

Voltando na classe de repositório:

/repositories/**ClienteRepository.java**

** Implementando os demais métodos para acesso ao banco de dados e CRUD (Create, Read, Update e Delete) de clientes.

java.sql

Biblioteca nativa do Java utilizada para realizar operações em bancos de dados, é composta por 3 principais componentes:

Connection

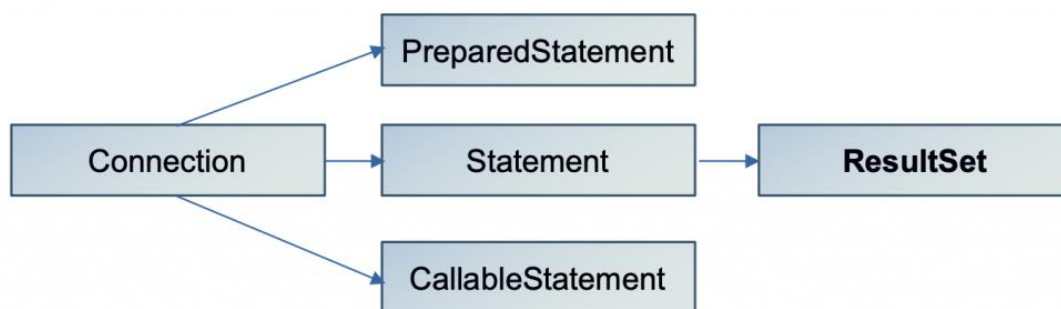
Interface da biblioteca java.sql utilizada para armazenar a conexão aberta com o banco de dados.

PreparedStatement

Interface da biblioteca java.sql utilizada para executar instruções SQL (comandos) no banco de dados, tais como INSERT, UPDATE, DELETE, SELECT etc.

ResultSet

Interface da biblioteca java.sql utilizada para armazenar resultados obtidos de consultas feitas no banco de dados. Sempre após a execução de um comando SELECT, iremos utilizar o ResultSet para armazenar e percorrer os registros obtidos da consulta realizada no banco.



```
package repositories;
```

```
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.util.ArrayList;  
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 {

        //abrindo conexão com o banco de dados
        Connection connection = ConnectionFactory.createConnection();

        //executando um comando SQL no banco de
        //dados para atualizar o cliente
        PreparedStatement preparedStatement
            = connection.prepareStatement("update cliente set
                                         nome=?, email=? where idcliente=?");

        //passando os parametros do comando SQL
        preparedStatement.setString(1, cliente.getNome());
        preparedStatement.setString(2, cliente.getEmail());
        preparedStatement.setInt(3, cliente.getIdCliente());

        //executar o comando SQL e fechar a conexão com o banco de dados
        preparedStatement.execute();
        connection.close();
    }

    // método para excluir um cliente no banco de dados
    public void delete(Cliente cliente) throws Exception {

        //abrindo conexão com o banco de dados
        Connection connection = ConnectionFactory.createConnection();

        //executando um comando SQL no banco
        //de dados para excluir o cliente
    }
}
```

```
PreparedStatement preparedStatement
    = connection.prepareStatement
        ("delete from cliente where idcliente=?");

//passando os parametros do comando SQL
preparedStatement.setInt(1, cliente.getIdCliente());

//executar o comando SQL e fechar a conexão com o banco de dados
preparedStatement.execute();
connection.close();
}

// método para retornar todos os clientes cadastrados no banco de dados
public List<Cliente> findAll() throws Exception {

    //abrindo conexão com o banco de dados
    Connection connection = ConnectionFactory.createConnection();

    //executando um comando SQL no banco
    //de dados para consultar os clientes
    PreparedStatement preparedStatement
        = connection.prepareStatement("select * from cliente");

    //ler e armazenar os registros obtidos do banco de dados
    ResultSet resultSet = preparedStatement.executeQuery();

    //declarando uma lista de clientes vazia
    List<Cliente> lista = new ArrayList<Cliente>();

    //percorrendo cada registro obtido do banco de dados
    while(resultSet.next())
    {
        Cliente cliente = new Cliente();

        cliente.setIdCliente(resultSet.getInt("idcliente"));
        cliente.setNome(resultSet.getString("nome"));
        cliente.setEmail(resultSet.getString("email"));

        lista.add(cliente); //adicionar o cliente na lista
    }

    //fechando a conexão com o banco de dados
    connection.close();

    //retornando a lista
    return lista;
}

// método para retornar 1 cliente cadastrados
// no banco de dados através do ID
public Cliente findById(Integer idCliente) throws Exception {

    //abrindo conexão com o banco de dados
    Connection connection = ConnectionFactory.createConnection();
```

```
//executando um comando SQL no banco
//de dados para consultar 1 cliente através do ID
PreparedStatement preparedStatement
    = connection.prepareStatement
    ("select * from cliente where idcliente=?");
preparedStatement.setInt(1, idCliente);

//ler e armazenar os registros obtidos do banco de dados
ResultSet resultSet = preparedStatement.executeQuery();

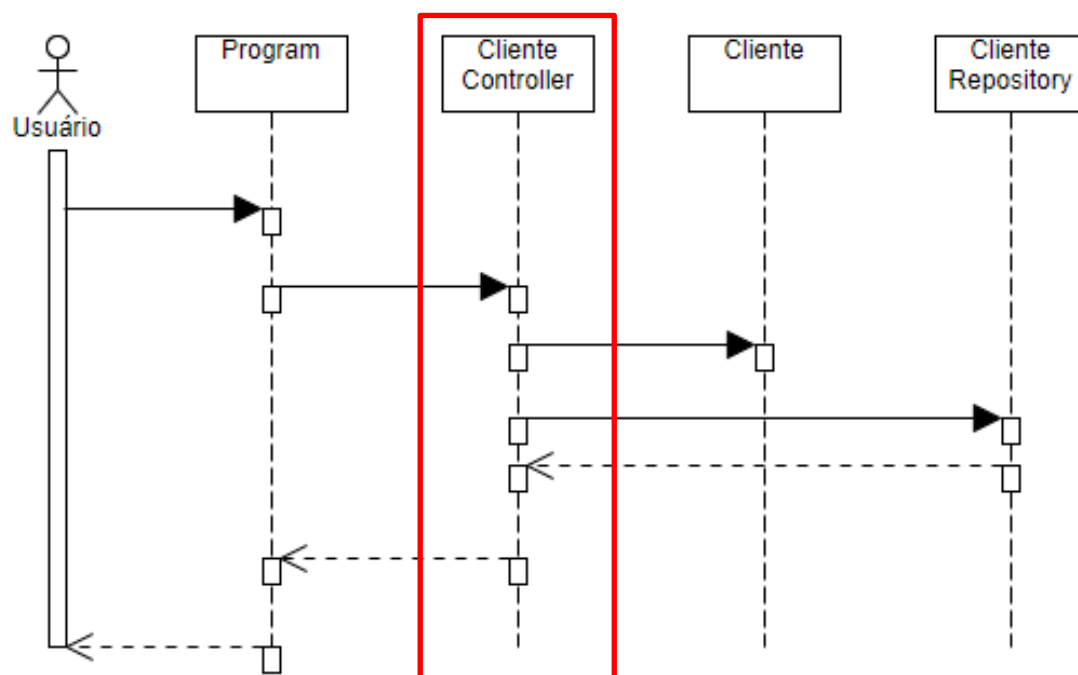
//criando um objeto Cliente vazio
Cliente cliente = null;

//verificando se algum cliente foi encontrado no banco de dados
if(resultSet.next())
{
    cliente = new Cliente();

    cliente.setIdCliente(resultSet.getInt("idcliente"));
    cliente.setNome(resultSet.getString("nome"));
    cliente.setEmail(resultSet.getString("email"));
}

//fechando a conexão com o banco de dados
connection.close();
//retornando o cliente
return cliente;
}
```

Voltando para o controlador:
/controllers/**ClienteController.java**



```
package controllers;

import java.util.List;
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();
        }
    }

    // método para executar o fluxo de atualização
    // de um cliente no banco de dados
    public void atualizarCliente() {

        try {

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

            Scanner scanner = new Scanner(System.in);

            System.out.print("INFORME O ID DO CLIENTE.....: ");
            Integer idCliente = Integer.parseInt(scanner.nextLine());

            //consultando o cliente no banco de dados através do id..
            ClienteRepository clienteRepository = new ClienteRepository();
            Cliente cliente = clienteRepository.findById(idCliente);

            //verificando se o cliente foi encontrado no banco de dados
            if(cliente != null) {

                System.out.print("ALTERE O NOME.....: ");
                cliente.setNome(scanner.nextLine());
            }
        }
    }
}
```

```
        System.out.print("ALTERE O EMAIL.....: ");
        cliente.setEmail(scanner.nextLine());

        //atualizando o cliente no banco de dados
        clienteRepository.update(cliente);

        System.out.println
            ("\nCLIENTE ATUALIZADO COM SUCESSO.");
    }
    else {
        System.out.println("\nCLIENTE NÃO ENCONTRADO.");
    }
}
catch(Exception e) {
    System.out.println("\nFALHA AO ATUALIZAR O CLIENTE.");
    e.printStackTrace();
}
}

// método para executar o fluxo de exclusão
// de um cliente no banco de dados
public void excluirCliente() {

    try {

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

        Scanner scanner = new Scanner(System.in);

        System.out.print("INFORME O ID DO CLIENTE.....: ");
        Integer idCliente = Integer.parseInt(scanner.nextLine());

        //consultando o cliente no banco de dados através do id..
        ClienteRepository clienteRepository = new ClienteRepository();
        Cliente cliente = clienteRepository.findById(idCliente);

        //verificando se o cliente foi encontrado
        if(cliente != null) {

            //excluindo o cliente
            clienteRepository.delete(cliente);

            System.out.println("\nCLIENTE EXCLUÍDO COM SUCESSO.");
        }
        else {
            System.out.println("\nCLIENTE NÃO ENCONTRADO.");
        }
    }
    catch(Exception e) {
        System.out.println("\nFALHA AO EXCLUIR O CLIENTE.");
        e.printStackTrace();
    }
}

// método para executar o fluxo de consulta
// de clientes no banco de dados
public void consultarClientes() {

    try {

        System.out.println("\nCONSULTA DE CLIENTES:\n");
```

```

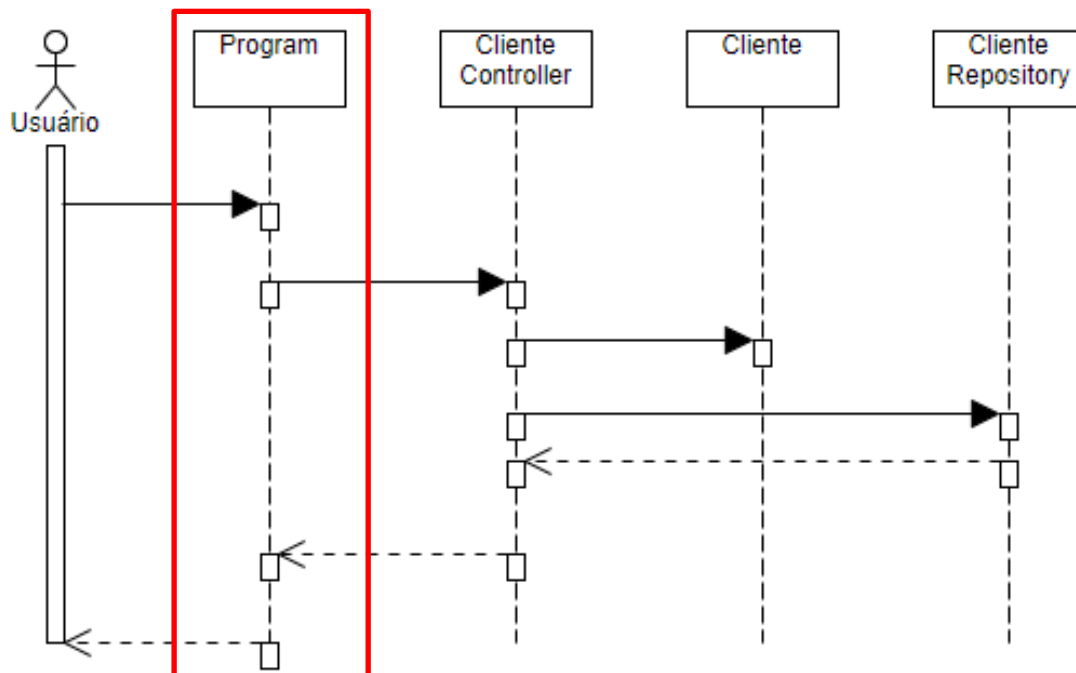
ClienteRepository clienteRepository = new ClienteRepository();
List<Cliente> lista = clienteRepository.findAll();

for(Cliente cliente : lista) {

    System.out.println("ID DO CLIENTE...: "
        + cliente.getIdCliente());
    System.out.println("NOME.....: "
        + cliente.getNome());
    System.out.println("EMAIL.....: "
        + cliente.getEmail());
    System.out.println("...");
}
}
catch(Exception e) {
    System.out.println("\nFALHA AO CONSULTAR CLIENTES.");
    e.printStackTrace();
}
}
}

```

Voltando para a classe **Program.java**
Classe de inicialização do projeto.



```

package principal;

import java.util.Scanner;
import controllers.ClienteController;

public class Program {

    public static void main(String[] args) {

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

```

```
System.out.println("(1) - CADASTRAR CLIENTE");
System.out.println("(2) - ATUALIZAR CLIENTE");
System.out.println("(3) - EXCLUIR CLIENTE");
System.out.println("(4) - CONSULTAR CLIENTES");
```

```
System.out.print("\nENTRE COM A OPÇÃO DESEJADA: ");
Integer opcao = Integer.parseInt(scanner.nextLine());
```

```
ClienteController clienteController
    = new ClienteController();
```

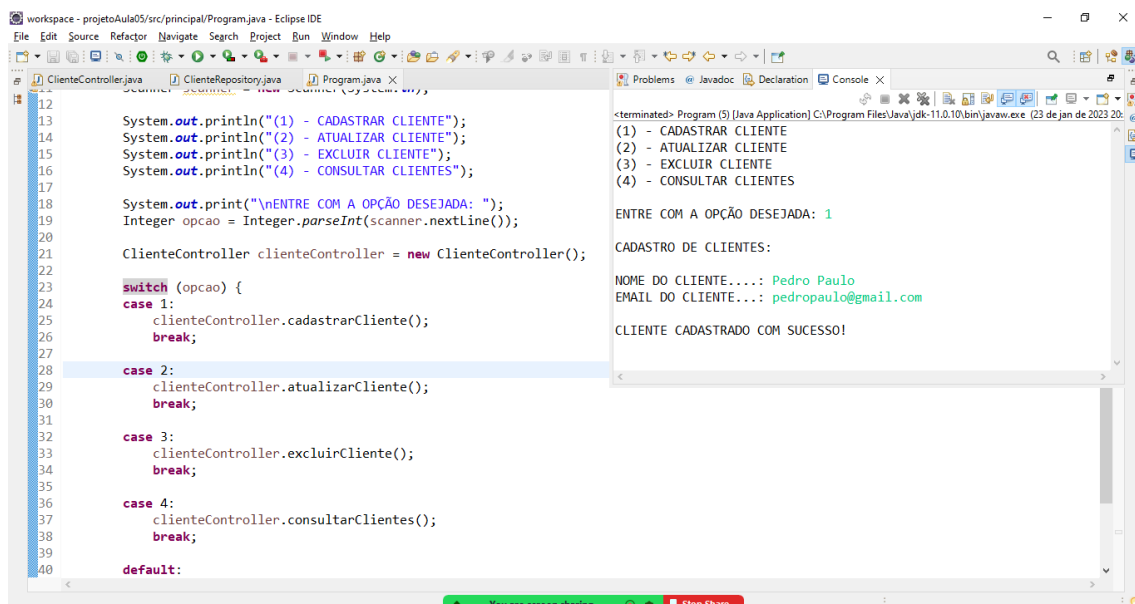
```
switch (opcao) {
case 1:
    clienteController.cadastrarCliente();
    break;

case 2:
    clienteController.atualizarCliente();
    break;

case 3:
    clienteController.excluirCliente();
    break;

case 4:
    clienteController.consultarClientes();
    break;

default:
    System.out.println("\nOPÇÃO INVÁLIDA.");
    break;
}
```



The screenshot shows the Eclipse IDE with the following code in the editor:

```
12
13 System.out.println("(1) - CADASTRAR CLIENTE");
14 System.out.println("(2) - ATUALIZAR CLIENTE");
15 System.out.println("(3) - EXCLUIR CLIENTE");
16 System.out.println("(4) - CONSULTAR CLIENTES");
17
18 System.out.print("\nENTRE COM A OPÇÃO DESEJADA: ");
19 Integer opcao = Integer.parseInt(scanner.nextLine());
20
21 ClienteController clienteController = new ClienteController();
22
23 switch (opcao) {
24 case 1:
25     clienteController.cadastrarCliente();
26     break;
27
28 case 2:
29     clienteController.atualizarCliente();
30     break;
31
32 case 3:
33     clienteController.excluirCliente();
34     break;
35
36 case 4:
37     clienteController.consultarClientes();
38     break;
39
40 default:
```

The console output shows the following sequence of events:

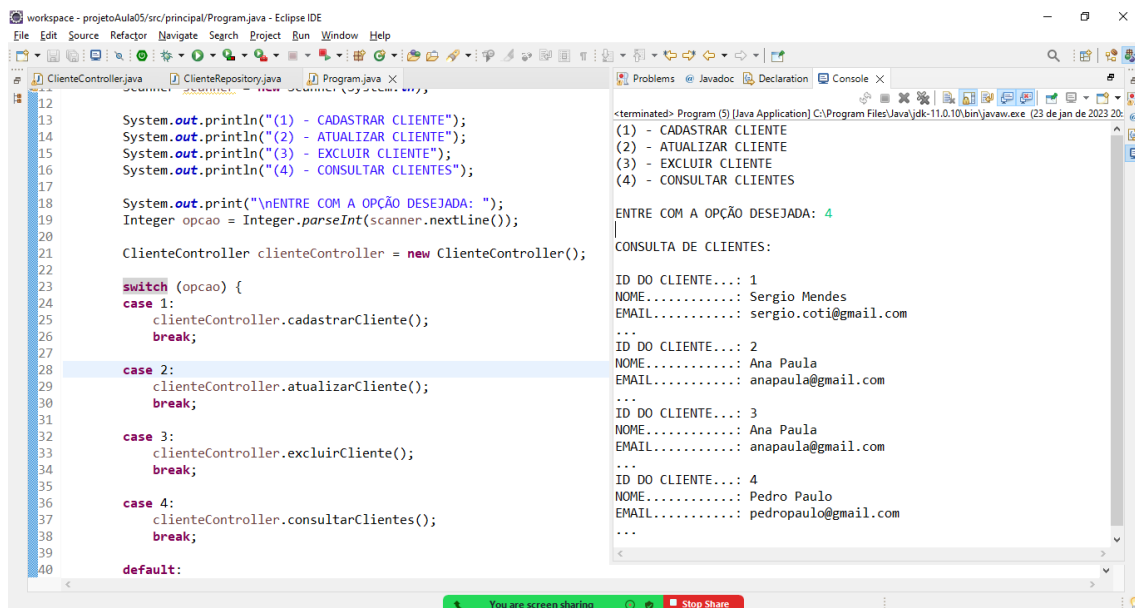
```
<terminated> Program (5) [Java Application] C:\Program Files\Java\jdk-11.0.10\bin\javaw.exe (23 de jan de 2023 20:
(1) - CADASTRAR CLIENTE
(2) - ATUALIZAR CLIENTE
(3) - EXCLUIR CLIENTE
(4) - CONSULTAR CLIENTES

ENTRE COM A OPÇÃO DESEJADA: 1

CADASTRO DE CLIENTES:

NOME DO CLIENTE....: Pedro Paulo
EMAIL DO CLIENTE....: pedropaulo@gmail.com

CLIENTE CADASTRADO COM SUCESSO!
```

```

12
13 System.out.println("(1) - CADASTRAR CLIENTE");
14 System.out.println("(2) - ATUALIZAR CLIENTE");
15 System.out.println("(3) - EXCLUIR CLIENTE");
16 System.out.println("(4) - CONSULTAR CLIENTES");
17
18 System.out.print("\nENTRE COM A OPÇÃO DESEJADA: ");
19 Integer opcao = Integer.parseInt(scanner.nextLine());
20
21 ClienteController clienteController = new ClienteController();
22
23 switch (opcao) {
24     case 1:
25         clienteController.cadastrarCliente();
26         break;
27
28     case 2:
29         clienteController.atualizarCliente();
30         break;
31
32     case 3:
33         clienteController.excluirCliente();
34         break;
35
36     case 4:
37         clienteController.consultarClientes();
38         break;
39
40     default:

```

```

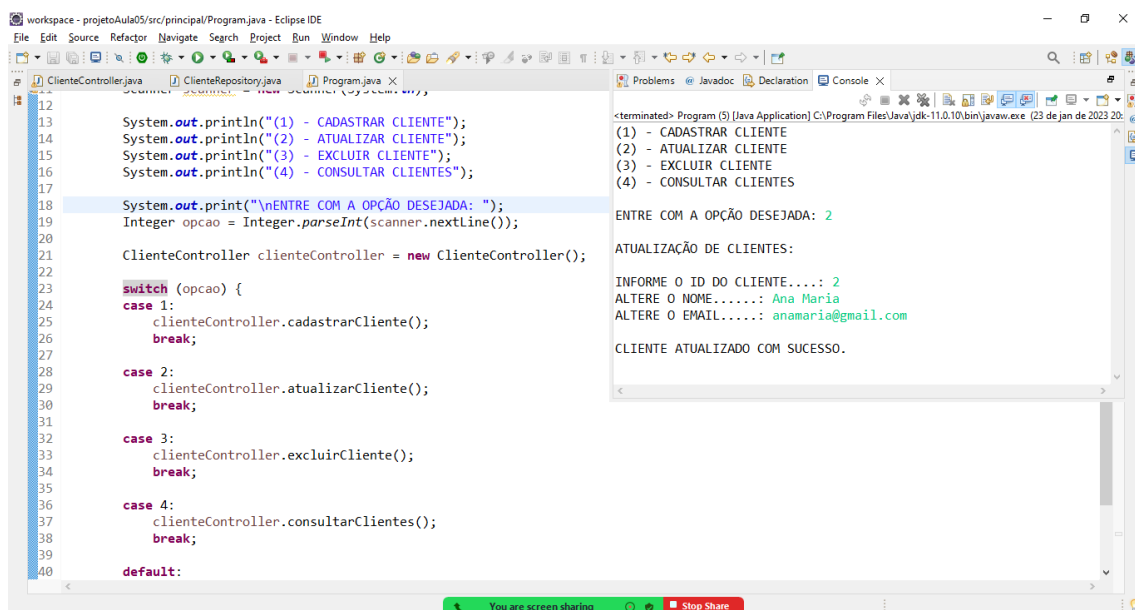
<terminated> Program (5) [Java Application] C:\Program Files\Java\jdk-11.0.10\bin\javaw.exe (23 de jan de 2023 20:
(1) - CADASTRAR CLIENTE
(2) - ATUALIZAR CLIENTE
(3) - EXCLUIR CLIENTE
(4) - CONSULTAR CLIENTES

ENTRE COM A OPÇÃO DESEJADA: 4

CONSULTA DE CLIENTES:

ID DO CLIENTE....: 1
NOME.....: Sergio Mendes
EMAIL.....: sergio.coti@gmail.com
...
ID DO CLIENTE....: 2
NOME.....: Ana Paula
EMAIL.....: anapaula@gmail.com
...
ID DO CLIENTE....: 3
NOME.....: Ana Paula
EMAIL.....: anapaula@gmail.com
...
ID DO CLIENTE....: 4
NOME.....: Pedro Paulo
EMAIL.....: pedropaulo@gmail.com
...

```



```

12
13 System.out.println("(1) - CADASTRAR CLIENTE");
14 System.out.println("(2) - ATUALIZAR CLIENTE");
15 System.out.println("(3) - EXCLUIR CLIENTE");
16 System.out.println("(4) - CONSULTAR CLIENTES");
17
18 System.out.print("\nENTRE COM A OPÇÃO DESEJADA: ");
19 Integer opcao = Integer.parseInt(scanner.nextLine());
20
21 ClienteController clienteController = new ClienteController();
22
23 switch (opcao) {
24     case 1:
25         clienteController.cadastrarCliente();
26         break;
27
28     case 2:
29         clienteController.atualizarCliente();
30         break;
31
32     case 3:
33         clienteController.excluirCliente();
34         break;
35
36     case 4:
37         clienteController.consultarClientes();
38         break;
39
40     default:

```

```

<terminated> Program (5) [Java Application] C:\Program Files\Java\jdk-11.0.10\bin\javaw.exe (23 de jan de 2023 20:
(1) - CADASTRAR CLIENTE
(2) - ATUALIZAR CLIENTE
(3) - EXCLUIR CLIENTE
(4) - CONSULTAR CLIENTES

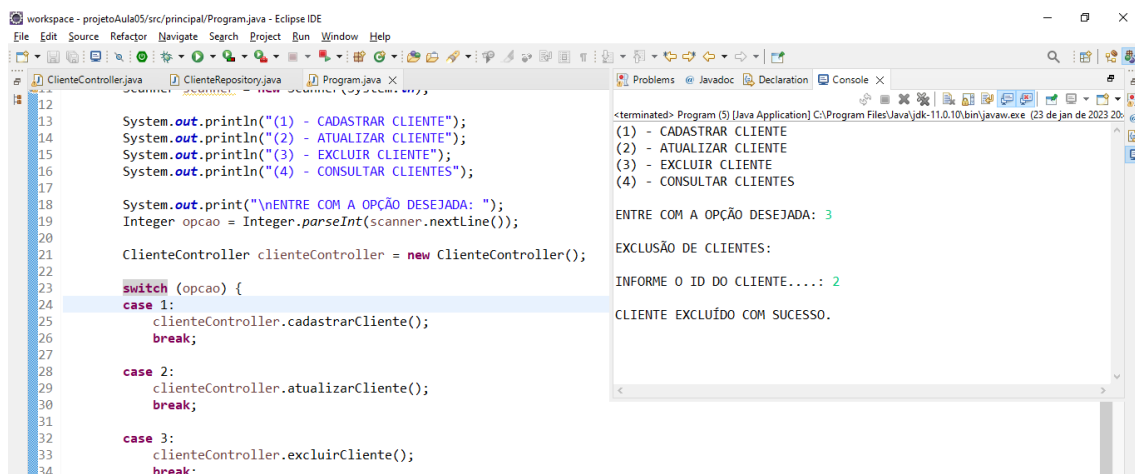
ENTRE COM A OPÇÃO DESEJADA: 2

ATUALIZAÇÃO DE CLIENTES:

INFORME O ID DO CLIENTE....: 2
ALTERE O NOME.....: Ana Maria
ALTERE O EMAIL.....: anamaria@gmail.com

CLIENTE ATUALIZADO COM SUCESSO.

```



```

12
13 System.out.println("(1) - CADASTRAR CLIENTE");
14 System.out.println("(2) - ATUALIZAR CLIENTE");
15 System.out.println("(3) - EXCLUIR CLIENTE");
16 System.out.println("(4) - CONSULTAR CLIENTES");
17
18 System.out.print("\nENTRE COM A OPÇÃO DESEJADA: ");
19 Integer opcao = Integer.parseInt(scanner.nextLine());
20
21 ClienteController clienteController = new ClienteController();
22
23 switch (opcao) {
24     case 1:
25         clienteController.cadastrarCliente();
26         break;
27
28     case 2:
29         clienteController.atualizarCliente();
30         break;
31
32     case 3:
33         clienteController.excluirCliente();
34         break;

```

```

<terminated> Program (5) [Java Application] C:\Program Files\Java\jdk-11.0.10\bin\javaw.exe (23 de jan de 2023 20:
(1) - CADASTRAR CLIENTE
(2) - ATUALIZAR CLIENTE
(3) - EXCLUIR CLIENTE
(4) - CONSULTAR CLIENTES

ENTRE COM A OPÇÃO DESEJADA: 3

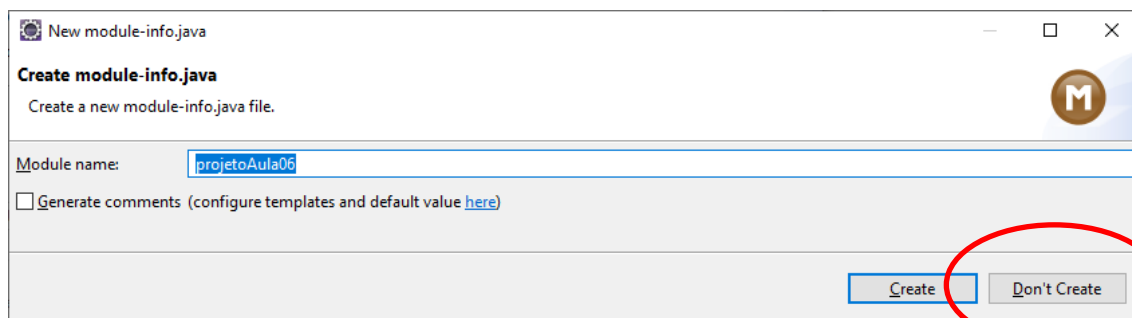
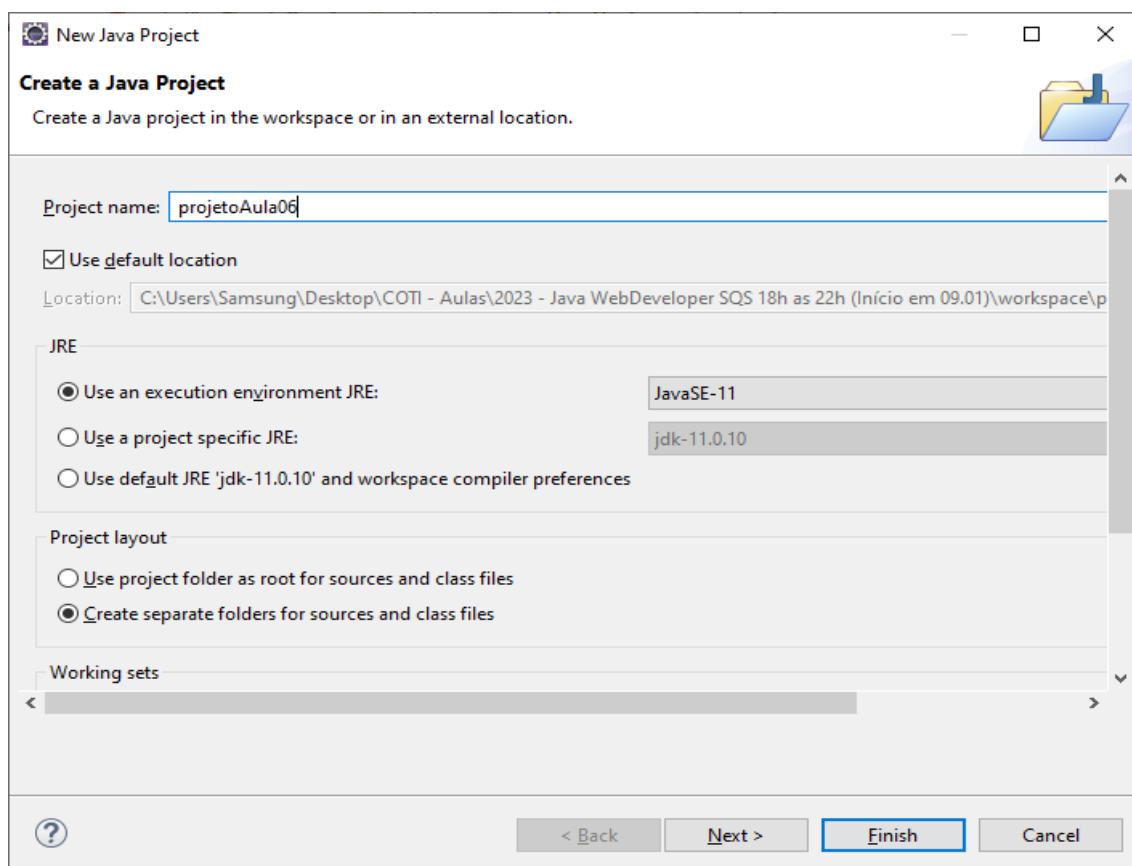
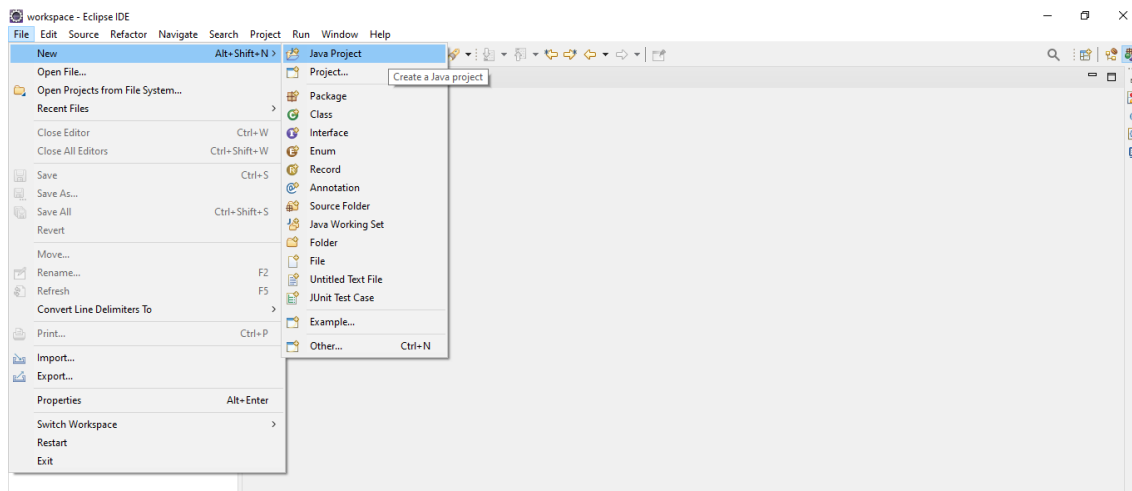
EXCLUSÃO DE CLIENTES:

INFORME O ID DO CLIENTE....: 2

CLIENTE EXCLUÍDO COM SUCESSO.

```

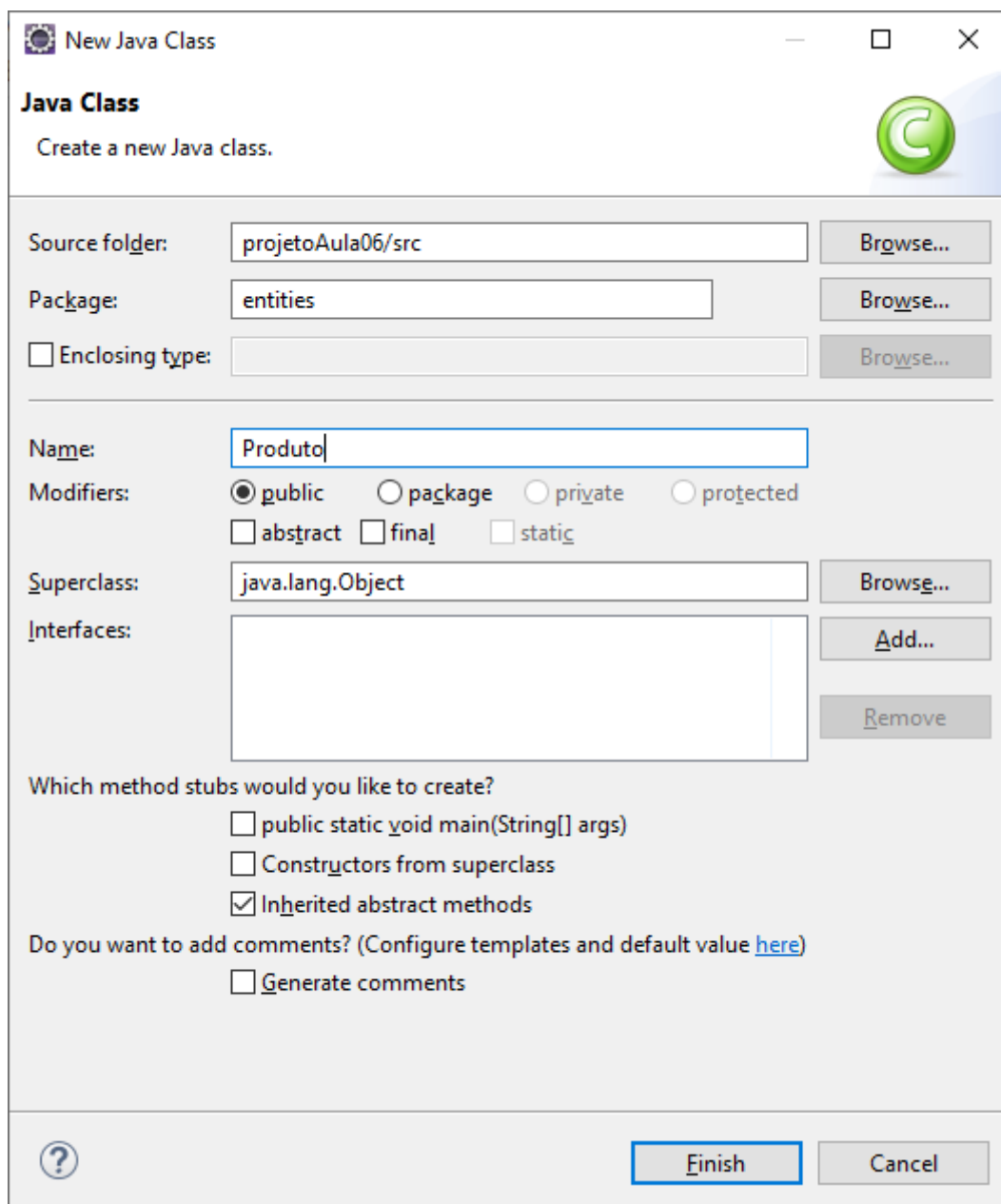
Novo projeto:



JavaBeans

Padrão Java para construção de classes que tem como objetivo modelar entidades de um sistema (modelo de dados). Essas classes são criadas em Java seguindo o padrão:

- Atributos privados
- Construtor sem entrada de argumentos
- Construtor com entrada de argumentos (sobrecarga de métodos)
- Métodos de encapsulamento
 - Setters
 - Getters
- Sobrescrita de métodos da classe Object



New Java Class

Java Class
Create a new Java class.

Source folder: [Browse...](#)

Package: [Browse...](#)

☐ Enclosing type: [Browse...](#)

Name:

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

Superclass: [Browse...](#)

Interfaces: [Add...](#) [Remove](#)

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

[?](#) [Finish](#) [Cancel](#)

Atributos privados

No JavaBean, todos os atributos são criados como “privados”.

```
package entities;

public class Produto {

    private Integer idProduto;
    private String nome;
    private Double preco;
    private Integer quantidade;
}
```

Construtor sem entrada de argumentos

Podemos declarar um método construtor default, ou seja, sem entrada de parâmetros / argumentos.

```
package entities;

public class Produto {

    private Integer idProduto;
    private String nome;
    private Double preco;
    private Integer quantidade;

    public Produto() {
        // TODO Auto-generated constructor stub
    }
}
```

Construtor com entrada de argumentos (sobrecarga de métodos)

Podemos declarar mais construtores fazendo sobrecarga de métodos, ou seja, construtores com entrada de argumentos, para facilitar o preenchimento de um objeto quando este for instanciado.

```
package entities;

public class Produto {

    private Integer idProduto;
    private String nome;
    private Double preco;
    private Integer quantidade;

    public Produto() {
        // TODO Auto-generated constructor stub
    }
}
```

```
public Produto(Integer idProduto, String nome, Double preco,
                Integer quantidade) {
    super();
    this.idProduto = idProduto;
    this.nome = nome;
    this.preco = preco;
    this.quantidade = quantidade;
}
}
```

Métodos de encapsulamento

Para cada atributo privado da classe, podemos declarar métodos públicos que irão fazer a entrada / saída de dados de cada atributo, estes métodos são chamados de setters e getters.

```
package entities;

public class Produto {

    private Integer idProduto;
    private String nome;
    private Double preco;
    private Integer quantidade;

    public Produto() {
        // TODO Auto-generated constructor stub
    }

    public Produto(Integer idProduto, String nome, Double preco,
                    Integer quantidade) {
        super();
        this.idProduto = idProduto;
        this.nome = nome;
        this.preco = preco;
        this.quantidade = quantidade;
    }

    public Integer getIdProduto() {
        return idProduto;
    }

    public void setIdProduto(Integer idProduto) {
        this.idProduto = idProduto;
    }

    public String getNome() {
        return nome;
    }

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

```
public Double getPreco() {  
    return preco;  
}  
  
public void setPreco(Double preco) {  
    this.preco = preco;  
}  
  
public Integer getQuantidade() {  
    return quantidade;  
}  
  
public void setQuantidade(Integer quantidade) {  
    this.quantidade = quantidade;  
}  
}
```

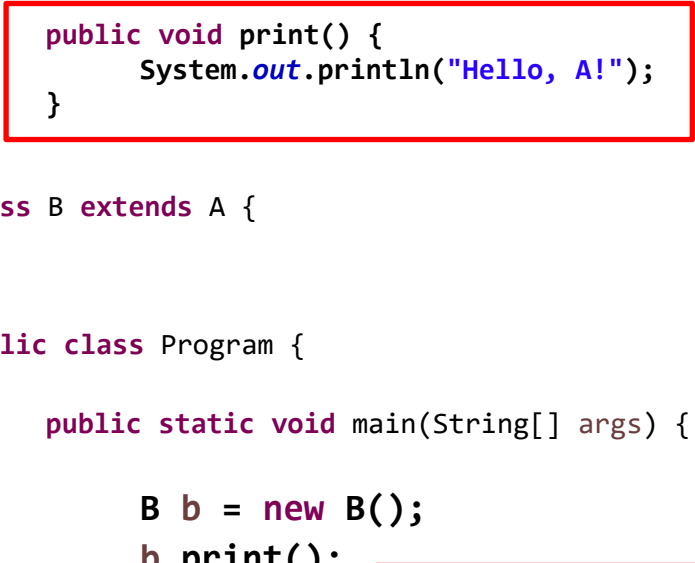
Sobrescrita de métodos (**Override**)

Ocorre quando uma subclasse redefine / sobrepõe métodos da sua superclasse. Ou seja, quando uma "classe filha" redefine o comportamento de um método da sua "classe pai".

Exemplo:

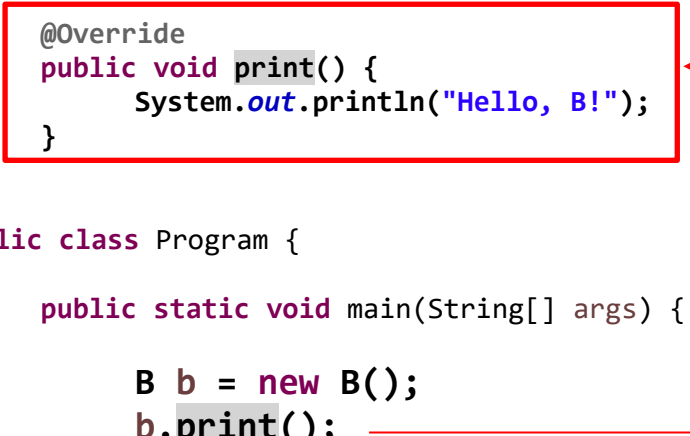
No código abaixo, a classe B executa o método print() herdado da classe A, e este método imprime o resultado "Hello, A!"

```
class A {  
    public void print() {  
        System.out.println("Hello, A!");  
    }  
}  
  
class B extends A {  
}  
  
public class Program {  
    public static void main(String[] args) {  
        B b = new B();  
        b.print();  
    }  
}
```

A red box highlights the print() method in class A. A red arrow points from the b.print() call in the Program class's main method to this box.

Na sobrescrita de métodos, a classe B poderia redefinir / sobrepor o método `print()` de A, dando a ele um novo comportamento:

```
class A {  
    public void print() {  
        System.out.println("Hello, A!");  
    }  
}  
  
class B extends A {  
    @Override  
    public void print() {  
        System.out.println("Hello, B!");  
    }  
}  
  
public class Program {  
    public static void main(String[] args) {  
        B b = new B();  
        b.print();  
    }  
}
```



Curiosidade:

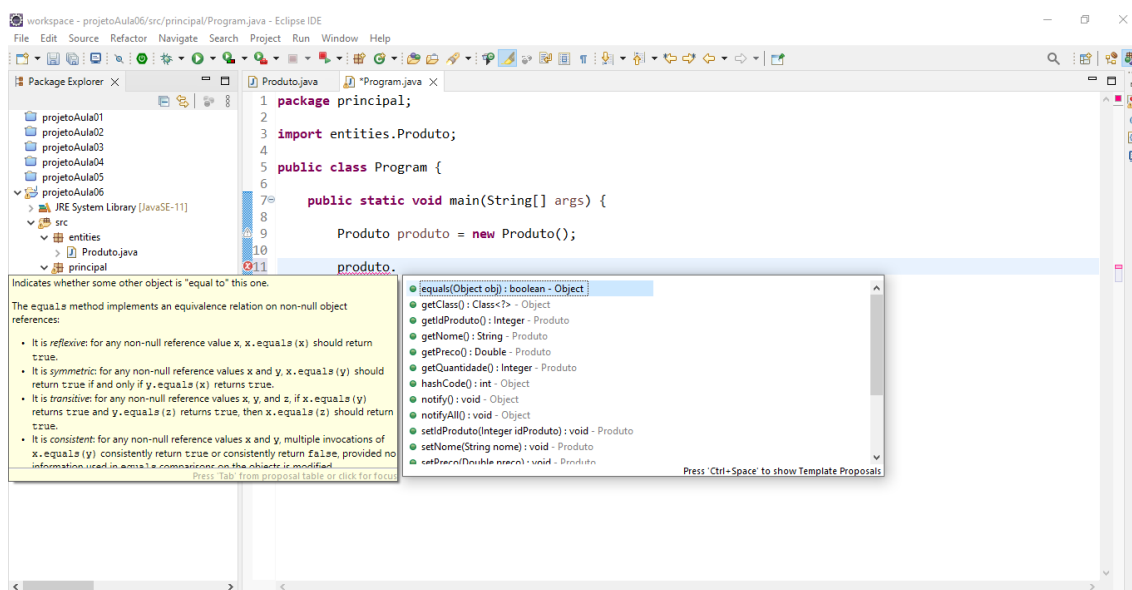
Se a "classe pai" declara um método como **final**, este método não pode ser sobrescrito pelas subclasses.

```
class A {  
    public final void print() {  
        System.out.println("Hello, A!");  
    }  
}  
  
class B extends A {  
    // This method cannot be overridden because A's print() is final  
}  
  
public class Program {  
    public static void main(String[] args) {  
        B b = new B();  
        b.print();  
    }  
}
```

Toda classe Java, implicitamente, herda uma superclasse chamada de Object

Dessa forma, toda classe Java herda métodos de Object, exemplo:

- equals
- hashCode
- getClass
- notify
- toString



Segundo o padrão JavaBean, podemos sobrescrever estes métodos conforme a necessidade da classe.

Exemplo:

Sobrescrita do método toString()

Método utilizado para retornar em uma única linha de texto todos os dados de uma entidade, voltado por exemplo para impressão dos campos de uma entidade.

```
package entities;
```

```
public class Produto {
```

```
    private Integer idProduto;
    private String nome;
    private Double preco;
    private Integer quantidade;
```

```
    public Produto() {
        // TODO Auto-generated constructor stub
    }
```



```
public Produto(Integer idProduto, String nome, Double preco,
                Integer quantidade) {
    super();
    this.idProduto = idProduto;
    this.nome = nome;
    this.preco = preco;
    this.quantidade = quantidade;
}

public Integer getIdProduto() {
    return idProduto;
}

public void setIdProduto(Integer idProduto) {
    this.idProduto = idProduto;
}

public String getNome() {
    return nome;
}

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

public Double getPreco() {
    return preco;
}

public void setPreco(Double preco) {
    this.preco = preco;
}

public Integer getQuantidade() {
    return quantidade;
}

public void setQuantidade(Integer quantidade) {
    this.quantidade = quantidade;
}

@Override
public String toString() {
    return "Produto [idProduto=" + idProduto + ", nome="
        + nome + ", preco=" + preco + ", quantidade="
        + quantidade
        + "]";
}
}
```

Testando o método toString()
/principal/**Program.java**

```
package principal;

import entities.Produto;

public class Program {

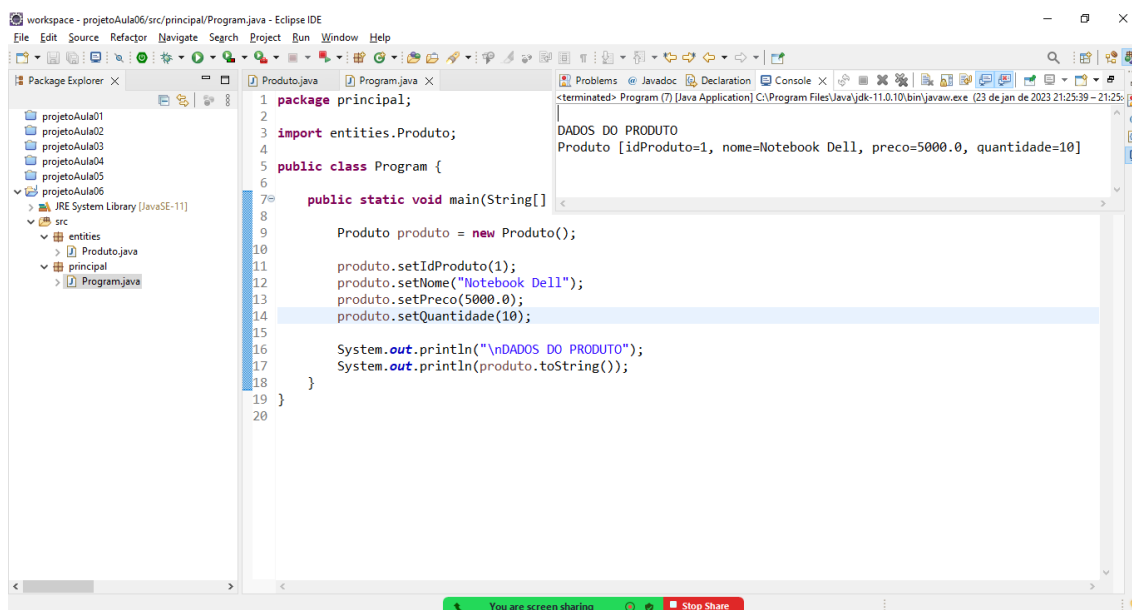
    public static void main(String[] args) {

        Produto produto = new Produto();

        produto.setIdProduto(1);
        produto.setNome("Notebook Dell");
        produto.setPreco(5000.0);
        produto.setQuantidade(10);

        System.out.println("\nDADOS DO PRODUTO");
        System.out.println(produto.toString());

    }
}
```

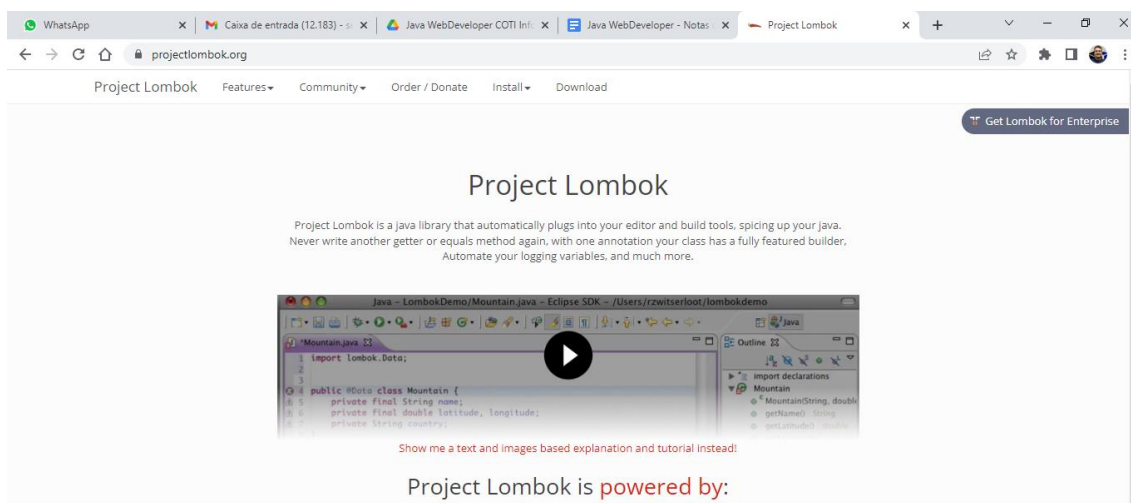


DADOS DO PRODUTO
Produto [idProduto=1, nome=Notebook Dell, preco=5000.0, quantidade=10]

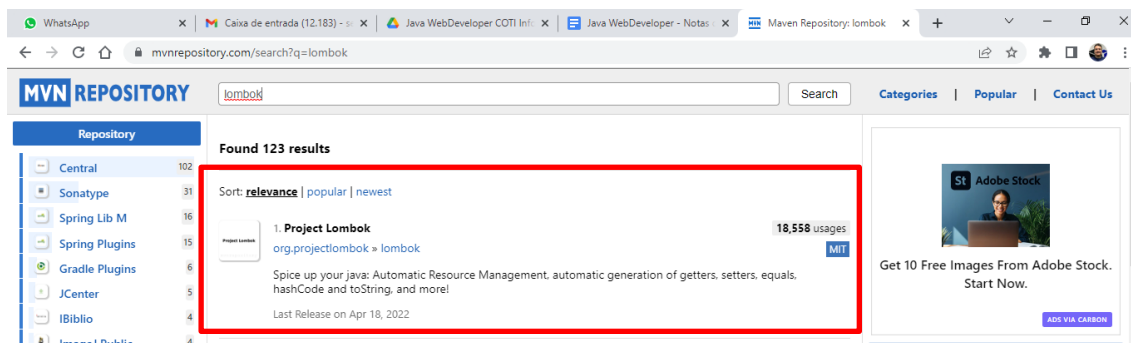
Lombok

Biblioteca utilizada para reduzir a “verbosidade” do código Java, ou seja, tem como proposta simplificar a escrita de classes já definindo de forma implícita métodos que a classe deverá ter.

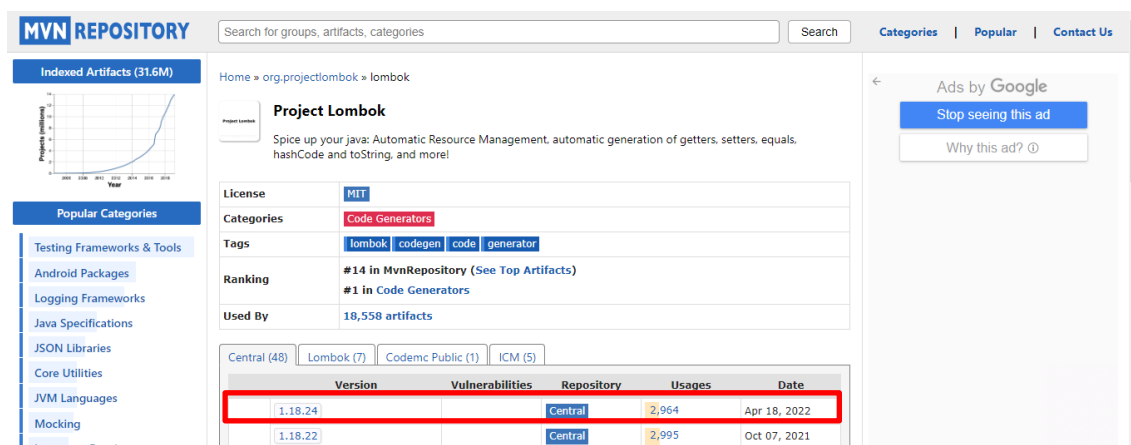
<https://projectlombok.org/>



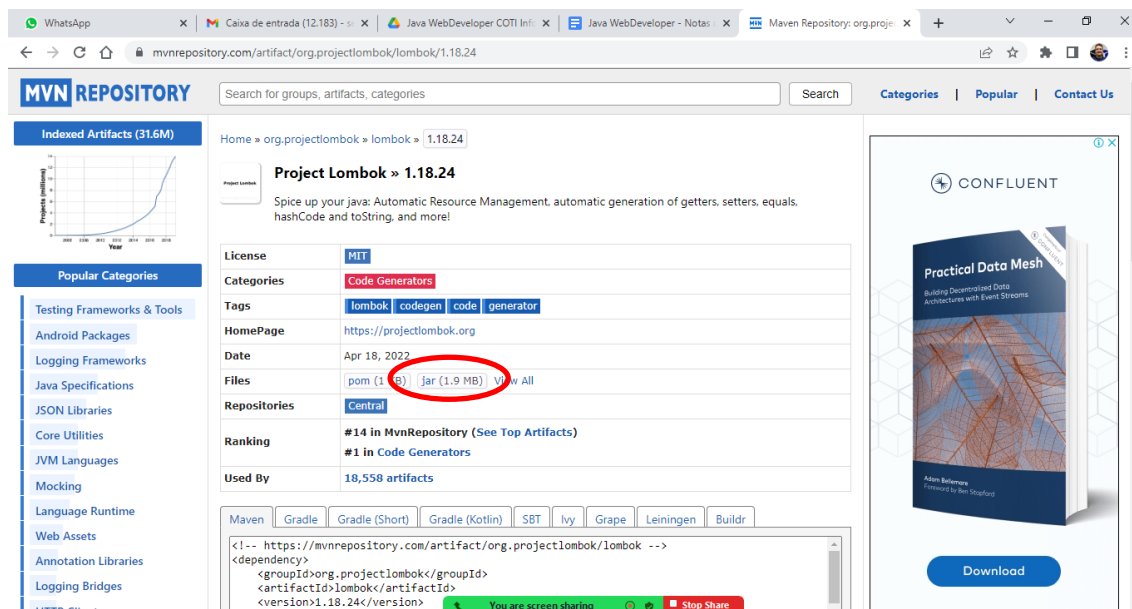
Baixando:



<https://mvnrepository.com/artifact/org.projectlombok/lombok>

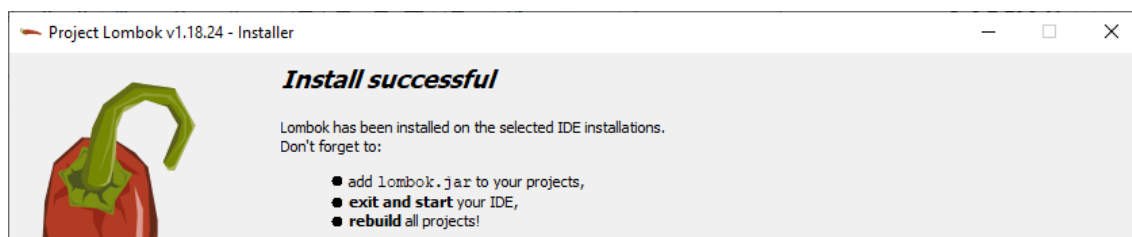


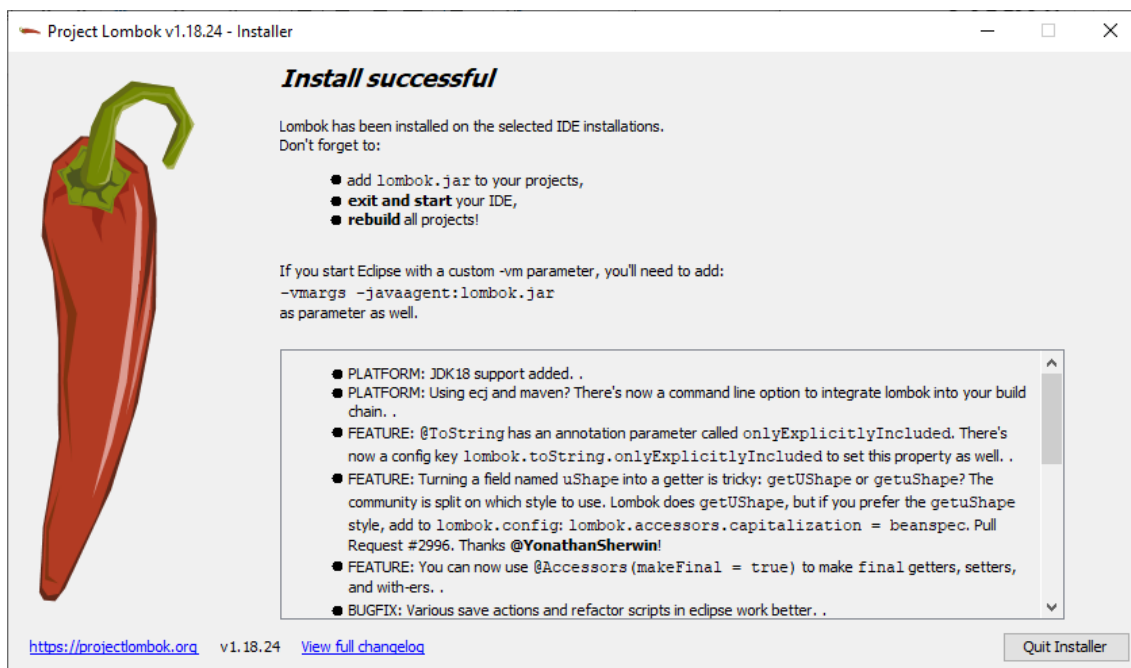
<https://mvnrepository.com/artifact/org.projectlombok/lombok/1.18.24>



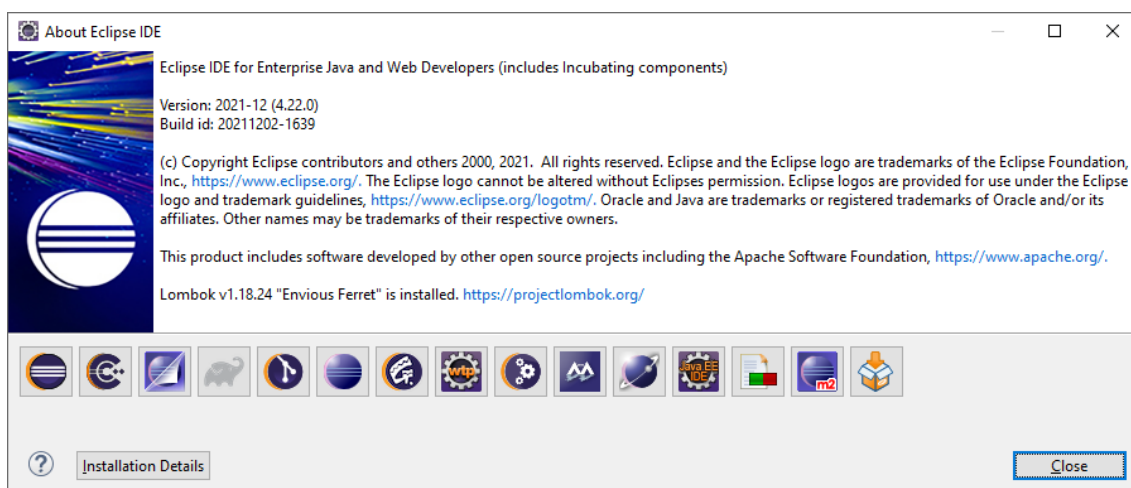
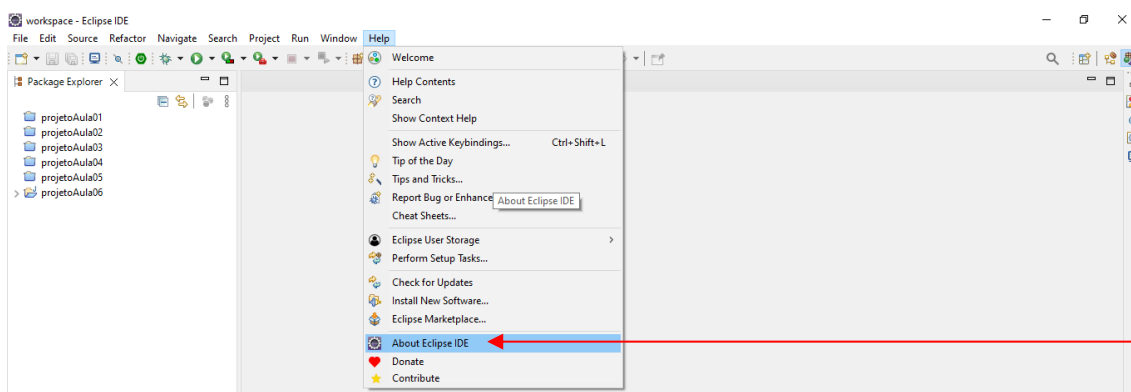
Instalando o plugin do Lombok no eclipse:

** Para isso, feche o eclipse.

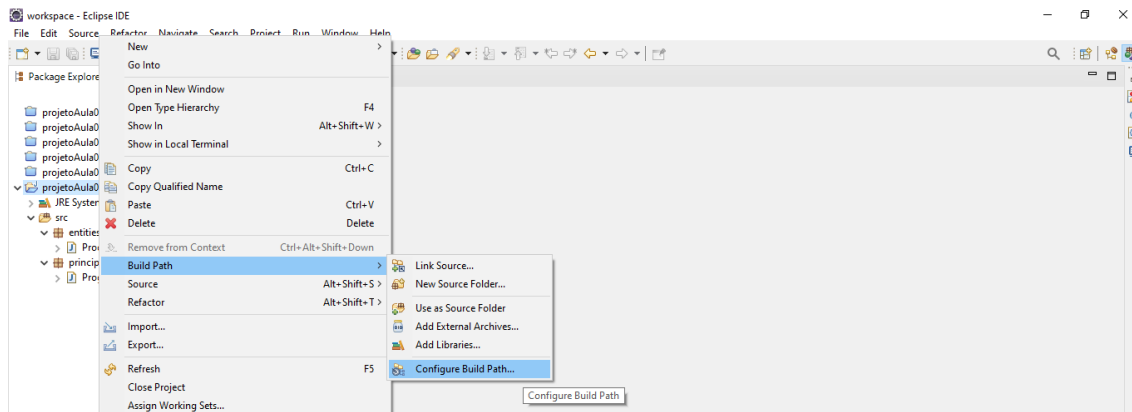


Voltando no eclipse.
Verificando a instalação:

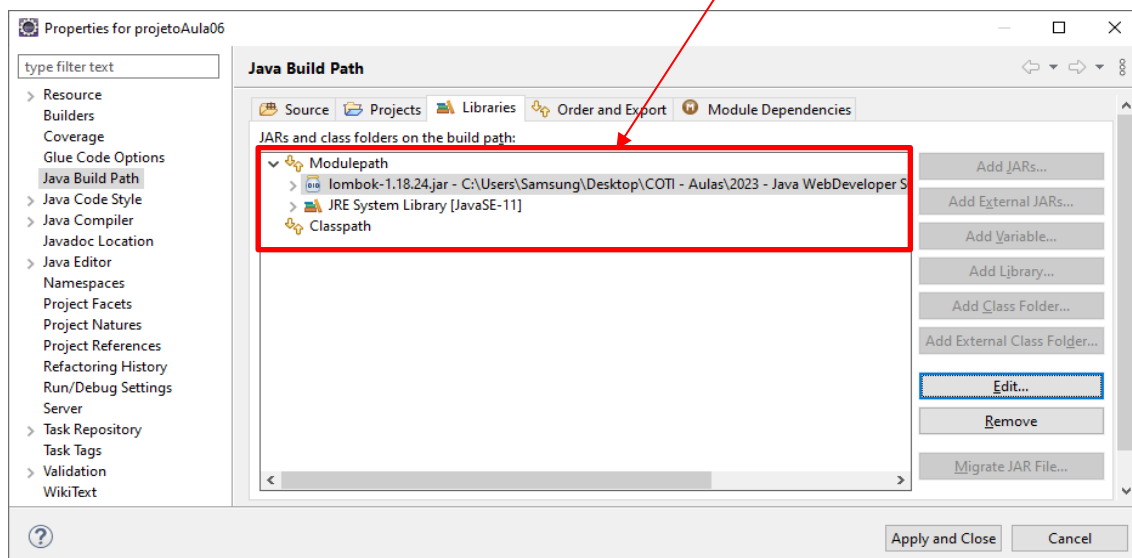
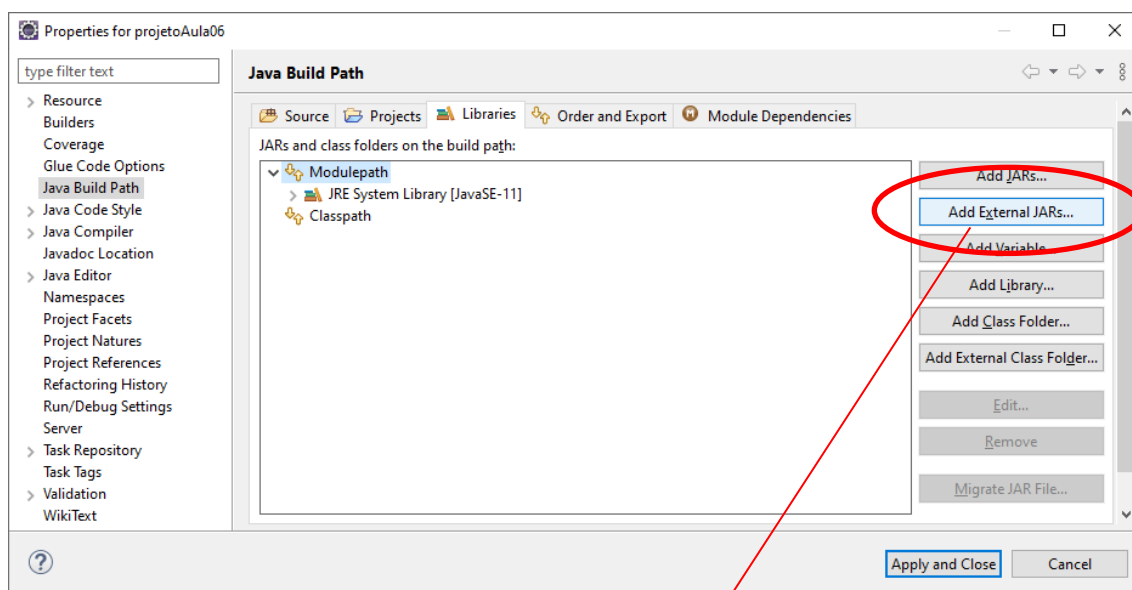


Lombok v1.18.24 "Envious Ferret" is installed. <https://projectlombok.org/>

Adicionando a biblioteca (JAR) do LOMBOK no projeto: BUILD PATH / CONFIGURE BUILD PATH



ADD EXTERNAL JARS



Aplicando o LOMBOK na classe de entidade

Refatorando o código da classe:

- Vamos utilizar as Annotations do LOMBOK para gerar de forma automática os elementos dos JavaBeans:
 - Atributos privados
 - Construtor sem entrada de argumentos
 - Construtor com entrada de argumentos (sobrecarga de métodos)
 - Métodos de encapsulamento
 - Setters
 - Getters
 - Sobrescrita de métodos da classe Object

```
package entities;
```

```
import lombok.AllArgsConstructor;
```

```
import lombok.Getter;
```

```
import lombok.NoArgsConstructor;
```

```
import lombok.Setter;
```

```
import lombok.ToString;
```

```
@Setter
```

```
@Getter
```

```
@NoArgsConstructor
```

```
@AllArgsConstructor
```

```
@ToString
```

```
public class Produto {
```

```
    private Integer idProduto;
```

```
    private String nome;
```

```
    private Double preco;
```

```
    private Integer quantidade;
```

```
}
```

Executando:

```
package principal;
```

```
import entities.Produto;
```

```
public class Program {
```

```
public static void main(String[] args) {  
  
    Produto produto = new Produto();  
  
    produto.setIdProduto(1);  
    produto.setNome("Notebook Dell");  
    produto.setPreco(5000.0);  
    produto.setQuantidade(10);  
  
    System.out.println("\nDADOS DO PRODUTO");  
    System.out.println(produto.toString());  
}  
}
```

