// **TODO**

**return** **null**;

}

}

Criando uma camada de controle para realizar

o fluxo de cadastro de cliente:

/controllers/ClienteController.java

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**package** controllers;

**import** java.util.Scanner;

**import** entities.Cliente;

**import** repositories.ClienteRepository;

**public** **class** ClienteController {

// método para executar o fluxo de cadastro

// de um cliente no banco de dados

**public** **void** cadastrarCliente() {

**try** {

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

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

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

System.***out***.print("NOME DO CLIENTE....: ");

cliente.setNome(scanner.nextLine());

System.***out***.print("EMAIL DO CLIENTE...: ");

cliente.setEmail(scanner.nextLine());

ClienteRepository clienteRepository

= **new** ClienteRepository();

clienteRepository.create(cliente);

System.***out***.println

("\nCLIENTE CADASTRADO COM SUCESSO!");

}

**catch**(Exception e) {

System.***out***.println

("\nFALHA AO CADASTRAR O CLIENTE");

e.printStackTrace();

}

}

}

**Criando a classe de inicialização do projeto:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**package** principal;

**import** controllers.ClienteController;

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

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

ClienteController clienteController

= **new** ClienteController();

clienteController.cadastrarCliente();

}

}

**Executando:**

RUN AS / JAVA APPLICATION

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Saída do programa:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**CADASTRO DE CLIENTES:**

**NOME DO CLIENTE....: Sergio Mendes**

**EMAIL DO CLIENTE...: sergio.coti@gmail.com**

**CLIENTE CADASTRADO COM SUCESSO!**

**Visualizando no banco de dados:**

create table cliente(

idcliente serial primary key,

nome varchar(150) not null,

email varchar(100) not null

);

**select \* from cliente;**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

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

Descrição gerada automaticamente