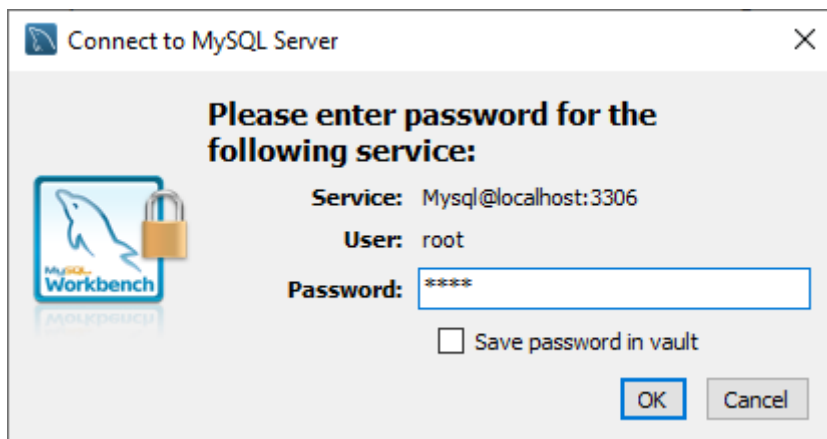


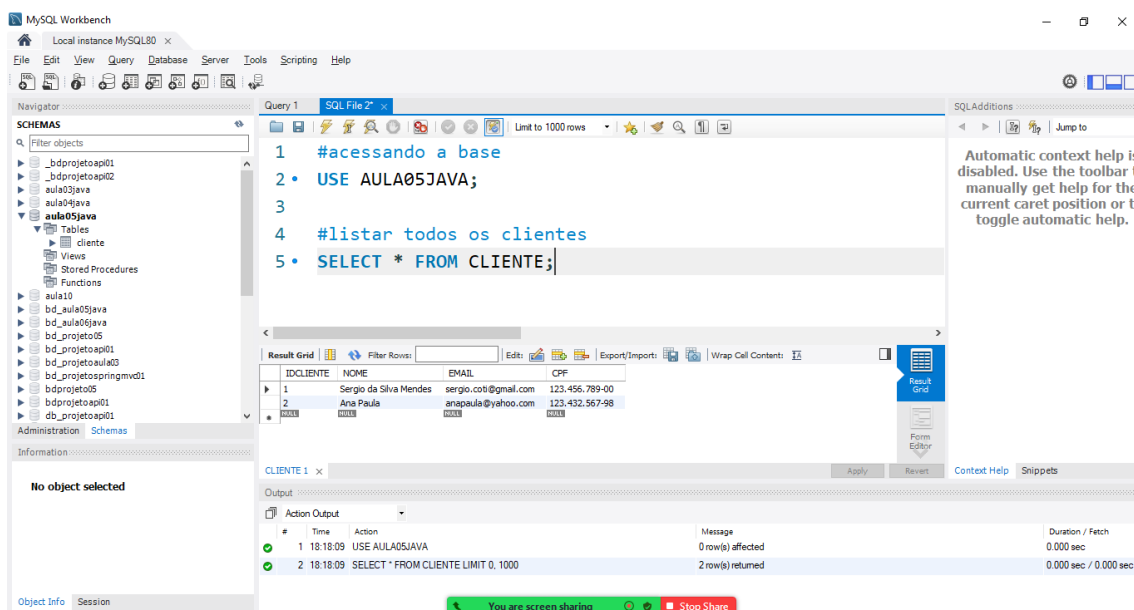
Abrindo o MYSQL:

Acessando a base de dados.



#acessando a base
USE AULA05JAVA;

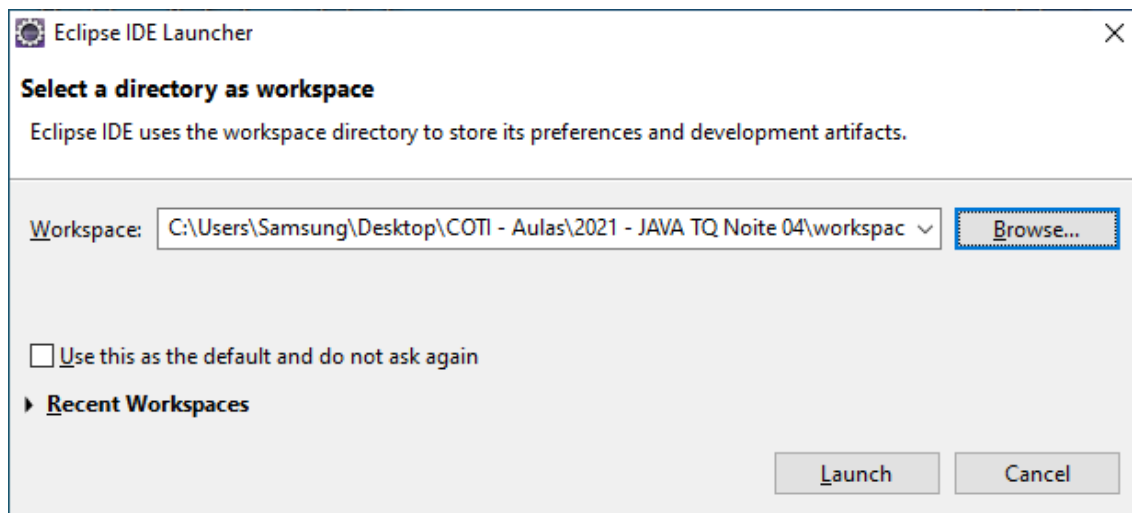
#listar todos os clientes
SELECT * FROM CLIENTE;



IDCLIENTE	NOME	EMAIL	CPF
1	Sergio da Silva Mendes	sergio.coti@gmail.com	123.456.789-00
2	Ana Paula	anapaula@yahoo.com	123.432.567-98
NULL	NULL	NULL	NULL

Abrindo o projeto Java:

ECLIPSE IDE:



/repositories/**ClienteRepository.java**

Classe para implementar os métodos de acesso a banco de dados para a entidade Cliente.

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.

```
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;
```

```
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 {

    Connection connection = ConnectionFactory.getConnection();

    PreparedStatement statement = connection.prepareStatement
        ("SELECT * FROM CLIENTE");

    //Executando a consulta e capturando os registros obtidos
    //atraves do componente ResultSet
    ResultSet result = statement.executeQuery();

    List<Cliente> lista = new ArrayList<Cliente>();

    //percorrer cada cliente obtido do banco de dados
    while(result.next()) {

        //criando um cliente..
        Cliente cliente = new Cliente();

        cliente.setIdCliente(result.getInt("IDCLIENTE"));
        cliente.setNome(result.getString("NOME"));
        cliente.setEmail(result.getString("EMAIL"));
        cliente.setCpf(result.getString("CPF"));

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

    result.close();
    statement.close();
    connection.close();

    return lista; //retornando a lista de clientes
}
}
```

Voltando na classe **Program.java**

Criando um menu de opções no projeto.

```
package principal;

import java.util.Scanner;

import entities.Cliente;
import inputs.ClienteInput;
import repositories.ClienteRepository;

public class Program {

    public static void main(String[] args) {

        try {
```

```
@SuppressWarnings("resource")
Scanner scanner = new Scanner(System.in);

System.out.println("\n *** SISTEMA DE CONTROLE
DE CLIENTES *** \n");

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("\nEscolha uma opção: ");
Integer opcao = Integer.parseInt(scanner.nextLine());

if(opcao == 1) {

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

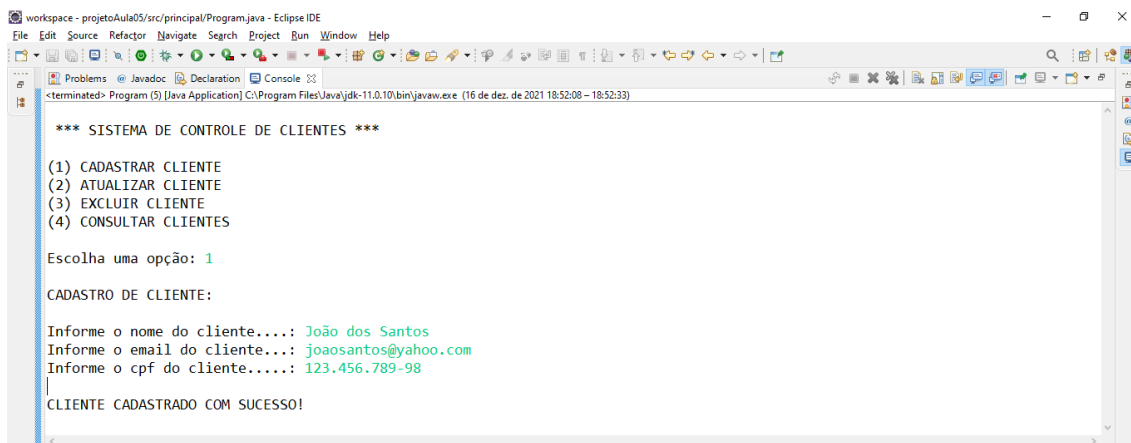
    Cliente cliente = new Cliente();

    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());
}
}
}
```



```
workspace - projetoAula05/src/principal/Program.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
[Problems] [Javadoc] [Declaration] [Console]
<terminated> Program (5) [Java Application] C:\Program Files\Java\jdk-11.0.10\bin\javaw.exe (16 de dez. de 2021 18:52:08 - 18:52:33)

*** SISTEMA DE CONTROLE DE CLIENTES ***

(1) CADASTRAR CLIENTE
(2) ATUALIZAR CLIENTE
(3) EXCLUIR CLIENTE
(4) CONSULTAR CLIENTES

Escolha uma opção: 1

CADASTRO DE CLIENTE:

Informe o nome do cliente....: João dos Santos
Informe o email do cliente....: joaosantos@yahoo.com
Informe o cpf do cliente.....: 123.456.789-98

CLIENTE CADASTRADO COM SUCESSO!
```

*** SISTEMA DE CONTROLE DE CLIENTES ***

- (1) CADASTRAR CLIENTE
- (2) ATUALIZAR CLIENTE
- (3) EXCLUIR CLIENTE
- (4) CONSULTAR CLIENTES

Escolha uma opção: 1

CADASTRO DE CLIENTE:

Informe o nome do cliente.....: João dos Santos
Informe o email do cliente...: joaosantos@yahoo.com
Informe o cpf do cliente.....: 123.456.789-98

CLIENTE CADASTRADO COM SUCESSO!

Desenvolvendo as demais operações:

throw new Exception

Comando que faz com que o fluxo de um método seja automaticamente redirecionado para o bloco catch (lança uma exceção). Sempre que este comando é executado o programa lança uma exceção e procura o primeiro bloco catch mais próximo.

```
package principal;
```

```
import java.util.Scanner;
```

```
import entities.Cliente;
```

```
import inputs.ClienteInput;
```

```
import repositories.ClienteRepository;
```

```
public class Program {
```

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

```
        try {
```

```
            @SuppressWarnings("resource")
```

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

```
            System.out.println("\n *** SISTEMA DE CONTROLE  
                                DE CLIENTES *** \n");
```

```
            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("\nEscolha uma opção: ");
```

```
            Integer opcao = Integer.parseInt(scanner.nextLine());
```

```
        if(opcao == 1) {

            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!");
        }
        else if(opcao == 2) {

            System.out.println("\nATUALIZAÇÃO DE CLIENTE:\n");
            //TODO
        }
        else if(opcao == 3) {

            System.out.println("\nEXCLUSÃO DE CLIENTE:\n");
            //TODO
        }
        else if(opcao == 4) {

            System.out.println("\nCONSULTA DE CLIENTES:\n");
            //TODO
        }
        else {
            //lançar uma exceção (redirecionar para o catch!)
            throw new Exception("Opção inválida.");
        }
    }
    catch (Exception e) {
        // imprimir mensagem de erro
        System.out.println("\nErro: " + e.getMessage());
    }
}
```

*** SISTEMA DE CONTROLE DE CLIENTES ***

- (1) CADASTRAR CLIENTE
- (2) ATUALIZAR CLIENTE
- (3) EXCLUIR CLIENTE
- (4) CONSULTAR CLIENTES

Escolha uma opção: 5
Erro: Opção inválida.

Desenvolvendo as demais operações:

```
package principal;

import java.util.List;
import java.util.Scanner;

import entities.Cliente;
import inputs.ClienteInput;
import repositories.ClienteRepository;

public class Program {

    public static void main(String[] args) {

        try {

            @SuppressWarnings("resource")
            Scanner scanner = new Scanner(System.in);

            System.out.println("\n *** SISTEMA DE CONTROLE
                                DE CLIENTES *** \n");

            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("\nEscolha uma opção: ");
            Integer opcao = Integer.parseInt(scanner.nextLine());

            if(opcao == 1) {

                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!");
            }
            else if(opcao == 2) {
```



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

        Cliente cliente = new Cliente();
        cliente.setIdCliente(ClienteInput.lerIdCliente());
        cliente.setNome(ClienteInput.lerNome());
        cliente.setEmail(ClienteInput.lerEmail());
        cliente.setCpf(ClienteInput.lerCpf());

        //atualizando no banco de dados
        ClienteRepository clienteRepository
            = new ClienteRepository();
        clienteRepository.update(cliente);

        System.out.println("\nCLIENTE ATUALIZADO
                            COM SUCESSO!");
    }
    else if(opcao == 3) {

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

        Cliente cliente = new Cliente();

        cliente.setIdCliente(ClienteInput.lerIdCliente());

        //excluindo no banco de dados
        ClienteRepository clienteRepository
            = new ClienteRepository();
        clienteRepository.delete(cliente);

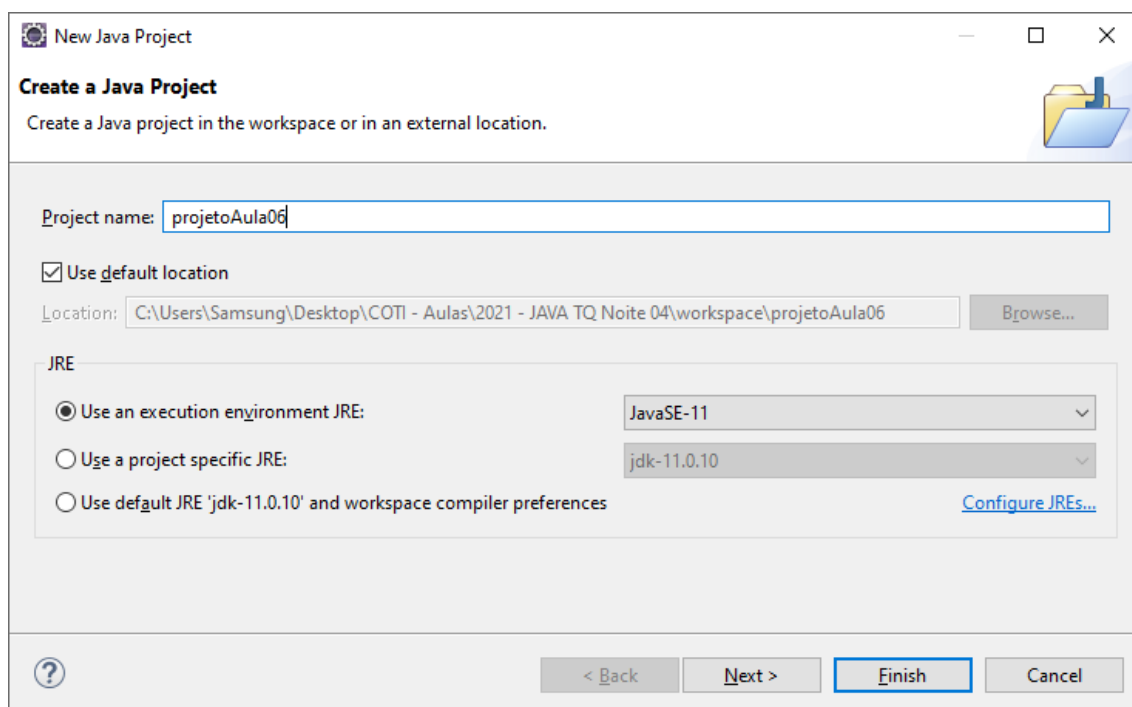
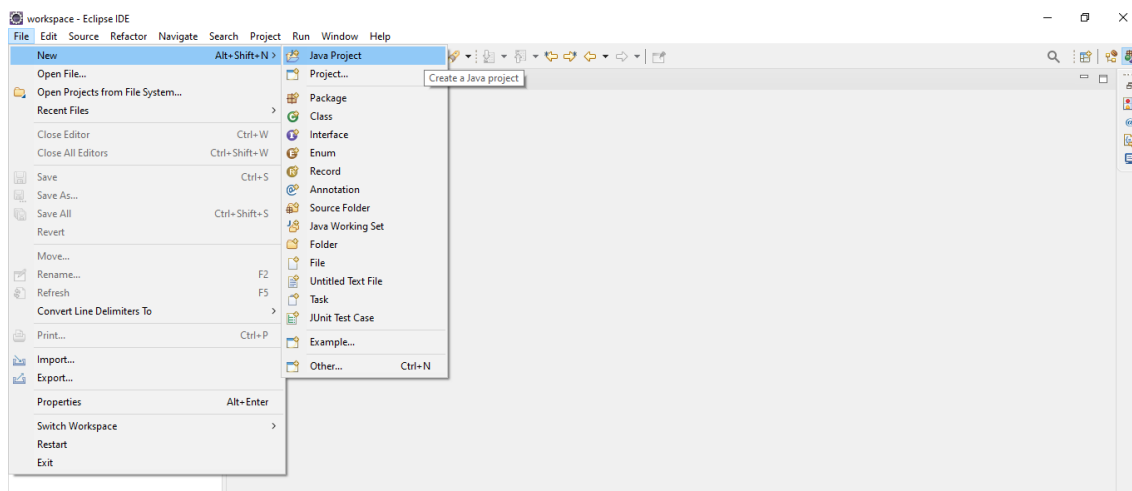
        System.out.println("\nCLIENTE EXCLUIDO COM SUCESSO!");
    }
    else if(opcao == 4) {

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

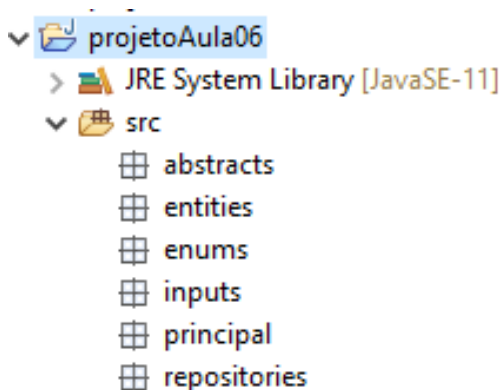
        ClienteRepository clienteRepository
            = new ClienteRepository();
        List<Cliente> lista = clienteRepository.findAll();

        for(Cliente cliente : lista) {
            System.out.println(cliente.toString());
        }
    }
    else {
        //lançar uma exceção (redirecionar para o catch!)
        throw new Exception("Opção inválida.");
    }
}
catch (Exception e) {
    // imprimir mensagem de erro
    System.out.println("\nErro: " + e.getMessage());
}
}
```

Novo projeto:

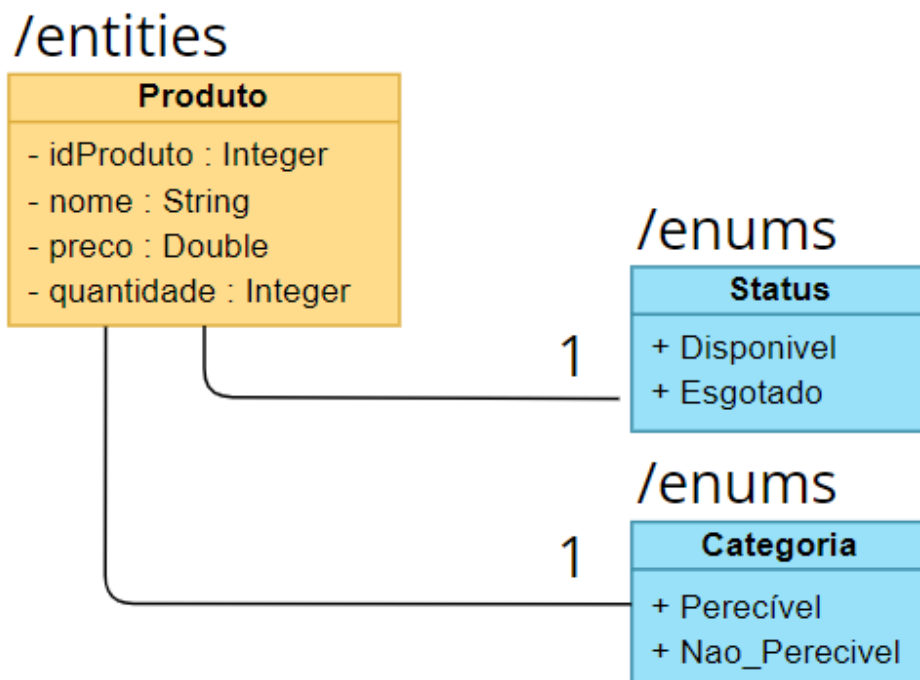


Estrutura de pacotes:



Modelo de dados:

Diagrama de Classes



```

package enums;

public enum Status {

    DISPONIVEL,
    ESGOTADO
}
  
```

```

package enums;

public enum Categoria {

    PERECIVEL,
    NAO_PERECIVEL
}
  
```

```

package entities;

import enums.Categoria;
import enums.Status;

public class Produto {

    private Integer idProduto;
    private String nome;
  
```

```
private Double preco;
private Integer quantidade;
private Status status;
private Categoria categoria;

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

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

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;
}
```

```

    public Status getStatus() {
        return status;
    }

    public void setStatus(Status status) {
        this.status = status;
    }

    public Categoria getCategoria() {
        return categoria;
    }

    public void setCategoria(Categoria categoria) {
        this.categoria = categoria;
    }

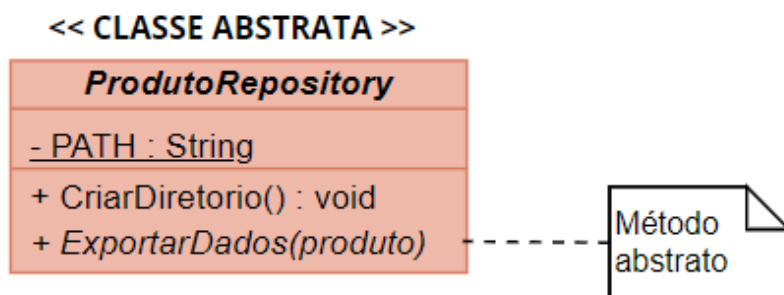
    @Override
    public String toString() {
        return "Produto [idProduto=" + idProduto + ", nome="
            + nome + ", preco=" + preco + ", quantidade="
            + quantidade + ", status=" + status + ", categoria="
            + categoria + "];"
    }
}

```

Classes Abstratas

Consiste em uma classe JAVA declarada com a palavra reservada **abstract** e que pode ter todas as características de uma classe comum, como atributos, construtores, métodos etc. mas também pode ter métodos abstratos (métodos que possuem somente assinatura) e que deverão ser implementados pelas classes que herdarem a classe abstrata.

Exemplo:



```
package abstracts;
```

```

public abstract class ProdutoRepository {

}

```

Modificadores de visibilidade:

private

Define acesso somente dentro da própria Classe. Indicado para atributos que serão posteriormente encapsulados (SETTERS e GETTERS).

public

Define acesso total para o elemento. O método público pode ser acessado por qualquer outra classe dentro do projeto.

protected

Define acesso somente dentro da própria classe (tal qual o private), porém ele abre uma exceção para herança. Ou seja, permite que subclasses tenham acesso ao método declarado como protected.

Exemplo:

```
package abstracts;
```

```
import java.io.File;
```

```
public abstract class ProdutoRepository {
```

```
    // atributo constante para definir o local onde
```

```
    // os arquivos serão armazenados
```

```
    private static final String PATH = "c:\\exportacao\\";
```

```
    // método para criar a pasta onde os arquivos serão armazenados
```

```
    // este método será definido como protected para que somente
```

```
    // as subclasses da ProdutoRepository possam acessar este método
```

```
    protected void criarDiretorio() throws Exception {
```

```
        // verificar se o diretorio não existe
```

```
        File directory = new File(PATH);
```

```
        if (!directory.exists()) {
```

```
            // criar o diretório
```

```
            directory.mkdir();
```

```
        }
```

```
    }
```

```
}
```

Declarando um método abstrato dentro da classe abstrata
Regra: As classes que herdarem a classe abstrata terão que implementar (fornecer corpo) os métodos abstratos.

Exemplo:

```
package abstracts;

import java.io.File;

import entities.Produto;

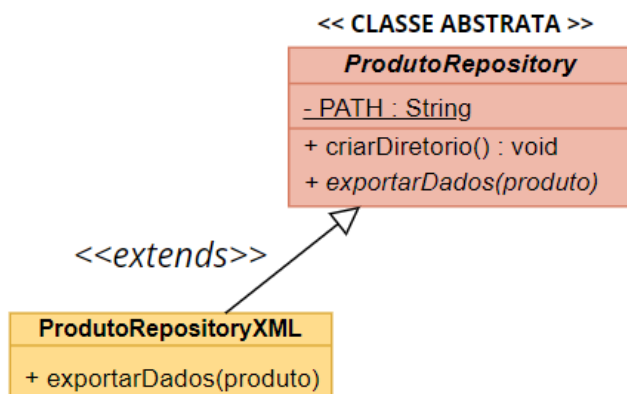
public abstract class ProdutoRepository {

    // atributo constante para definir o local onde
    // os arquivos serão armazenados
    protected static final String PATH = "c:\\exportacao\\";

    // método para criar a pasta onde os arquivos serão armazenados
    // este método será definido como protected para que somente
    // as subclasses da ProdutoRepository possam acessar este método
    protected void criarDiretorio() throws Exception {
        // verificar se o diretorio não existe
        File directory = new File(PATH);
        if (!directory.exists()) {
            // criar o diretório
            directory.mkdir();
        }
    }

    //método abstrato
    public abstract void exportarDados(Produto produto)
    throws Exception;
}
```

Implementando a classe abstrata para exportar os dados do produto em formato XML:



/repositories/**ProdutoRepositoryXML.java**

```
package repositories;

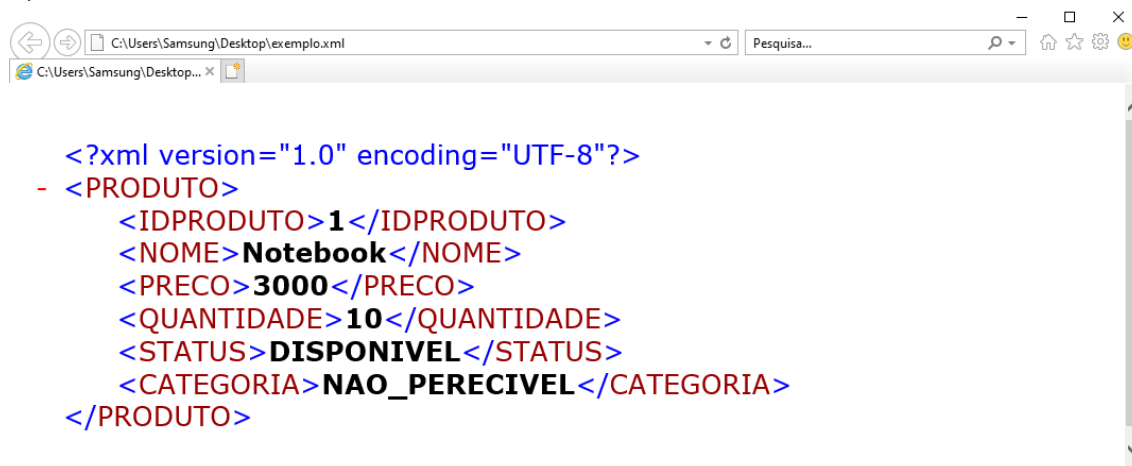
import abstracts.ProdutoRepository;
import entities.Produto;

public class ProdutoRepositoryXML extends ProdutoRepository {

    @Override
    public void exportarDados(Produto produto) throws Exception {
        // TODO Auto-generated method stub
    }
}
```

Exemplo de documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<PRODUTO>
  <IDPRODUTO>1</IDPRODUTO>
  <NOME>Notebook</NOME>
  <PRECO>3000</PRECO>
  <QUANTIDADE>10</QUANTIDADE>
  <STATUS>DISPONIVEL</STATUS>
  <CATEGORIA>NAO_PERECIVEL</CATEGORIA>
</PRODUTO>
```



```
package repositories;

import java.io.File;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
```



```
import org.w3c.dom.Document;
import org.w3c.dom.Element;

import abstracts.ProdutoRepository;
import entities.Produto;

public class ProdutoRepositoryXML extends ProdutoRepository {

    @Override
    public void exportarDados(Produto produto) throws Exception {

        // chamando o método da classe abstrata
        criarDiretorio();

        // criando um documento XML
        DocumentBuilderFactory factory = DocumentBuilderFactory
            .newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.newDocument();

        // criando a tag raiz do XML <PRODUTO>
        Element raiz = document.createElement("PRODUTO");
        document.appendChild(raiz);

        // criando as tags onde serão impressos
        // os dados do produto
        Element idProduto = document.createElement("IDPRODUTO");
        idProduto.setTextContent(produto.getIdProduto()
            .toString());
        raiz.appendChild(idProduto);

        Element nome = document.createElement("NOME");
        nome.setTextContent(produto.getNome());
        raiz.appendChild(nome);

        Element preco = document.createElement("PRECO");
        preco.setTextContent(produto.getPreco().toString());
        raiz.appendChild(preco);

        Element quantidade = document.createElement("QUANTIDADE");
        quantidade.setTextContent(produto.getQuantidade()
            .toString());
        raiz.appendChild(quantidade);

        Element status = document.createElement("STATUS");
        status.setTextContent(produto.getStatus().toString());
        raiz.appendChild(status);

        Element categoria = document.createElement("CATEGORIA");
        status.setTextContent(produto.getCategoria().toString());
        raiz.appendChild(categoria);
    }
}
```

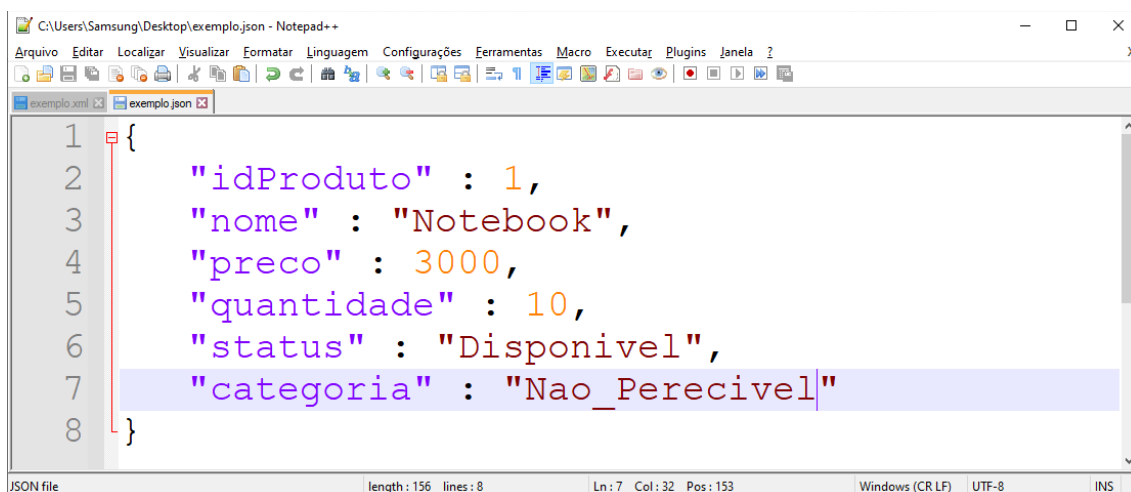
```
// finalizando e gravando o arquivo XML
TransformerFactory transformerFactory
    = TransformerFactory.newInstance();
Transformer transformer
    = transformerFactory.newTransformer();
DOMSource domSource = new DOMSource(document);

StreamResult stream = new StreamResult
    (new File(PATH + "produto.xml"));
transformer.transform(domSource, stream);
}
}
```

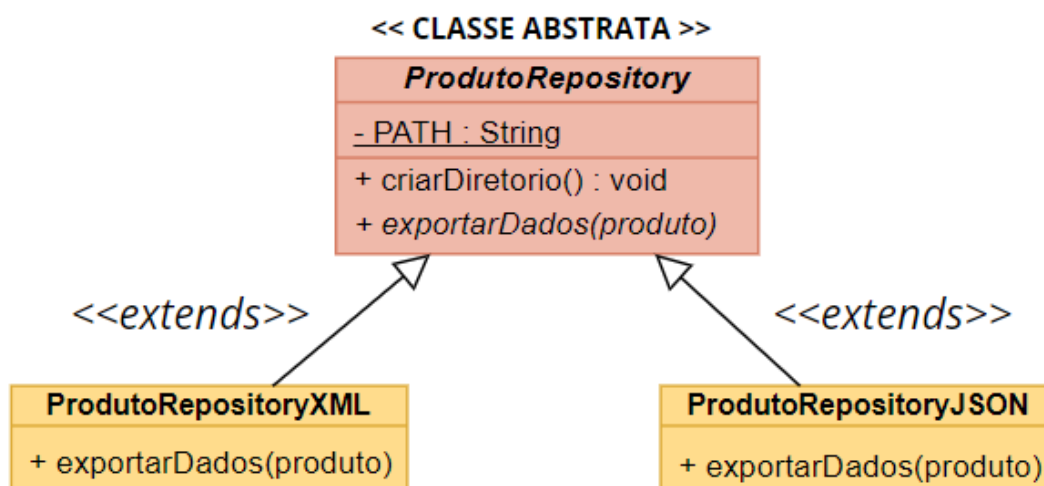
Criando uma classe para implementar a exportação dos dados para o formato JSON:

JSON – JAVASCRIPT OBJECT NOTATION

Exemplo:



```
C:\Users\Samsung\Desktop\exemplo.json - Notepad++
Arquivo Editar Localizar Visualizar Formatar Linguagem Configurações Ferramentas Macro Executar Plugins Janela ?
exemplo.xml exemplo.json
1 {
2   "idProduto" : 1,
3   "nome" : "Notebook",
4   "preco" : 3000,
5   "quantidade" : 10,
6   "status" : "Disponivel",
7   "categoria" : "Nao_Perecivel"
8 }
```



```
package repositories;
```

```
import abstracts.ProdutoRepository;
```

```
import entities.Produto;
```

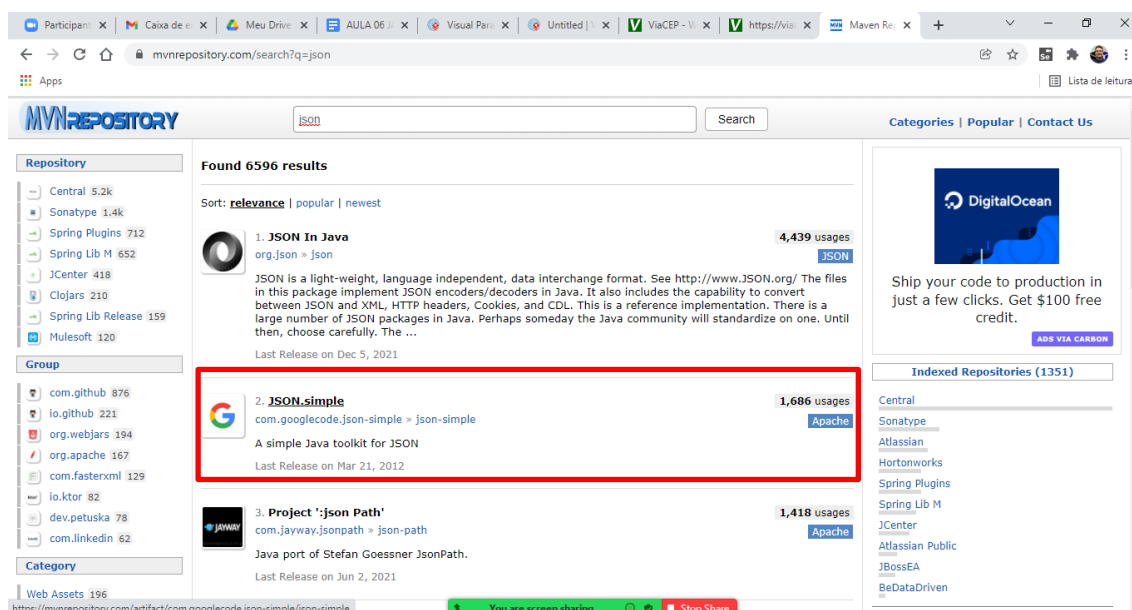
```
public class ProdutoRepositoryJSON extends ProdutoRepository {
```

```
    @Override
```

```
    public void exportarDados(Produto produto) throws Exception {
        // TODO Auto-generated method stub
    }
}
```

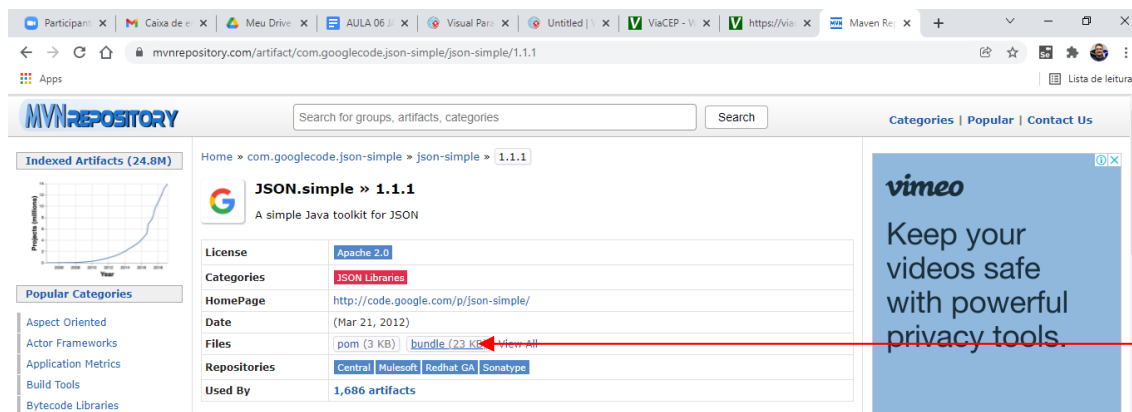
Baixando uma biblioteca (.JAR) para geração de JSON:

<https://mvnrepository.com/>



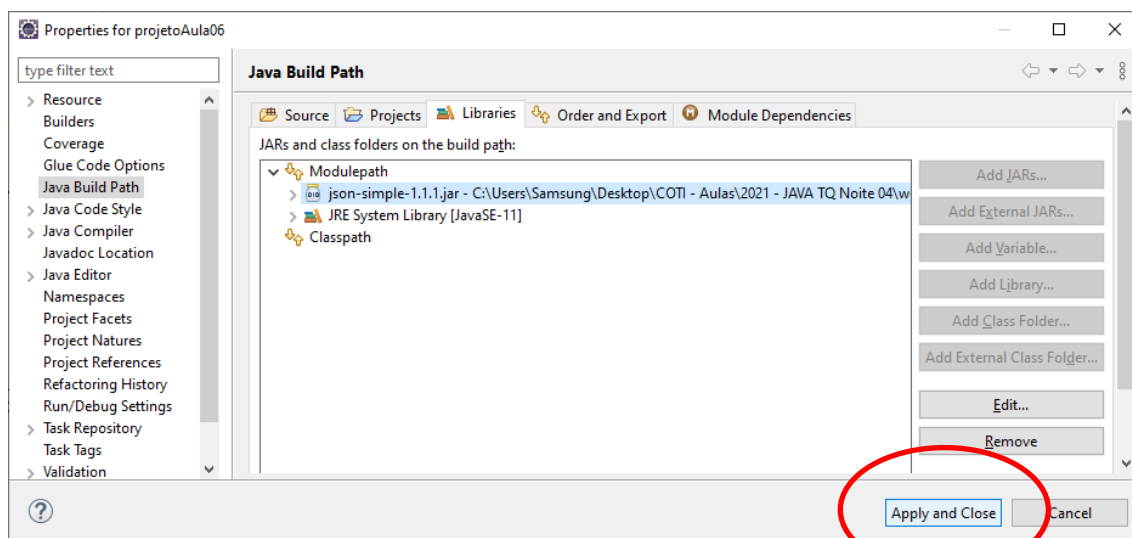
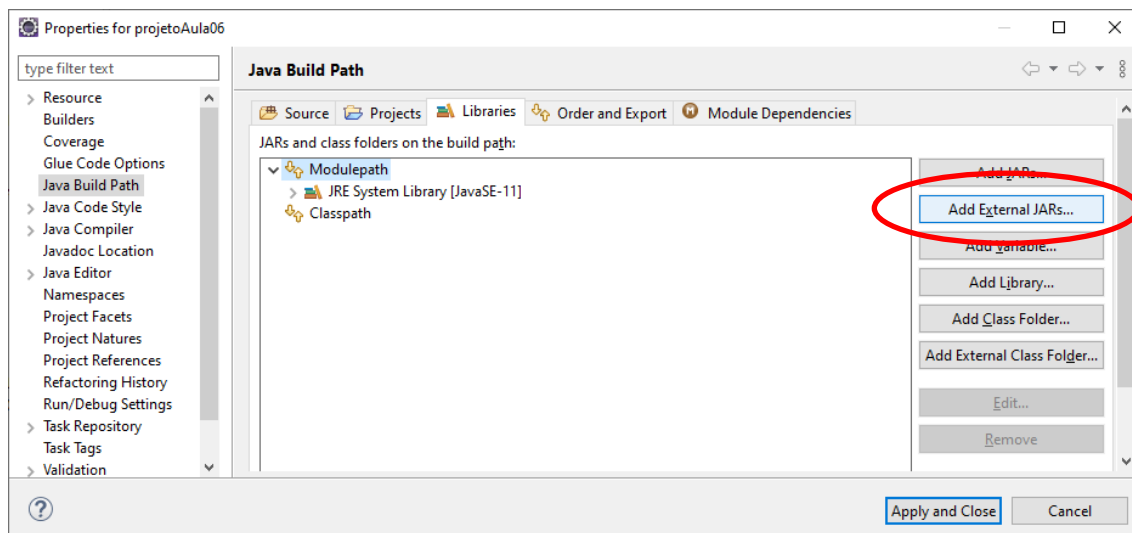
The screenshot shows the Maven Repository search results for the keyword 'json'. The search bar at the top contains 'json' and the search button is labeled 'Search'. The results are sorted by 'relevance'. The first result is 'JSON In Java' by 'org.json', which has 4,439 usages. The second result, highlighted with a red box, is 'JSON.simple' by 'com.googlecode.json-simple', which has 1,686 usages. The third result is 'Project 'json Path'' by 'com.jayway.jsonpath', which has 1,418 usages. The left sidebar shows a list of repositories and groups. The right sidebar shows a list of indexed repositories.

<https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple/1.1.1>



The screenshot shows the Maven Repository artifact page for 'com.googlecode.json-simple/json-simple/1.1.1'. The page title is 'JSON.simple » 1.1.1'. The description is 'A simple Java toolkit for JSON'. The license is 'Apache 2.0'. The categories are 'JSON Libraries'. The homepage is 'http://code.google.com/p/json-simple/'. The date is '(Mar 21, 2012)'. The files section shows 'pom (3 KB)' and 'bundle (23 KB)'. The repositories section shows 'Central', 'Mulesoft', 'Redhat GA', and 'Sonatype'. The used by section shows '1,686 artifacts'. The left sidebar shows a list of indexed artifacts and popular categories. The right sidebar shows a Vimeo advertisement.

Adicionando referência para a biblioteca no projeto: BUILD PATH / CONFIGURE BUILD PATH



package repositories;

```
import java.io.FileWriter;
```

```
import java.util.HashMap;
```

```
import org.json.simple.JSONObject;
```

```
import abstracts.ProdutoRepository;
```

```
import entities.Produto;
```

```
public class ProdutoRepositoryJSON extends ProdutoRepository {
```

```
@Override
public void exportarDados(Produto produto) throws Exception {

    HashMap<String, Object> mapa
        = new HashMap<String, Object>();

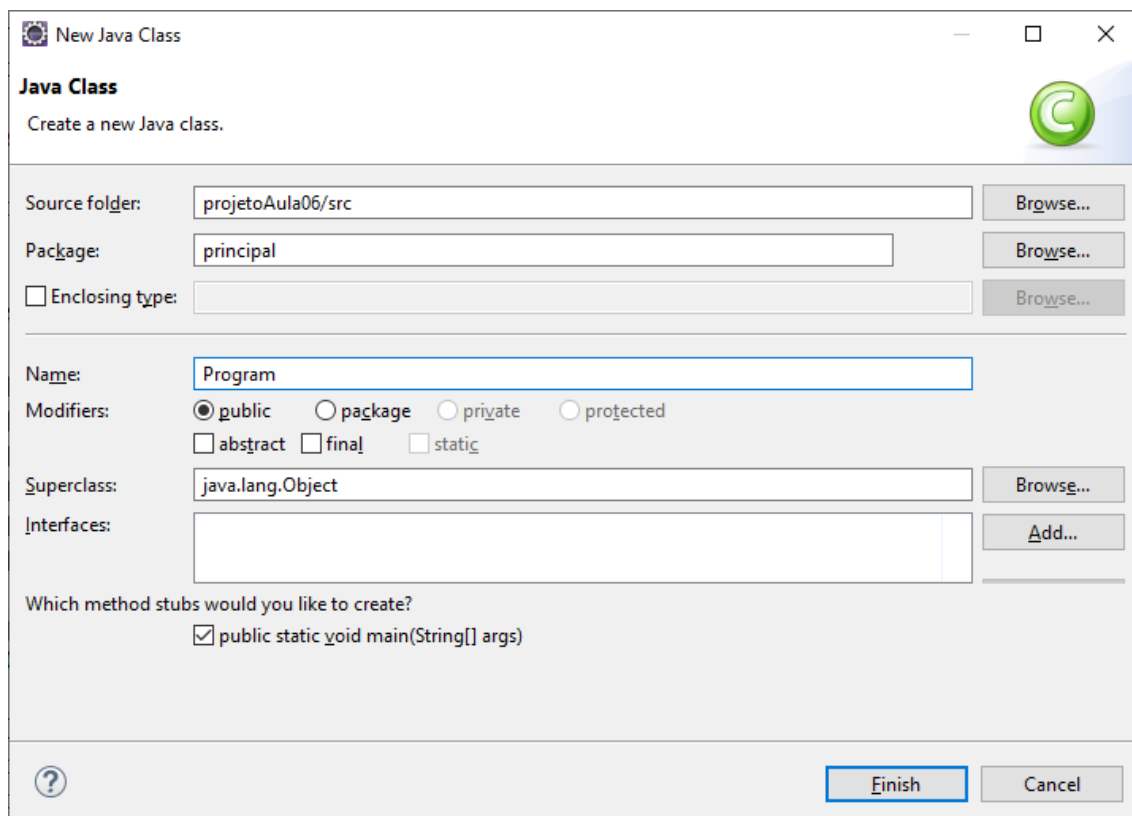
    mapa.put("idProduto", produto.getIdProduto());
    mapa.put("nome", produto.getNome());
    mapa.put("preco", produto.getPreco());
    mapa.put("quantidade", produto.getQuantidade());
    mapa.put("status", produto.getStatus().toString());
    mapa.put("categoria", produto.getCategoria().toString());

    //gerando o conteudo JSON para gravar no arquivo
    JSONObject json = new JSONObject(mapa);

    FileWriter writer = new FileWriter(PATH + "produto.json");
    writer.write(json.toString());
    writer.flush();
    writer.close();
}
```

/principal/Program.java

Testando



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)

```
package principal;

import entities.Produto;
import enums.Categoria;
import enums.Status;
import repositories.ProdutoRepositoryJSON;
import repositories.ProdutoRepositoryXML;

public class Program {

    public static void main(String[] args) {

        try {

            Produto produto = new Produto(1, "Notebook", 3000.0,
            10, Status.DISPONIVEL, Categoria.NAO_PERECIVEL);

            ProdutoRepositoryXML xml
                = new ProdutoRepositoryXML();

            xml.exportarDados(produto);

            ProdutoRepositoryJSON json
                = new ProdutoRepositoryJSON();

            json.exportarDados(produto);

            System.out.println("\nARQUIVOS GERADOS
                                COM SUCESSO!");

        } catch (Exception e) {

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

        }

    }

}
```

