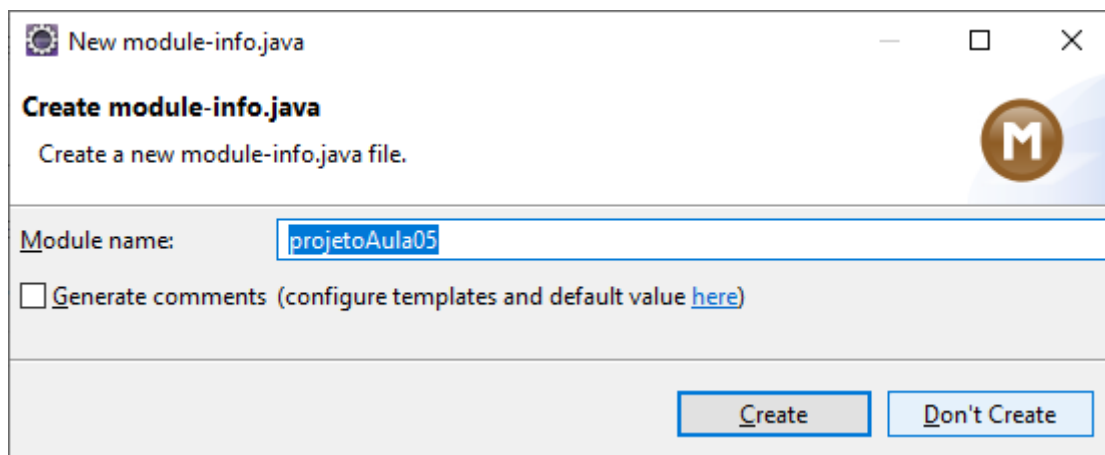
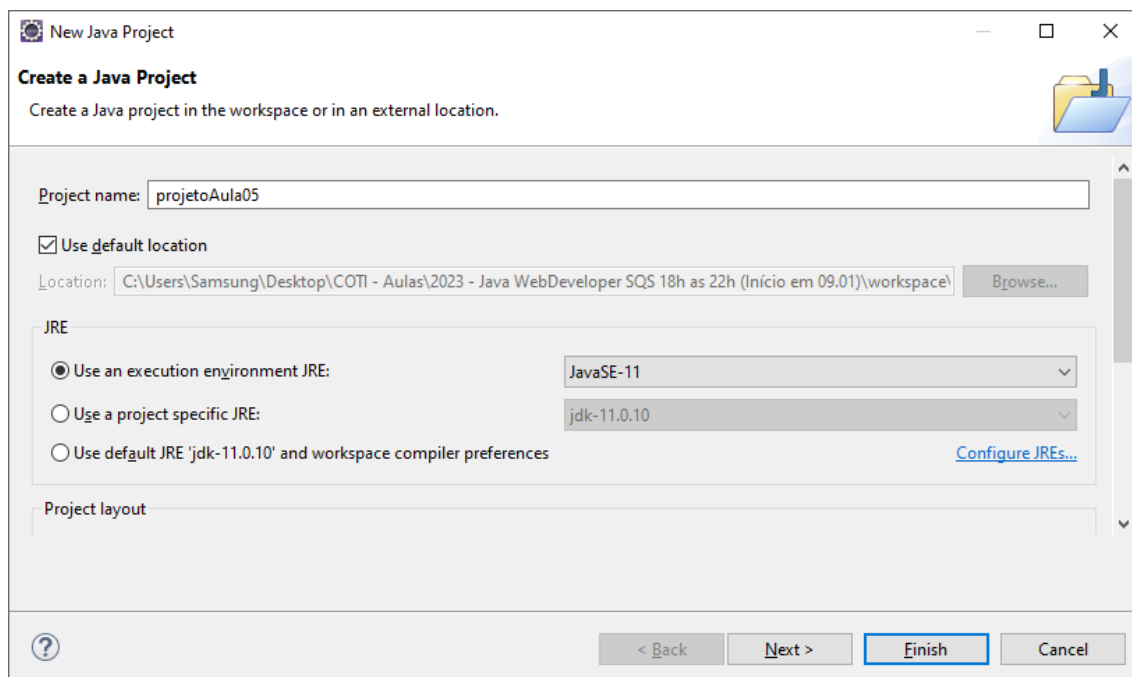
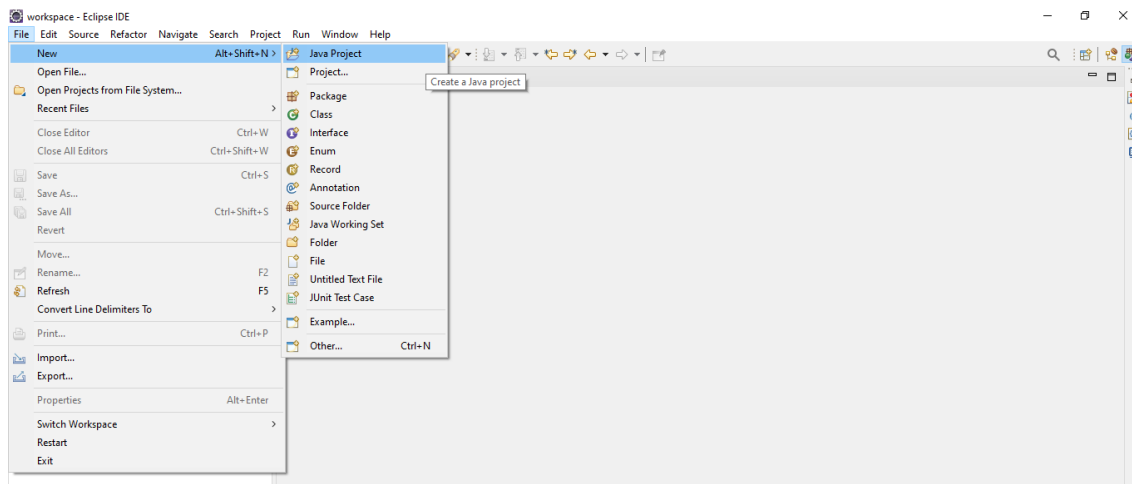


Novo projeto:



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.

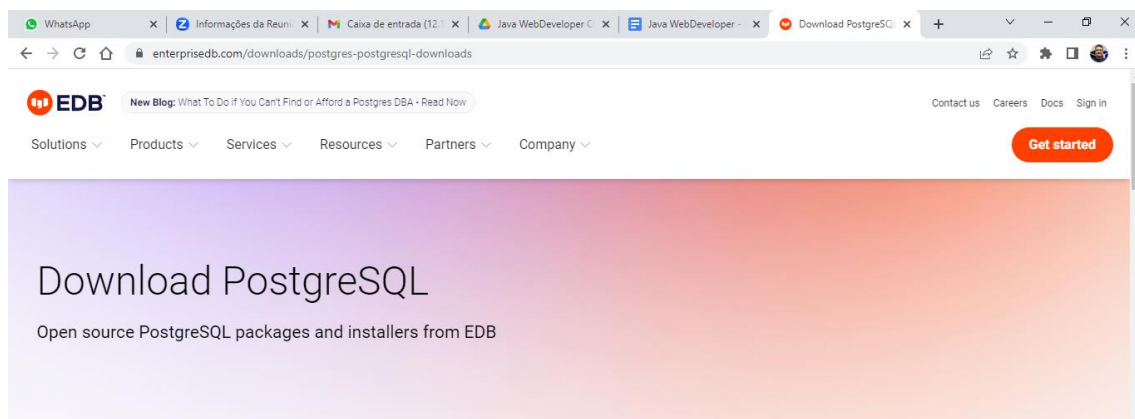
CRUD – CREATE, READ, UPDATE e DELETE

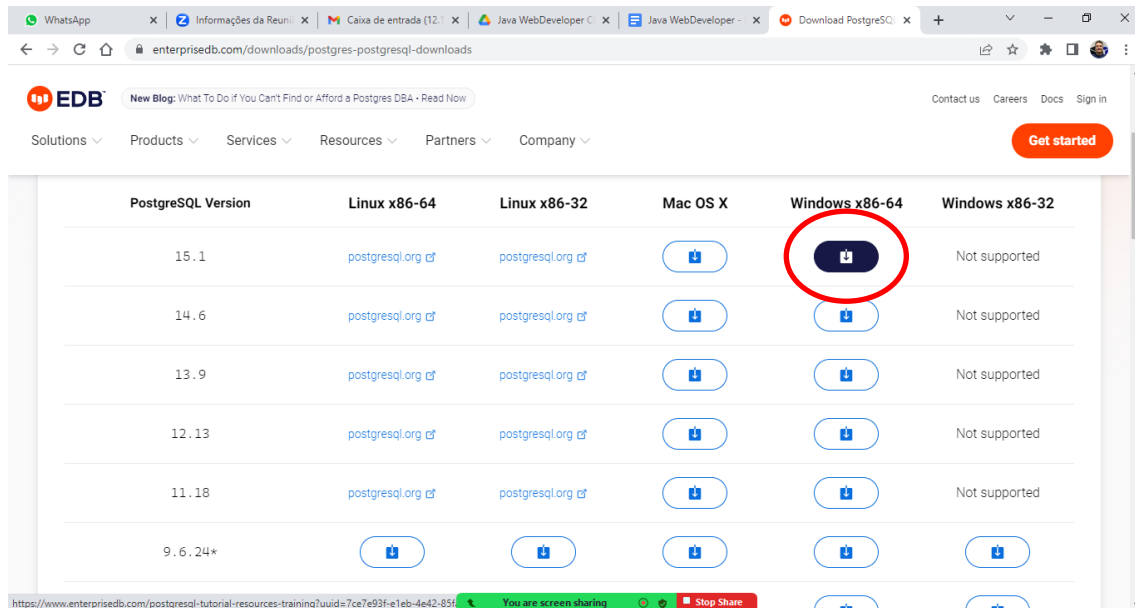
Gravar, ler, alterar e excluir.



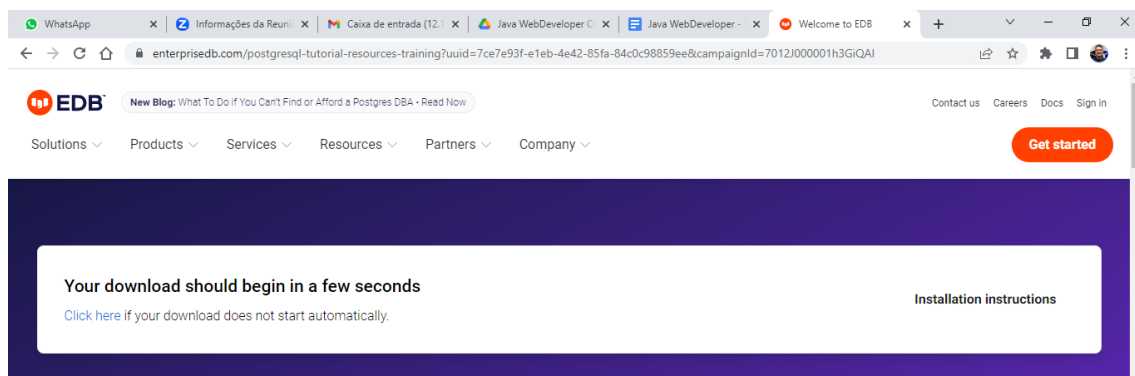
Instalando o **PostgreSQL**

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





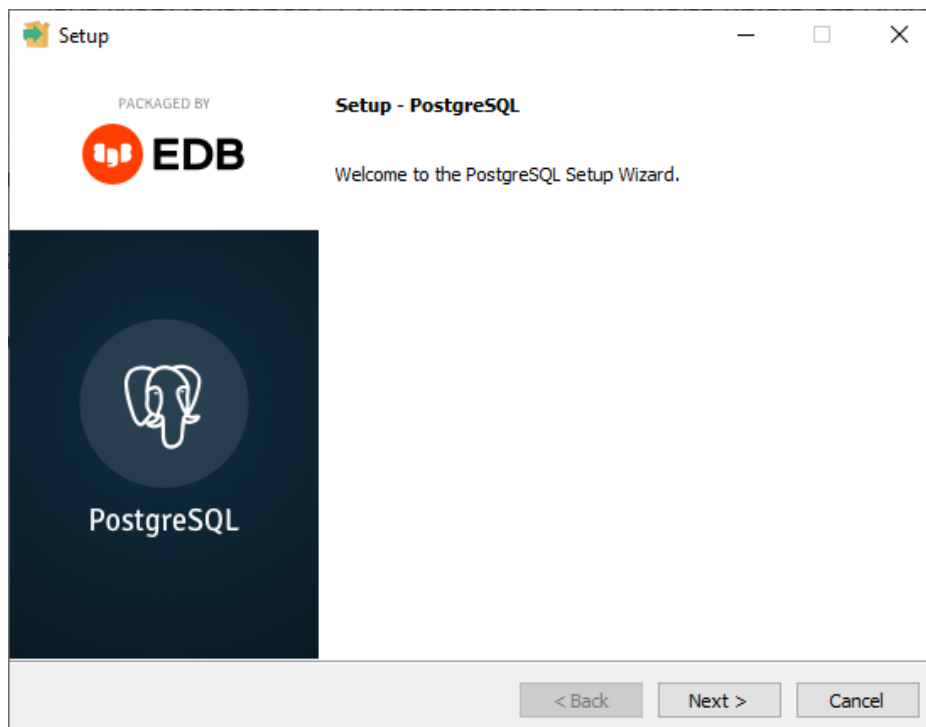
PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
15.1	postgresql.org	postgresql.org			Not supported
14.6	postgresql.org	postgresql.org			Not supported
13.9	postgresql.org	postgresql.org			Not supported
12.13	postgresql.org	postgresql.org			Not supported
11.18	postgresql.org	postgresql.org			Not supported
9.6.24*					



Your download should begin in a few seconds

[Click here if your download does not start automatically.](#)

[Installation instructions](#)




Setup

PACKAGED BY

EDB

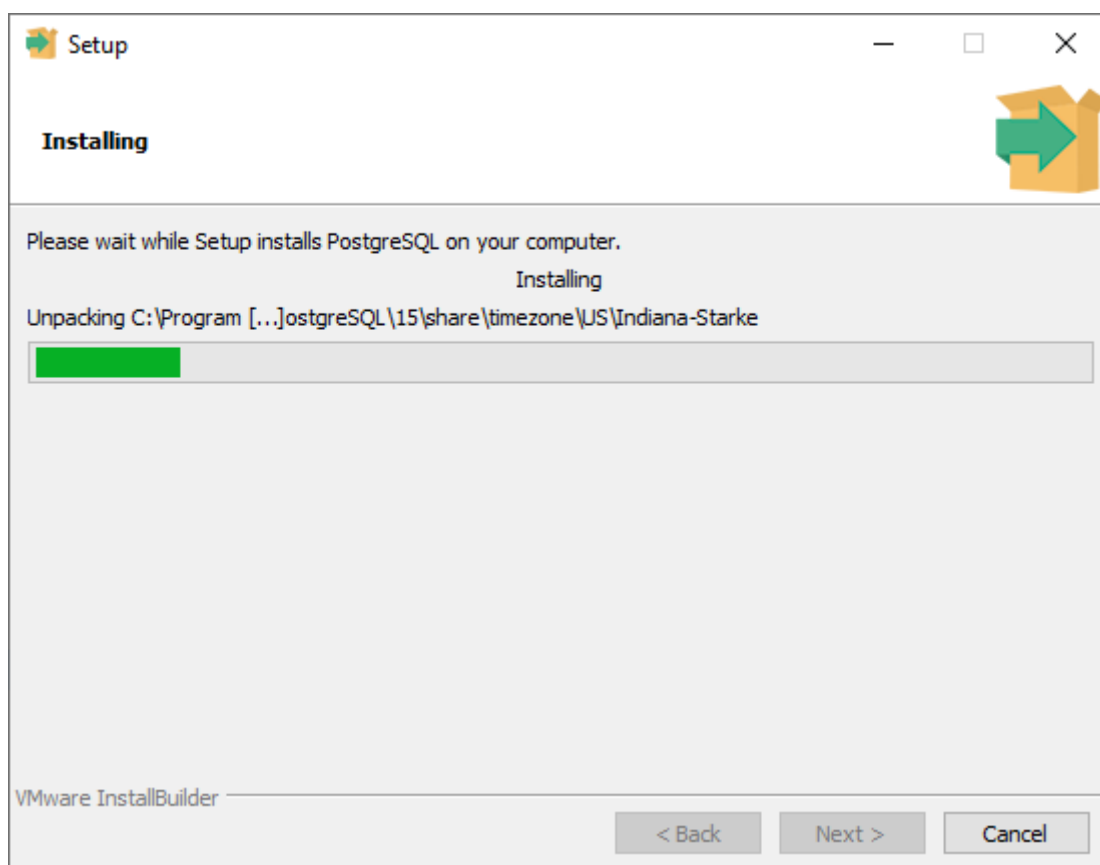
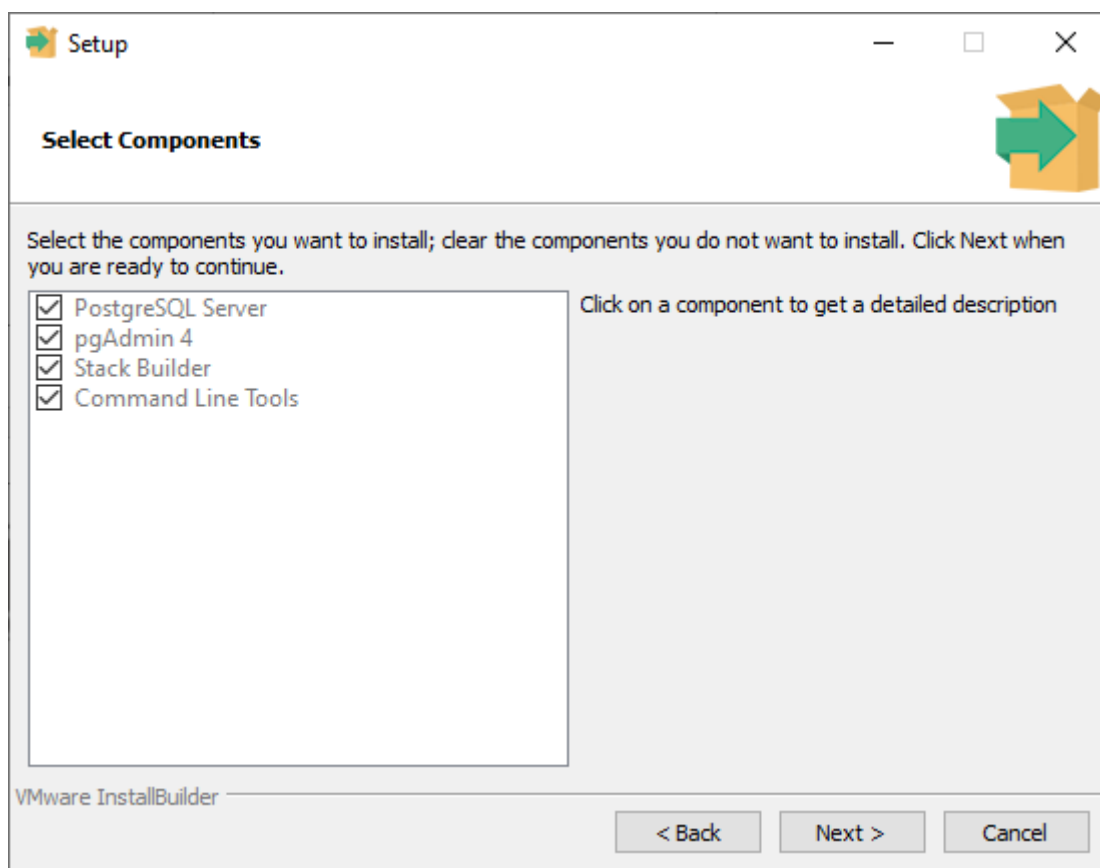
Setup - PostgreSQL

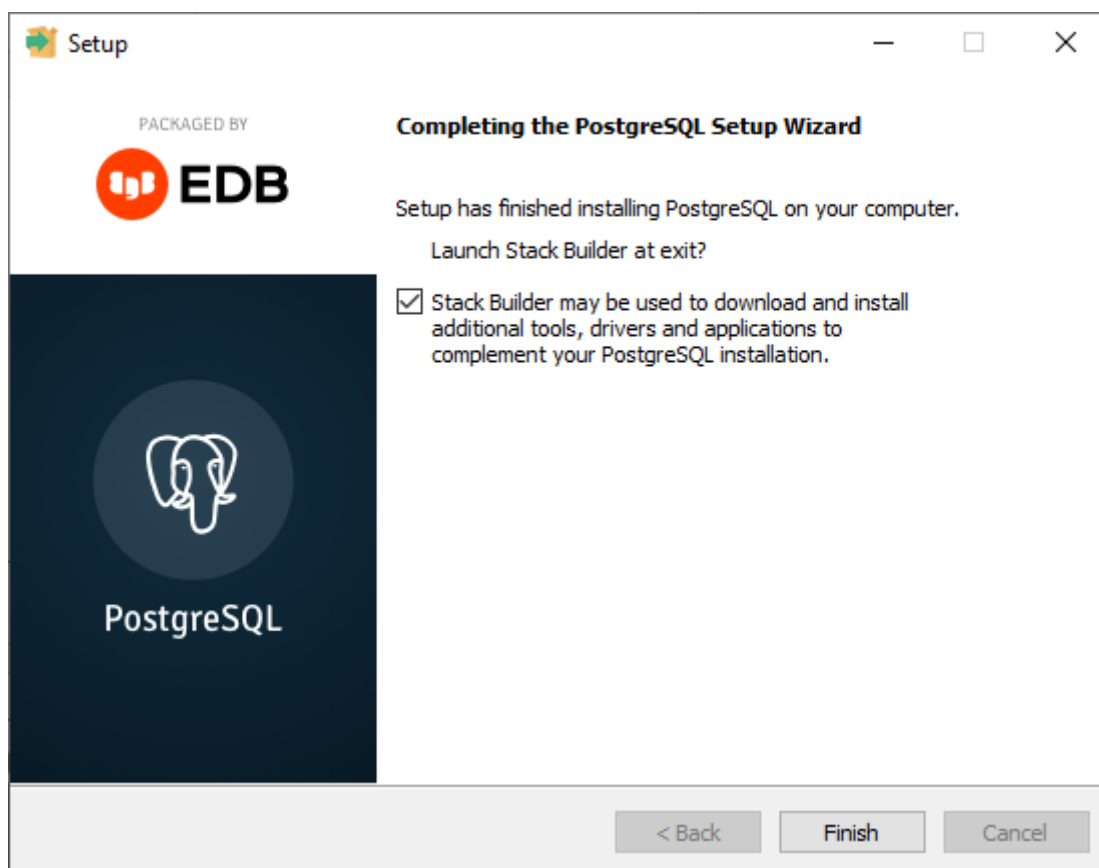
Welcome to the PostgreSQL Setup Wizard.



PostgreSQL

< Back Next > Cancel

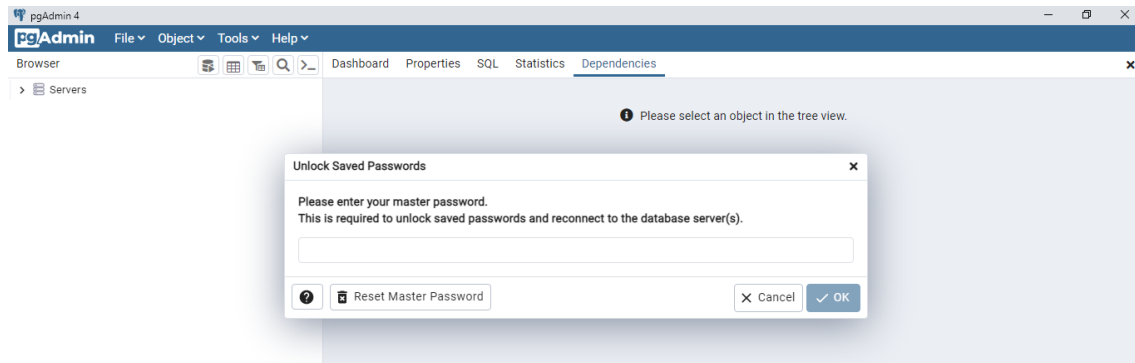




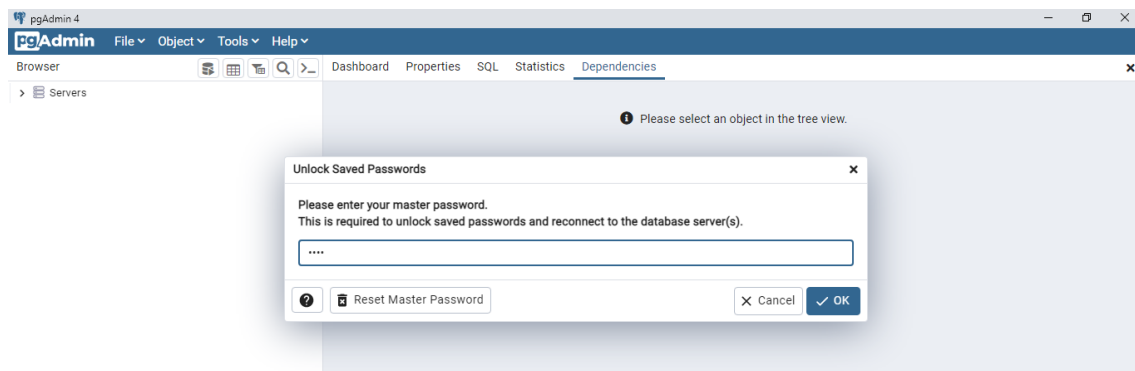
PgAdmin

Ferramenta para administração dos bancos de dados criados no PostgreSQL.



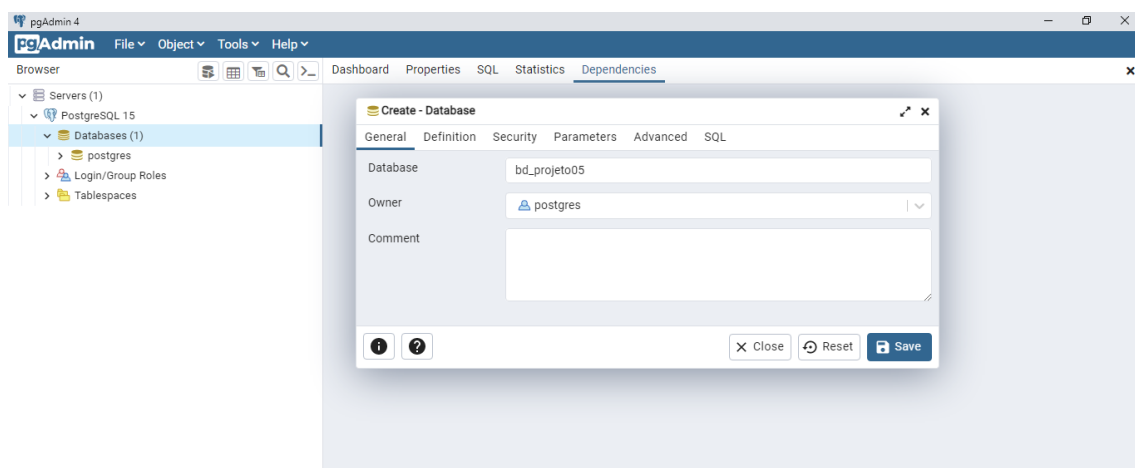
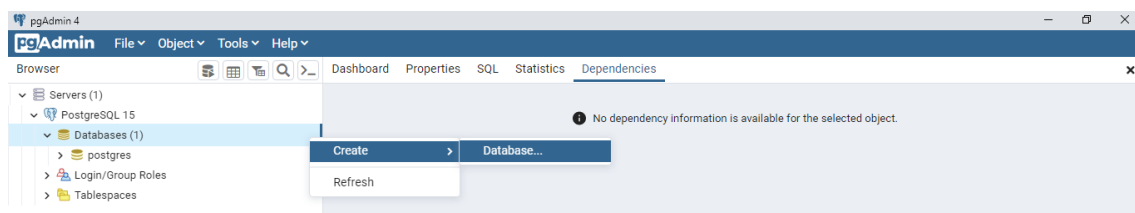


Entre com a senha do administrador do banco de dados:

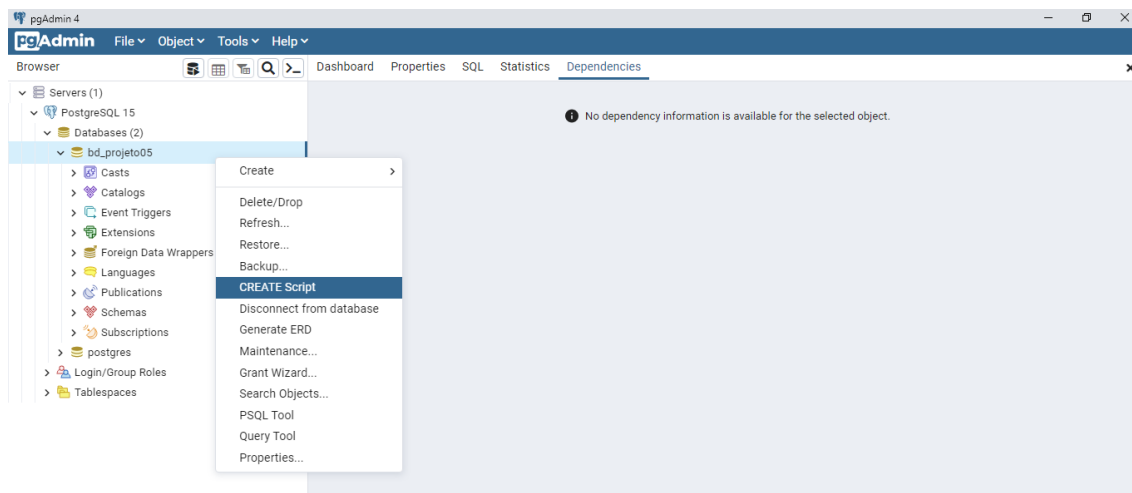


Primeiro passo:

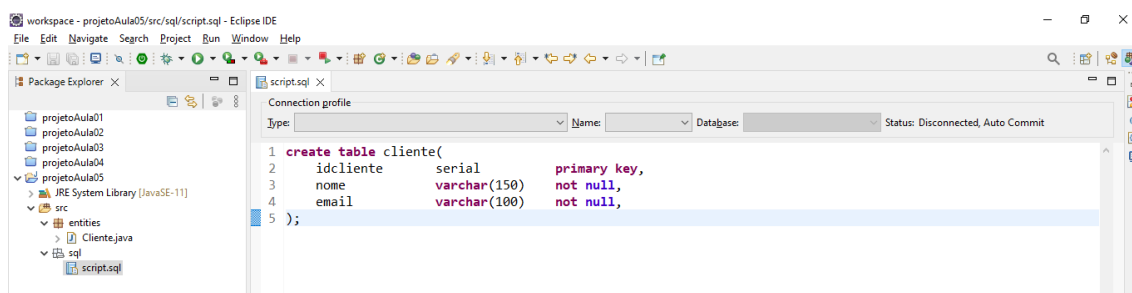
Criar um banco de dados no PostgreSQL para o desenvolvimento do projeto da aula.



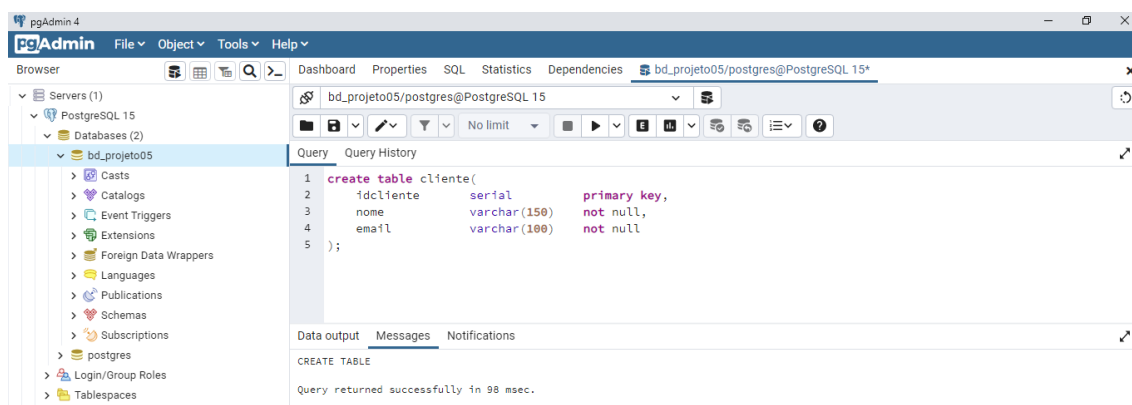
Criando uma tabela no banco de dados para Clientes: Linguagem SQL.



Criando um arquivo SQL no projeto Java /sql/script.sql

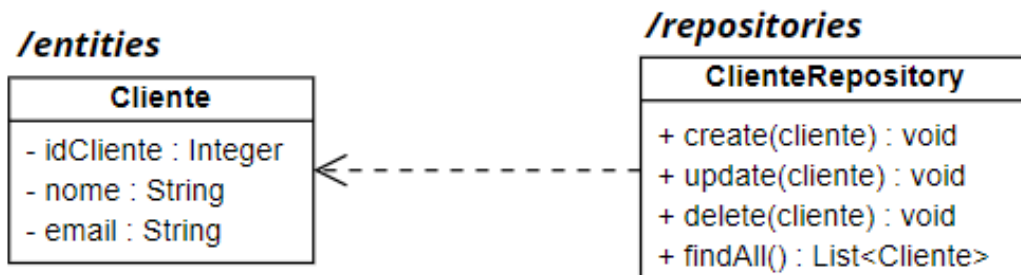
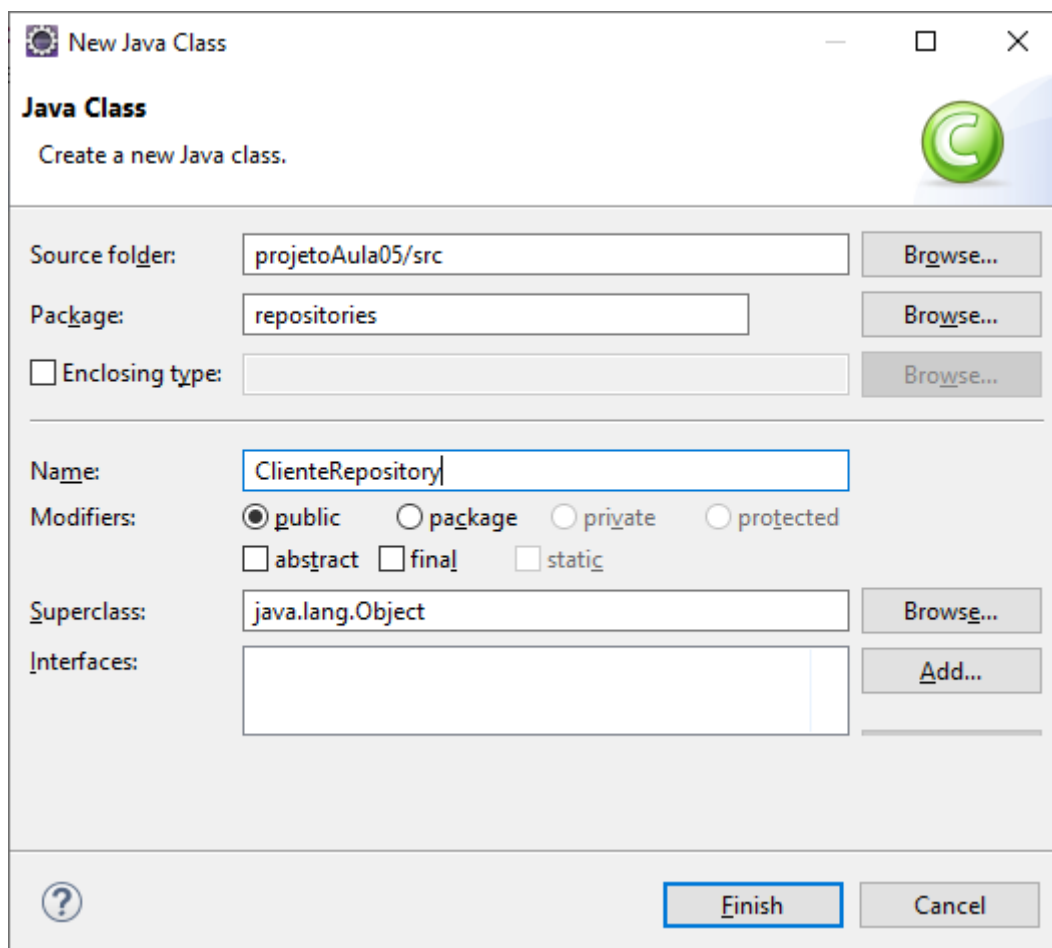


```
create table cliente(
    idcliente      serial          primary key,
    nome           varchar(150)    not null,
    email          varchar(100)    not null
);
```



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

/repositories/**ClienteRepository.java**

New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

```
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
    }

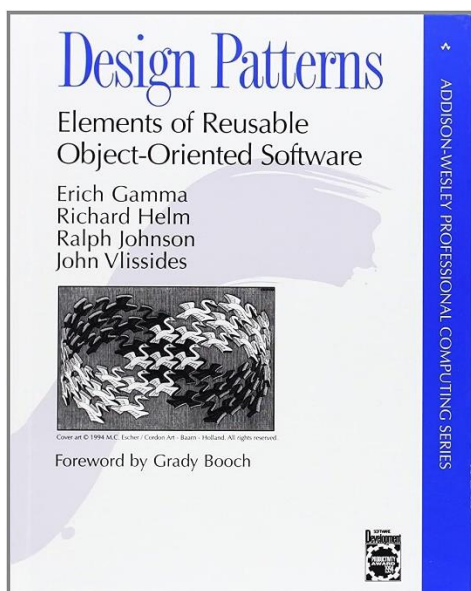
    // método para atualizar os dados
    // de um cliente no banco de dados
    public void update(Cliente cliente) throws Exception {
        // TODO
    }

    // 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**

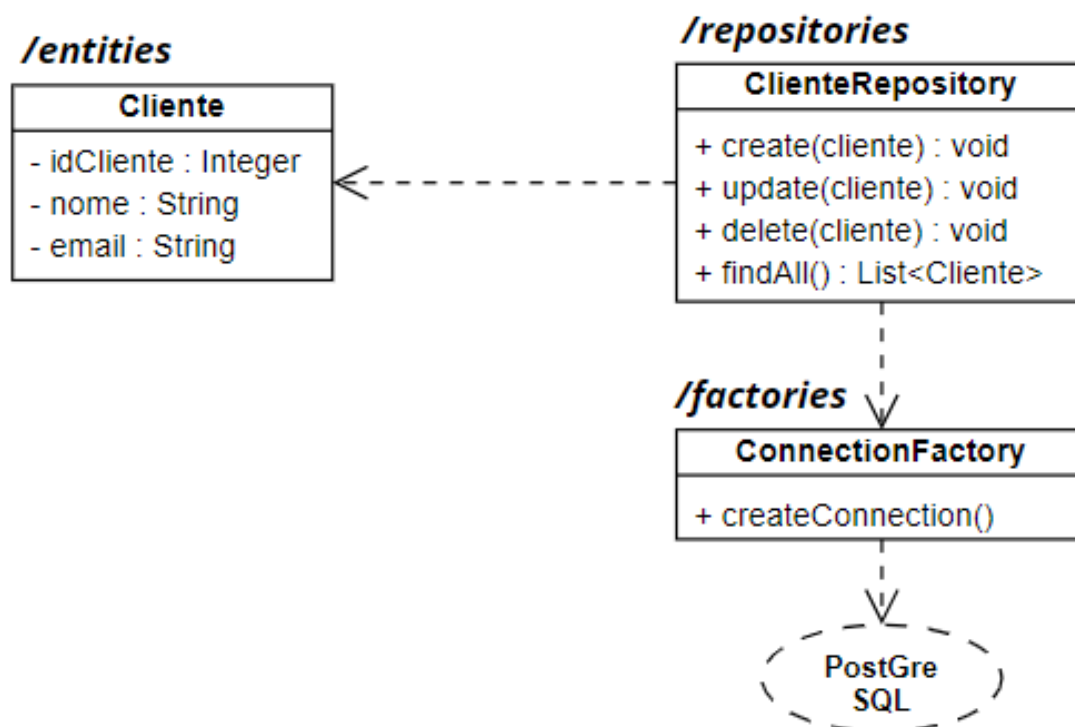
C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

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:



/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();

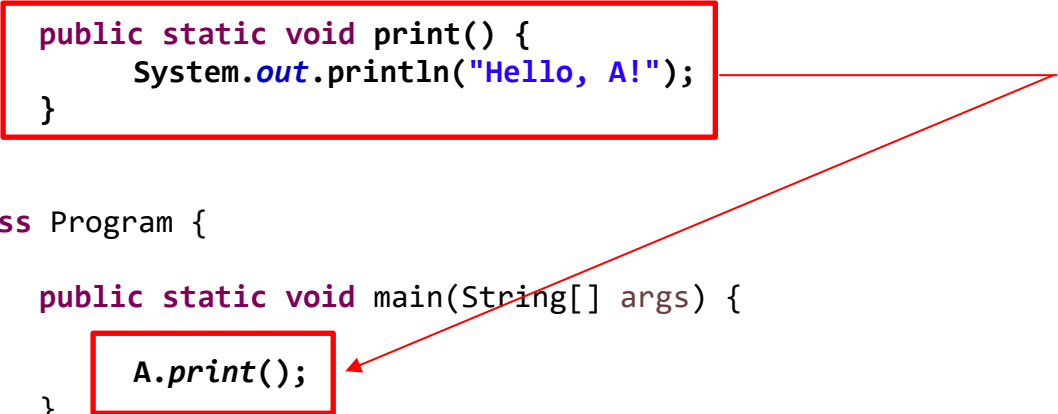
        a.print();

    }

}
```

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();  
    }  
}
```

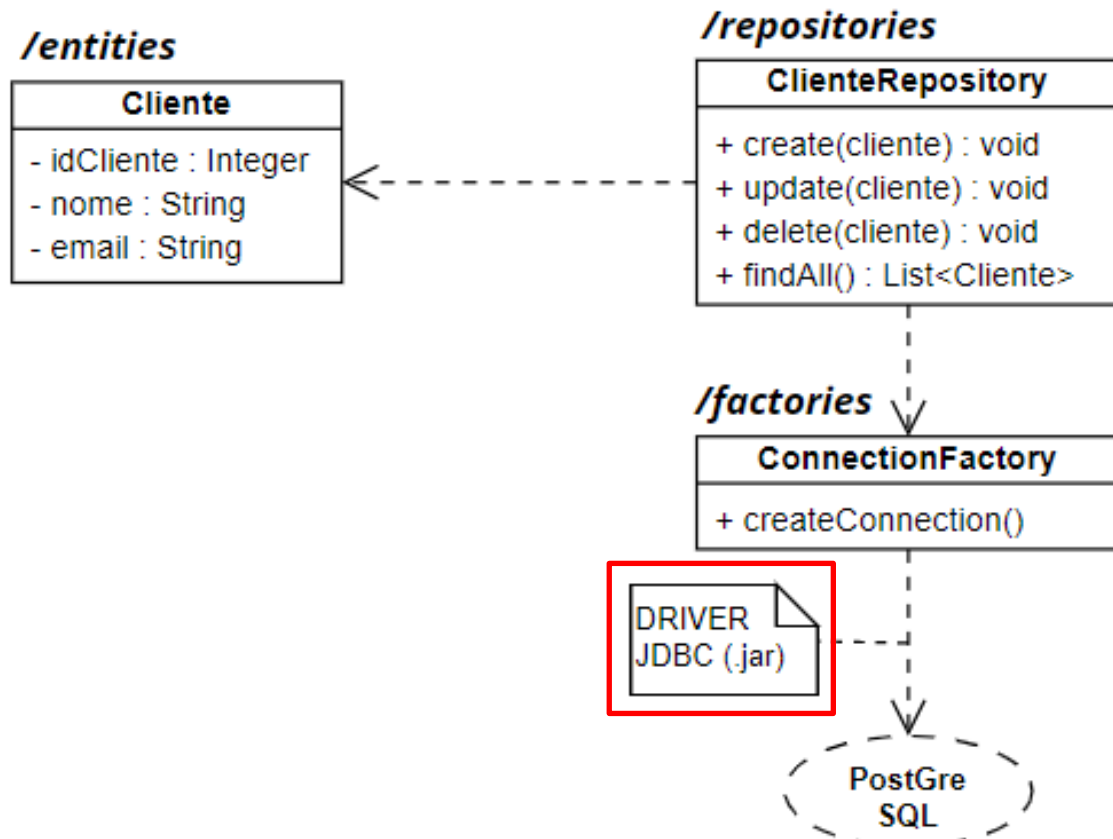


Desenvolvendo a classe para conexão com bancos de dados:

```
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);  
    }  
}
```

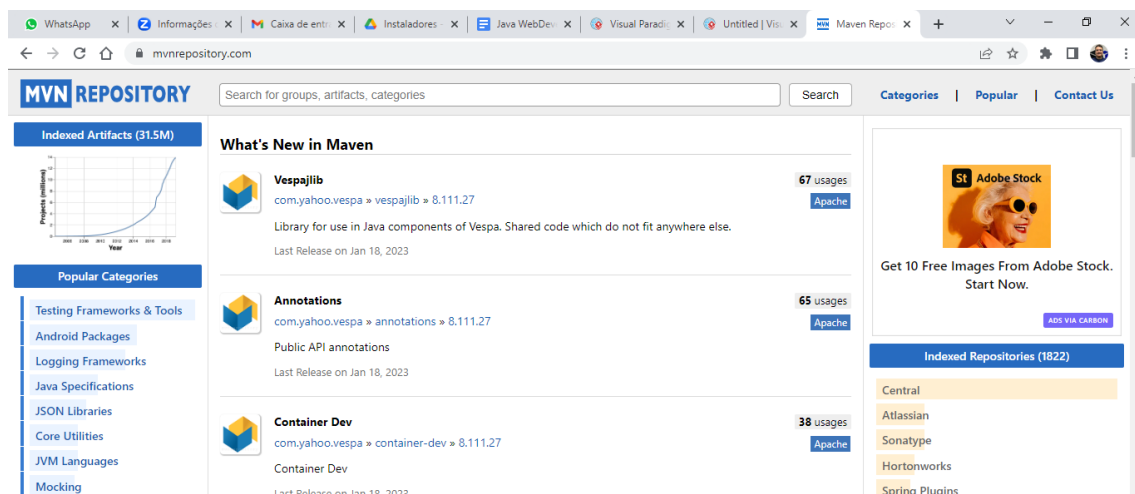
Drivers JDBC (Java Database Connectivity)

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.

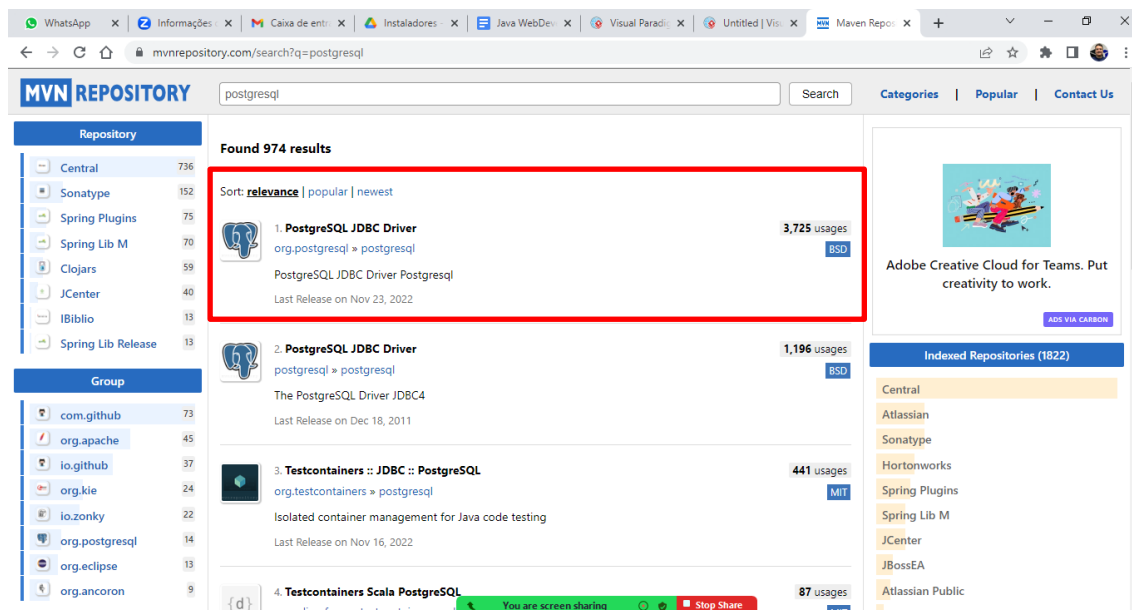


Baixando o Driver JDBC para conexão com o PostGreSQL:

<https://mvnrepository.com/>



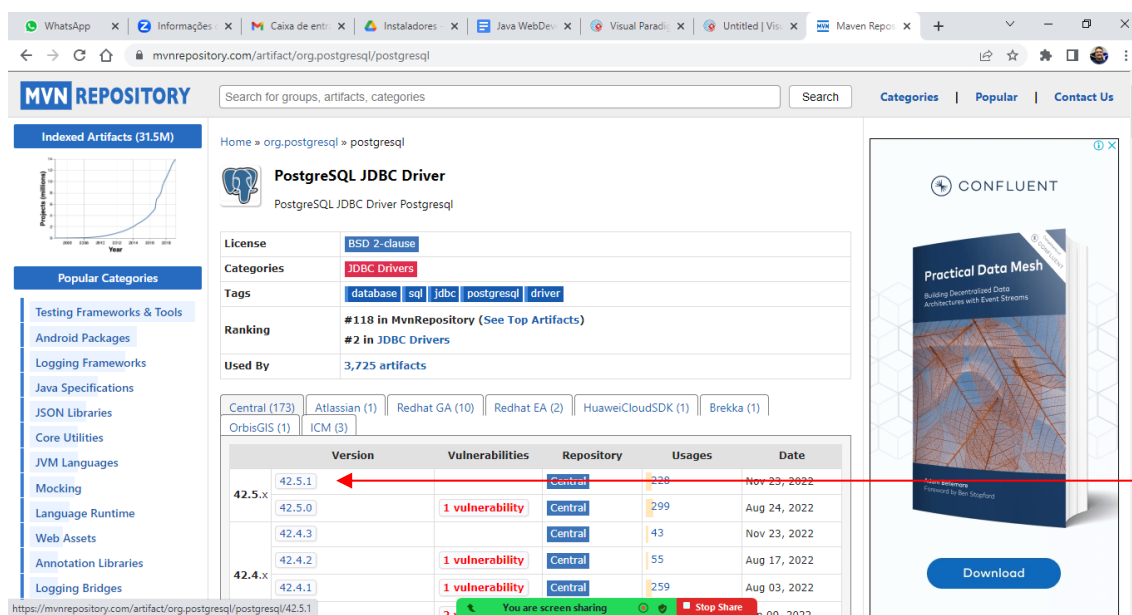
<https://mvnrepository.com/search?q=postgresql>



The screenshot shows the Maven Repository search results for 'postgresql'. The search bar contains 'postgresql' and the results are sorted by 'relevance'. The first result is 'PostgreSQL JDBC Driver' by 'org.postgresql', with 3,725 usages and a last release on Nov 23, 2022. The second result is 'PostgreSQL JDBC Driver' by 'postgresql', with 1,196 usages and a last release on Dec 18, 2011. The third result is 'Testcontainers :: JDBC :: PostgreSQL' by 'org.testcontainers', with 441 usages and a last release on Nov 16, 2022. The fourth result is 'Testcontainers Scala PostgreSQL' by 'com.dimafeng', with 87 usages and a last release on Nov 16, 2022.


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

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

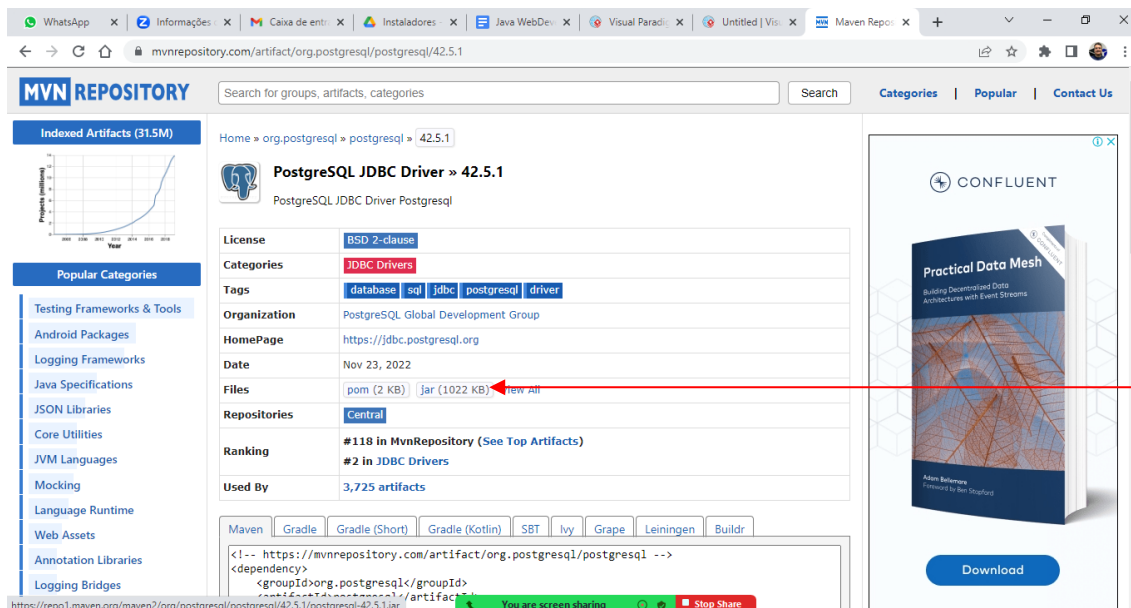


The screenshot shows the Maven Repository artifact page for 'org.postgresql:postgresql'. The page displays the 'PostgreSQL JDBC Driver' with a license of 'BSD 2-clause' and categories of 'JDBC Drivers', 'database', 'sql', 'jdbc', 'postgresql', and 'driver'. The ranking is #118 in Maven Repository and #2 in JDBC Drivers. The page shows a table of versions with columns for Version, Vulnerabilities, Repository, Usages, and Date. The table lists versions 42.5.1, 42.5.0, 42.4.3, 42.4.2, and 42.4.1. Version 42.5.1 is highlighted with a red arrow and has 228 usages. Version 42.5.0 has 299 usages and is marked as '1 vulnerability'. Version 42.4.3 has 43 usages. Version 42.4.2 has 55 usages and is marked as '1 vulnerability'. Version 42.4.1 has 259 usages and is marked as '1 vulnerability'.

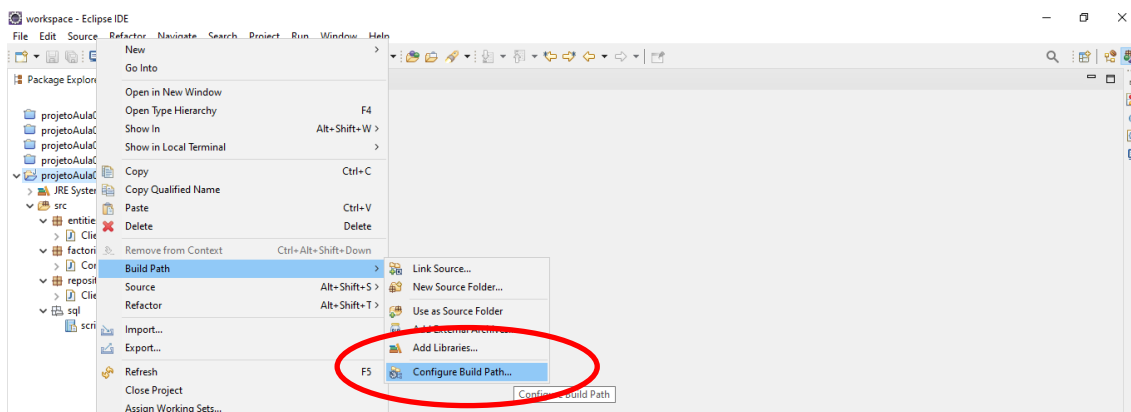
<https://mvnrepository.com/artifact/org.postgresql/postgresql/42.5.1>



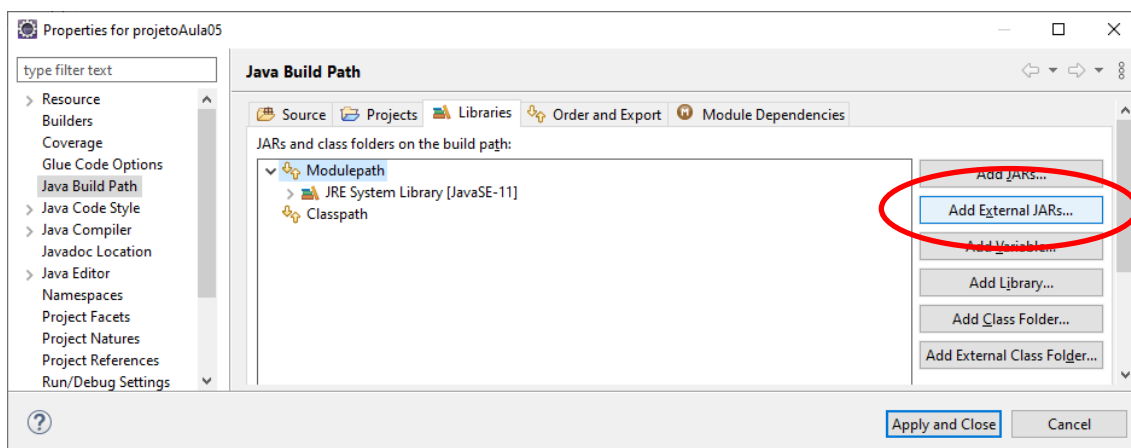
The screenshot shows the Maven Repository artifact page for 'org.postgresql:postgresql:42.5.1'. The page displays the 'PostgreSQL JDBC Driver' with a license of 'BSD 2-clause' and categories of 'JDBC Drivers', 'database', 'sql', 'jdbc', 'postgresql', and 'driver'. The ranking is #118 in Maven Repository and #2 in JDBC Drivers. The page shows a table of versions with columns for Version, Vulnerabilities, Repository, Usages, and Date. The table lists versions 42.5.1, 42.5.0, 42.4.3, 42.4.2, and 42.4.1. Version 42.5.1 is highlighted with a red arrow and has 228 usages. Version 42.5.0 has 299 usages and is marked as '1 vulnerability'. Version 42.4.3 has 43 usages. Version 42.4.2 has 55 usages and is marked as '1 vulnerability'. Version 42.4.1 has 259 usages and is marked as '1 vulnerability'.

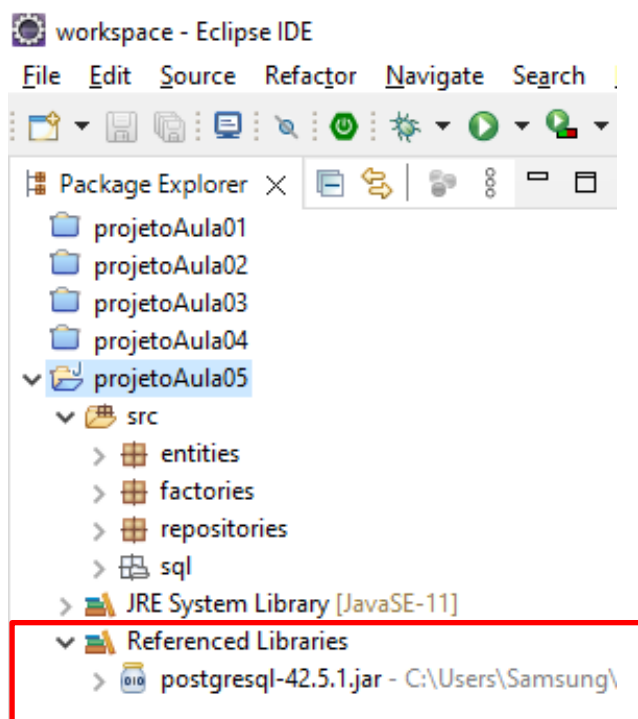
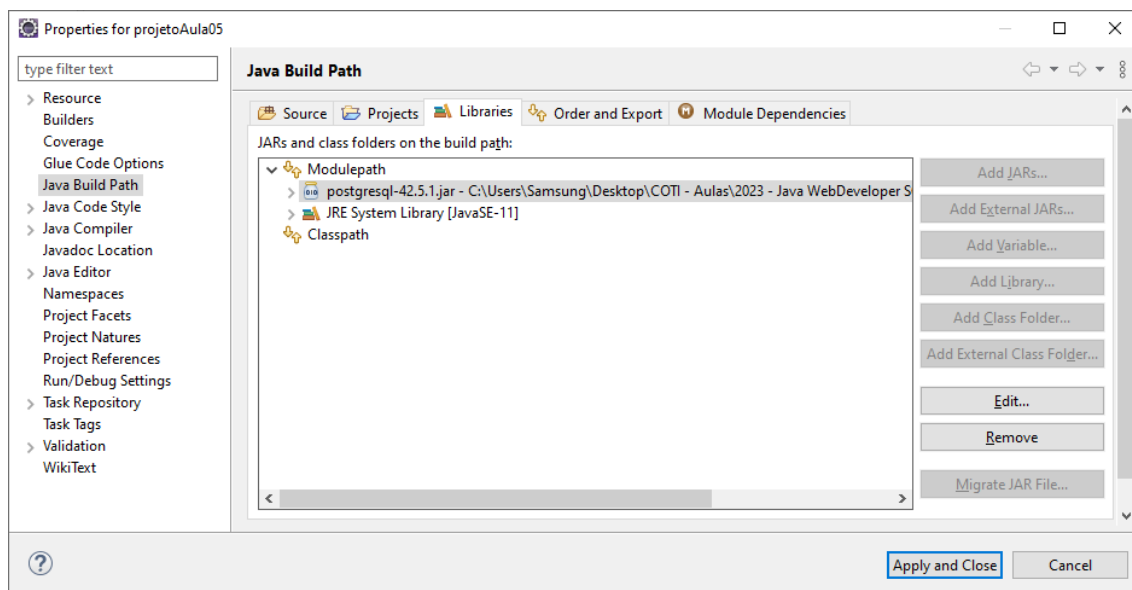


Adicionando esta biblioteca no projeto: BUILD PATH / CONFIGURE BUILD PATH



LIBRARIES / ADD EXTERNAL JARS





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();
    }

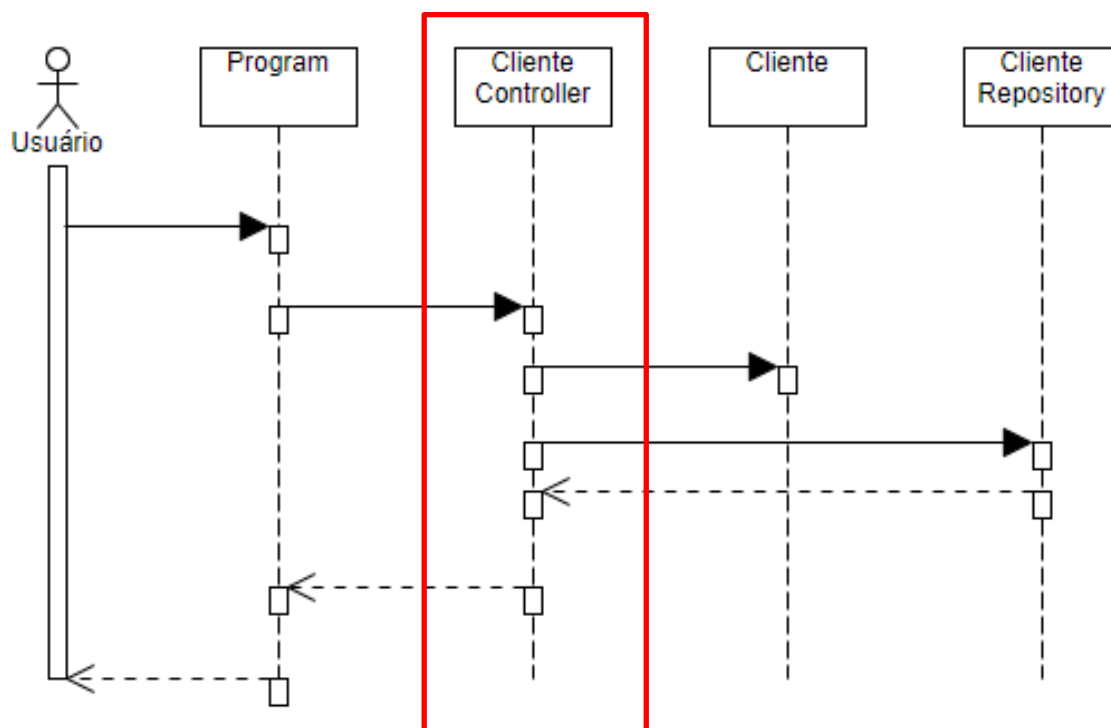
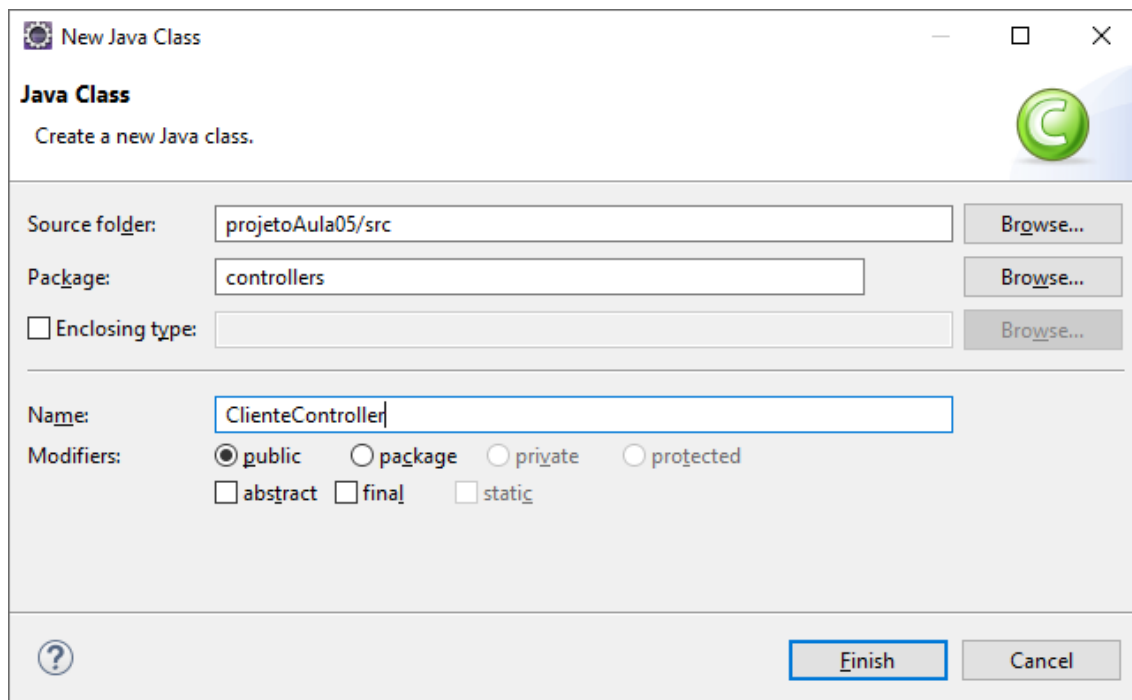
    // método para atualizar os dados
    //de um cliente no banco de dados
    public void update(Cliente cliente) throws Exception {
        // TODO
    }

    // 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**



```
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!");

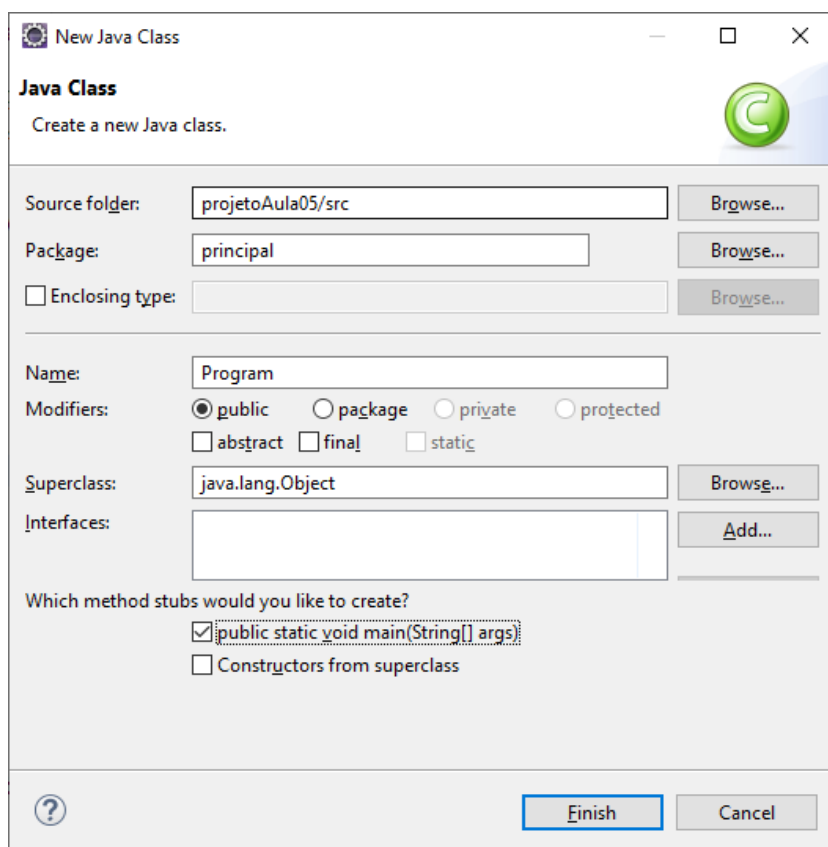
        }
    }
}
```

```

        catch(Exception e) {
            System.out.println
                ("\nFALHA AO CADASTRAR O CLIENTE");
            e.printStackTrace();
        }
    }
}

```

Criando a classe de inicialização do projeto:



New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☒ `public static void main(String[] args)`

☐ Constructors from superclass

```
package principal;
```

```
import controllers.ClienteController;
```

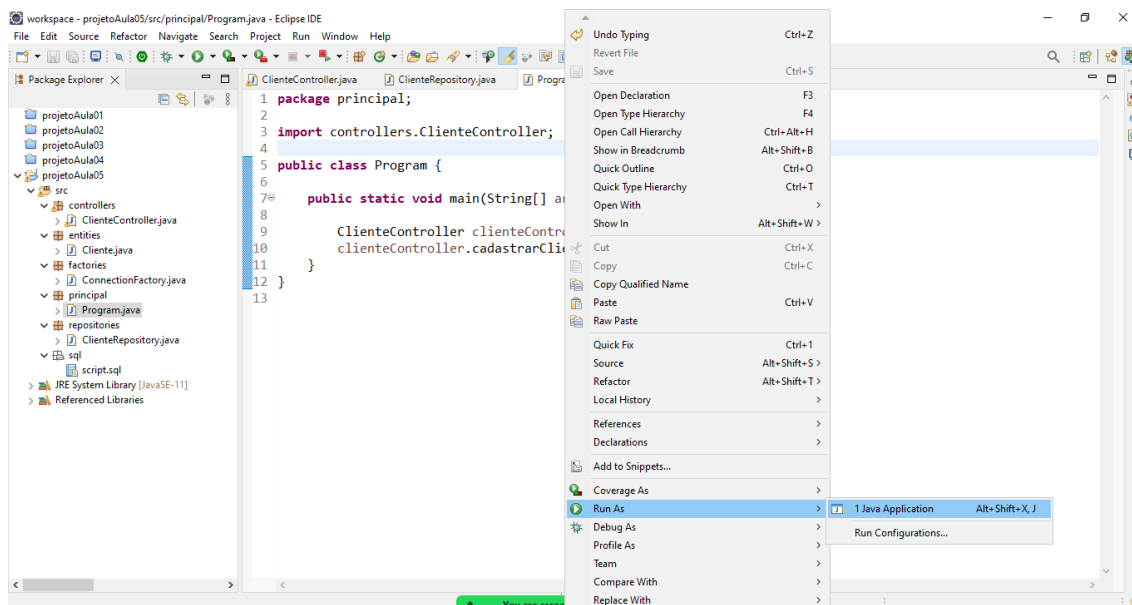
```
public class Program {
```

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

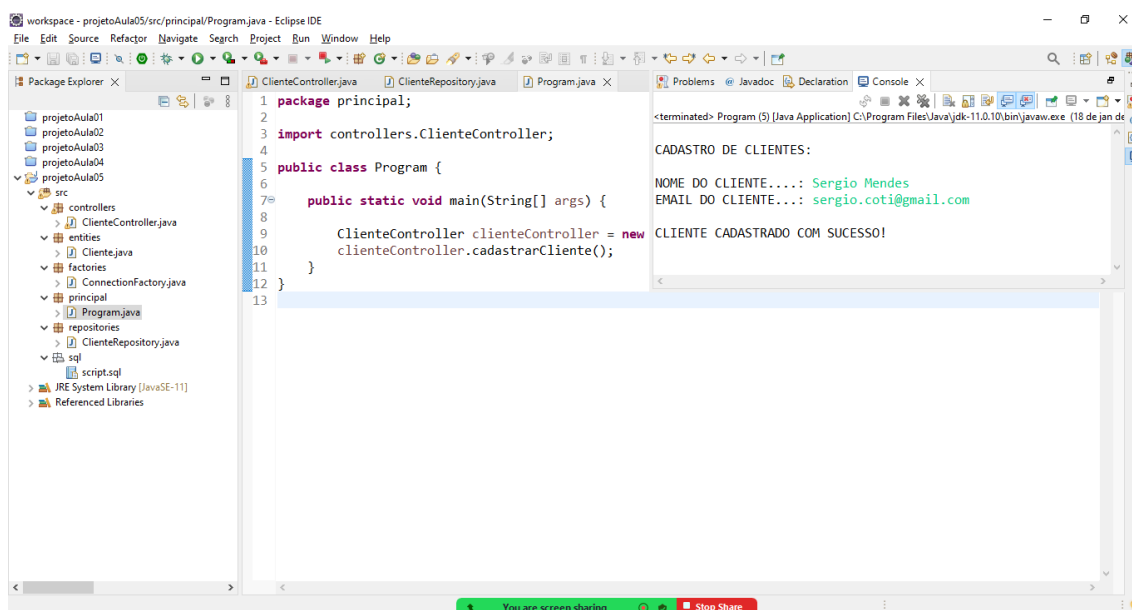
```
        ClienteController clienteController
            = new ClienteController();
        clienteController.cadastrarCliente();
    }
```

```
}
```

Executando: RUN AS / JAVA APPLICATION



Saída do programa:



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;
```

