

Criando uma estrutura de pacotes:

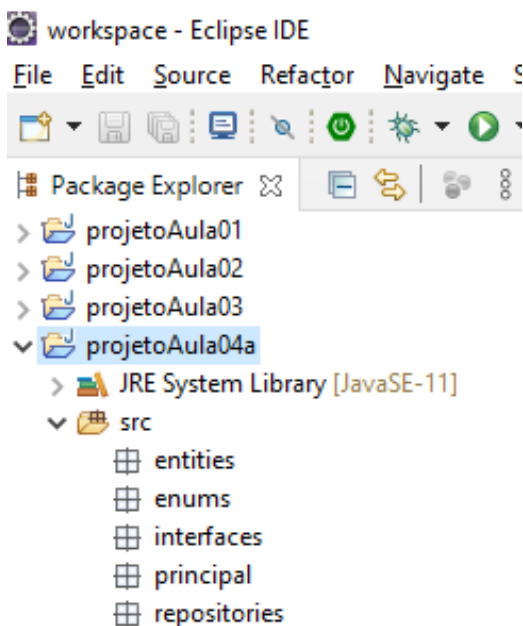
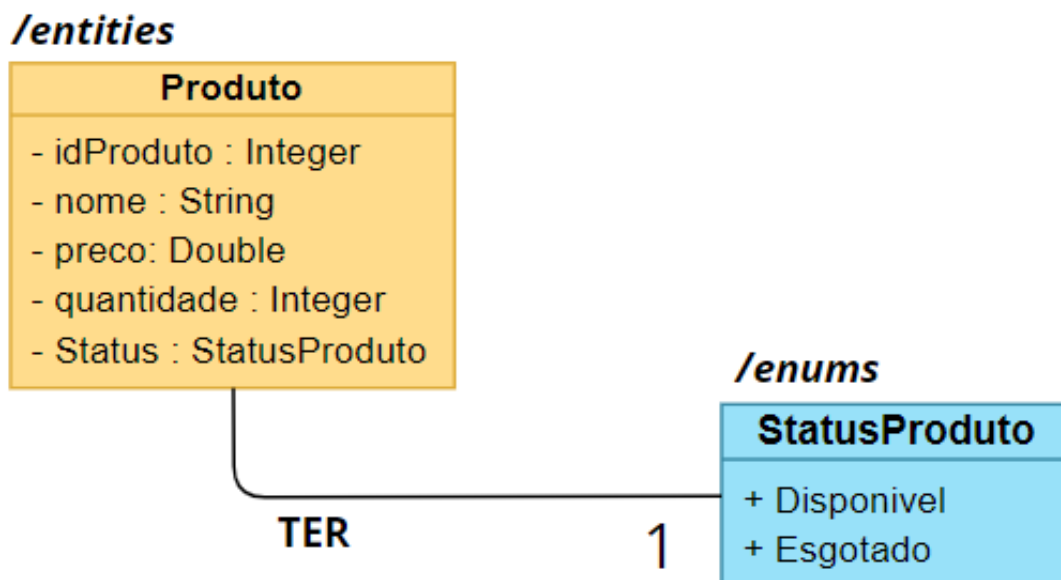


Diagrama de modelo de entidades: Padrão JAVABEN



ENUMS

São classes Java utilizados para definir campos multivalorados, ou seja, campos que já possuem valores pré-definidos. No exemplo abaixo teremos um Produto que pode ter Status Disponível ou Status Esgotado, de acordo com as opções criadas no ENUM.

/enums/**StatusProduto.java**

```

package enums;

public enum StatusProduto {
    Disponivel,
    Esgotado
}
  
```

Criando a classe JAVABEAN para Produto:

Características:

- Atributos privados
- Construtor sem argumentos
- Construtor com entrada de argumentos
- Métodos de encapsulamento set e get
- Sobrescrita de métodos da classe Object
 - toString()

/entities/**Produto.java**

```

package entities;

import enums.StatusProduto;
  
```

```
public class Produto {

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

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

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

    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;
    }

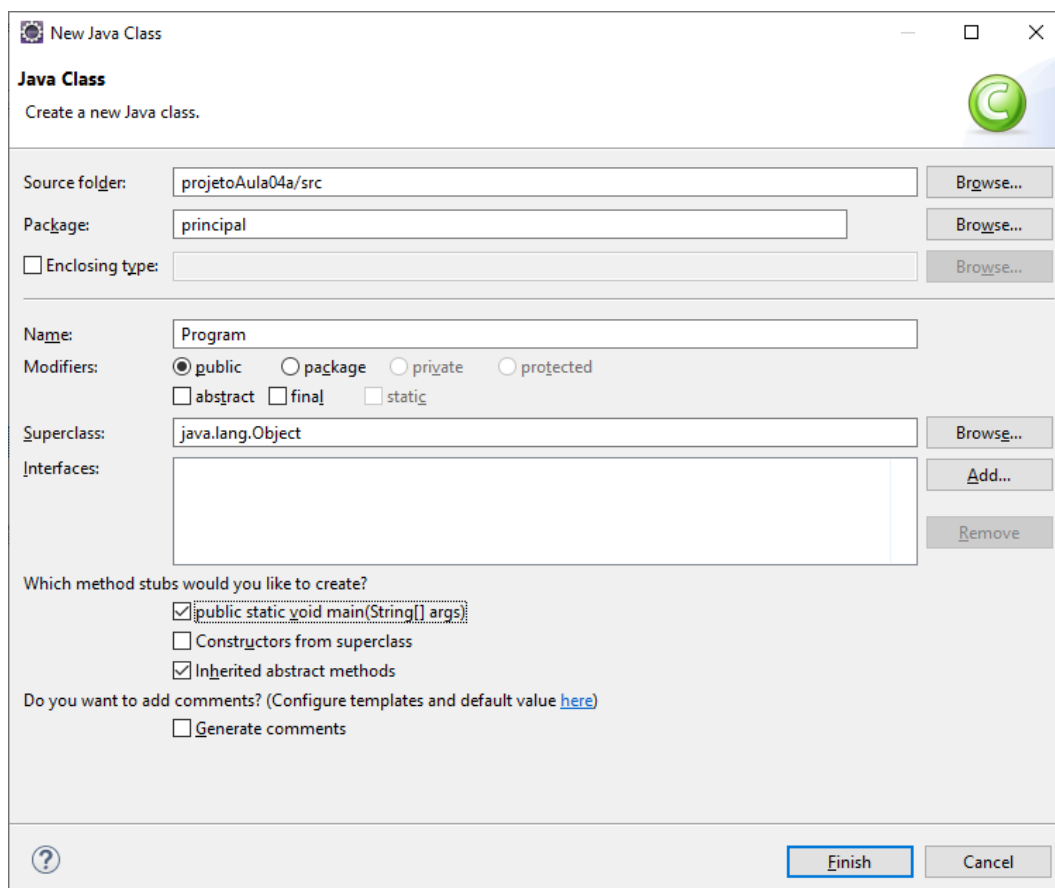
    public StatusProduto getStatus() {
        return status;
    }

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

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

```

Criando uma classe para executar o projeto:
/principal/**Program.java**



New Java Class

Java Class
Create a new Java class.

Source folder: projetoAula04a/src Browse...

Package: principal Browse...

☐ Enclosing type: Browse...

Name: Program

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

Superclass: java.lang.Object 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

JOptionPane

Componente utilizado em desenvolvimento Java Desktop para criação de interfaces de janelas para exibir mensagens ou capturar dados do usuário.

```
package principal;

import javax.swing.JOptionPane;

import entities.Produto;

public class Program {

    public static void main(String[] args) {

        try {
            System.out.println("\n*** CADASTRO DE PRODUTO ***\n");
            //criando uma variável de instância para Produto

            Produto produto = new Produto();

            produto.setIdProduto(Integer.parseInt
                (JOptionPane.showInputDialog("Entre com
                    o ID do produto:")));
            produto.setNome(JOptionPane.showInputDialog
                ("Entre com o nome do produto:"));
            produto.setPreco(Double.parseDouble
                (JOptionPane.showInputDialog("Entre com o preço
                    do produto:")));
            produto.setQuantidade(Integer.parseInt
                (JOptionPane.showInputDialog("Entre com a
                    quantidade do produto:")));
            //imprimir os dados do produto:
            System.out.println(produto.toString());
        }
        catch(Exception e) {
            System.out.println("\nErro: " + e.getMessage());
        }
    }
}
```

Testando:

Input

Entre com o ID do produto:

OK Cancel

Input

Entre com o nome do produto:

OK Cancel

Input

Entre com o preço do produto:

OK Cancel

Input

Entre com a quantidade do produto:

OK Cancel

```
workspace - projetoAula04/src/principal/Program.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
  projetoAula01
  projetoAula02
  projetoAula03
  projetoAula04
    JRE System Library [JavaSE-11]
    src
      entities
        Produto.java
      enums
        StatusProduto.java
      interfaces
        principal
          Program.java
      repositories
StatusProduto.java
4
5 import enti
6
7 public clas
8
9     public
10
11     try
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
*** CADASTRO DE PRODUTO ***
Produto [idProduto=1, nome=Notebook DELL, preco=4000.0, quantidade=10, status=null]

System.out.println("\n*** CADASTRO DE PRODUTO ***\n");

//criando uma variável de instância para Produto
Produto produto = new Produto();

produto.setIdProduto(Integer.parseInt(JOptionPane.showInputDialog("Entre com o ID do produ
produto.setNome(JOptionPane.showInputDialog("Entre com o nome do produto:"));
produto.setPreco(Double.parseDouble(JOptionPane.showInputDialog("Entre com o preço do prod
produto.setQuantidade(Integer.parseInt(JOptionPane.showInputDialog("Entre com a quantidade

//imprimir os dados do produto:
System.out.println(produto.toString());

catch(Exception e) {
    System.out.println("\nErro: " + e.getMessage());
}
```

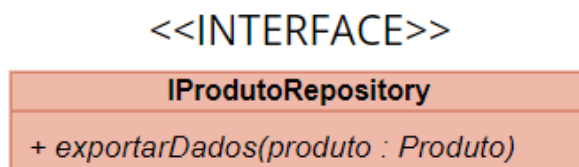
Interfaces

Artefato de programação orientado a objetos onde só podemos declarar **métodos abstratos**, ou seja, métodos que só possuem assinatura e não possuem corpo (conteúdo).

Em uma interface nós podemos definir quais métodos uma classe deverá implementar. Quando uma classe HERDAR uma interface, a classe será obrigada a implementar todos os métodos abstratos definidos na interface.

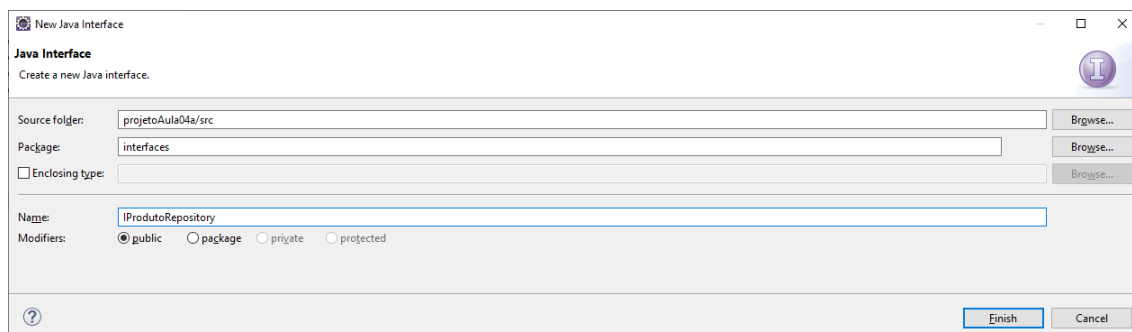
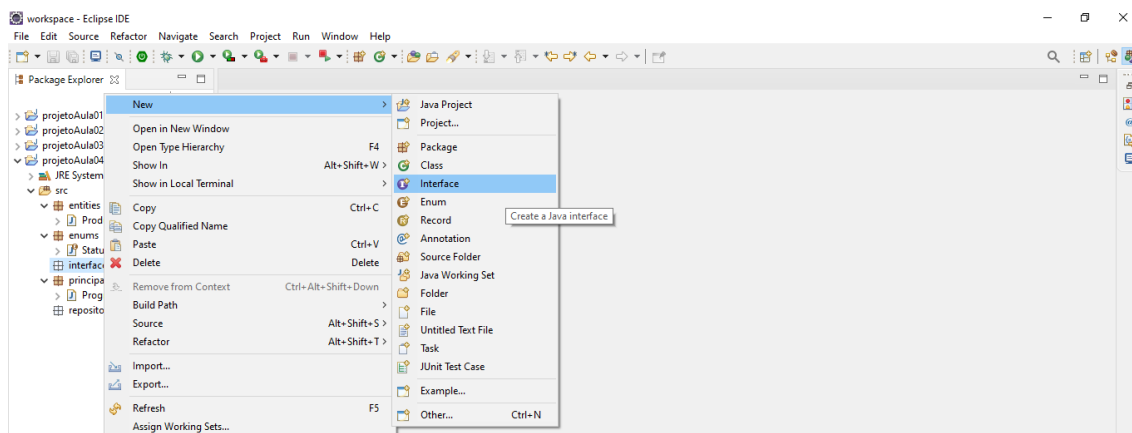
** Não é regra, mas nomes de interfaces sempre podem ser declarados começando com a letra **I**.

Neste exemplo iremos criar uma interface denominada **IProdutoRepository** onde iremos declarar um método abstrato chamado **exportarDados**:



/interfaces/IProdutoRepository.java

Criando a primeira interface:



Método abstratos:

São métodos que não possuem corpo, apenas assinatura (declaração). Estes métodos podem ser declarados em uma interface para que as classes que HERDAREM a interface sejam obrigadas a implementar estes métodos.

Exemplo:

```
package interfaces;

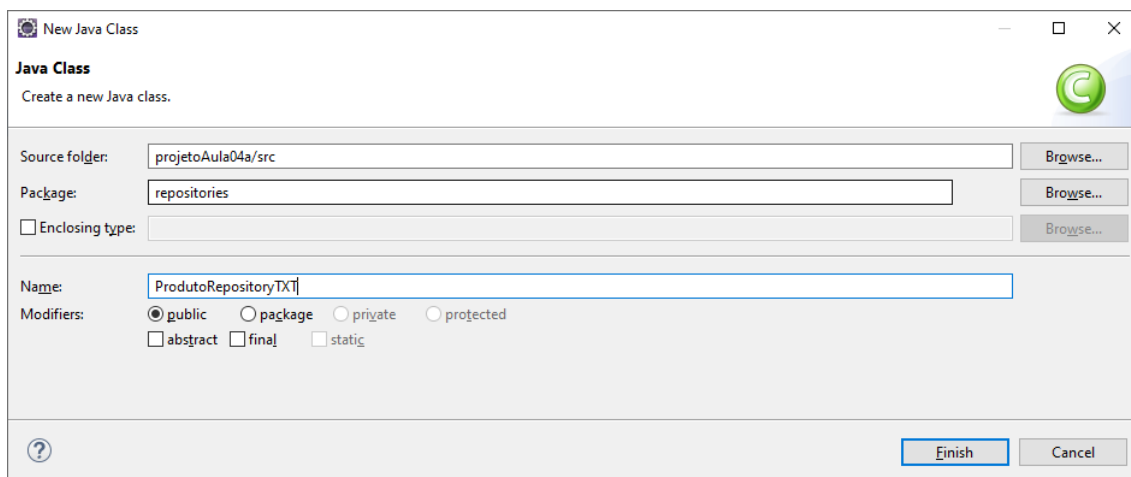
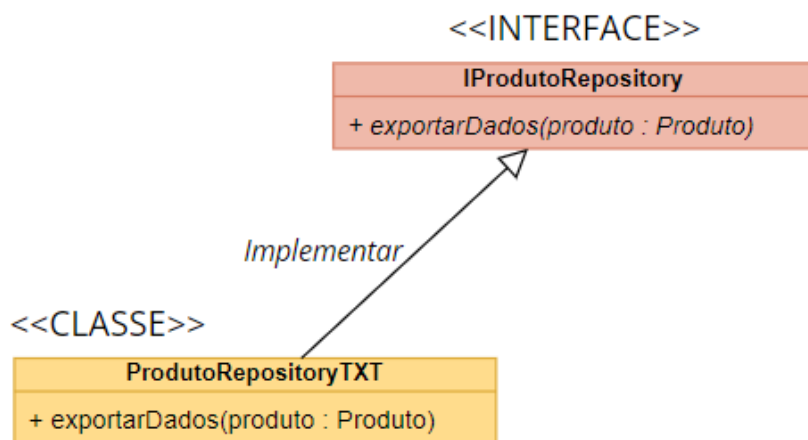
import entities.Produto;

public interface IProdutoRepository {

    //métodos abstratos (somente a declaração)
    void exportarDados(Produto produto) throws Exception;
}
```

O próximo passo será criarmos classes que HERDEM e IMPLEMENTEM a interface, ou seja, que forneçam corpo para todos os métodos abstratos da interface.

Exemplo:




```
package repositories;

import entities.Produto;
import interfaces.IProdutoRepository;

public class ProdutoRepositoryTXT implements IProdutoRepository {

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

        // TODO Auto-generated method stub

    }
}
```

Note que a classe acima está **IMPLEMENTANDO (implements)** a interface e por conta disso programando o método **exportarDados()**.

```
package repositories;

import java.io.File;
import java.io.PrintWriter;

import entities.Produto;
import interfaces.IProdutoRepository;

public class ProdutoRepositoryTXT implements IProdutoRepository {

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

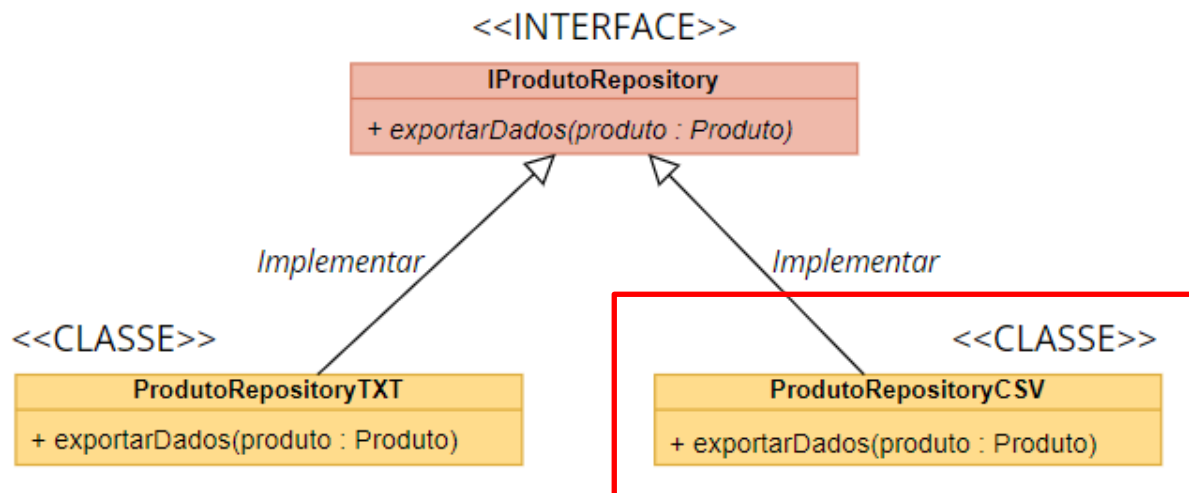
        PrintWriter printWriter = new PrintWriter
            (new File("c:\\temp\\produto.txt"));

        printWriter.write("\n *** DADOS DO PRODUTO *** \n");
        printWriter.write("\nID DO PRODUTO.....: "
            + produto.getIdProduto());
        printWriter.write("\nNOME.....: "
            + produto.getNome());
        printWriter.write("\nPREÇO.....: "
            + produto.getPreco());
        printWriter.write("\nQUANTIDADE.....: "
            + produto.getQuantidade());
        printWriter.write("\nSTATUS.....: "
            + produto.getStatus());

        printWriter.flush();
        printWriter.close();

    }
}
```

Criando uma outra classe também **IMPLEMENTANDO** a interface só que fazendo a gravação dos dados do produto em arquivo **CSV**.



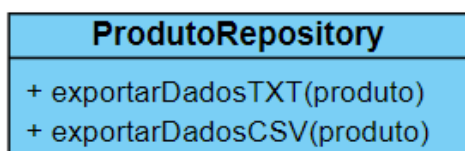
Voltando ao nosso estudo sobre SOLID, podemos afirmar que esta prática de criação de métodos abstratos para depois implementarmos em classes de formas diferentes está de acordo com o princípio **OCP – OPEN CLOSED PRINCIPLE**.



PRINCÍPIO DE ABERTO E FECHADO

De acordo com este princípio, toda classe deve ser **FECHADA para modificação** e **ABERTA para extensão**.

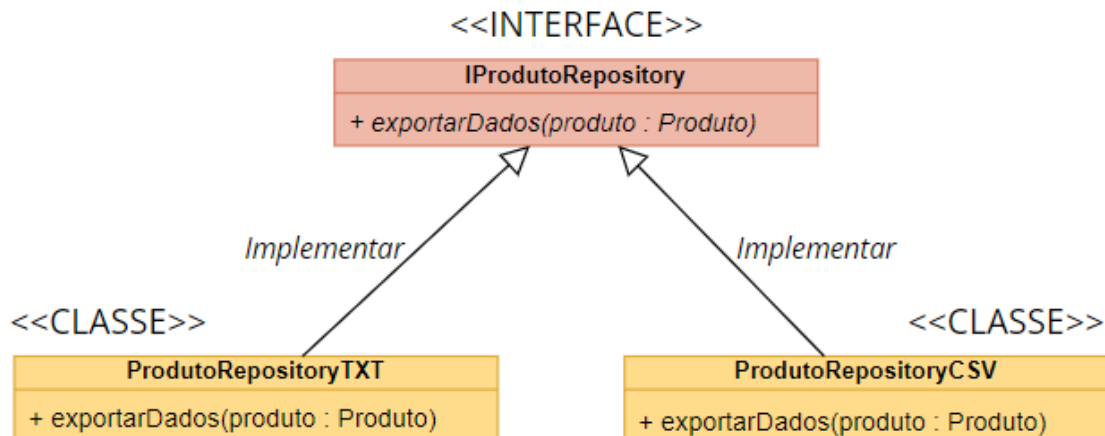
Modelo RUIM do projeto:



Na classe acima estamos criando em uma única classe todos os tipos de exportação de dados de produto possíveis (TXT, CSV, XML etc.)

Toda vez que precisarmos dar manutenção em um tipo de exportação vamos precisar mexer nesta classe.

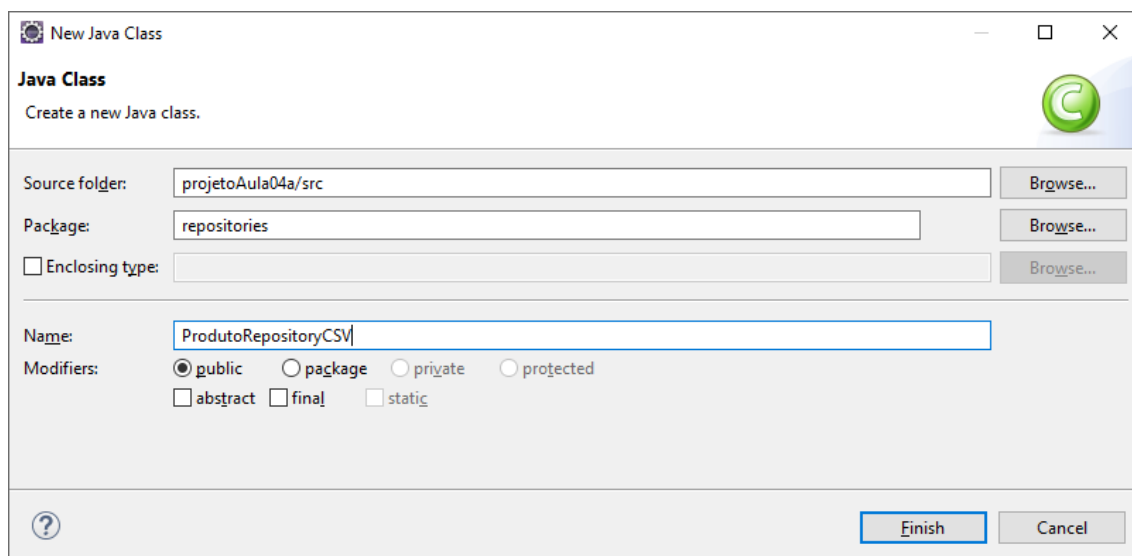
Modelo IDEAL do projeto:



No modelo acima nós criamos classes específicas para cada tipo de exportação de dados de produto. Ou seja, a interface IProdutoRepository é aberta para extensão, ou seja, é implementada de muitas formas diferentes.

/repositories/ProdutoRepositoryCSV.java

Implementando a interface para gravar arquivos de produto em formato CSV.



```
package repositories;
```

```
import entities.Produto;
```

```
import interfaces.IProdutoRepository;
```

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

Implementando a gravação do arquivo CSV:

```
package repositories;  
  
import java.io.File;  
import java.io.PrintWriter;  
  
import entities.Produto;  
import interfaces.IProdutoRepository;  
  
public class ProdutoRepositoryCSV implements IProdutoRepository {  
  
    @Override  
    public void exportarDados(Produto produto) throws Exception {  
  
        PrintWriter printWriter = new PrintWriter  
            (new File("c:\\temp\\produto.csv"));  
  
        printWriter.write("IDPRODUTO;NOME;PRECO;QUANTIDADE;STATUS\n");  
        printWriter.write(produto.getIdProduto()  
            + ";" + produto.getNome()  
            + ";" + produto.getPreco()  
            + ";" + produto.getQuantidade()  
            + ";" + produto.getStatus());  
  
        printWriter.flush();  
        printWriter.close();  
    }  
}
```

Testando na classe Program.java

```
package principal;

import javax.swing.JOptionPane;

import entities.Produto;
import enums.StatusProduto;
import repositories.ProdutoRepositoryCSV;
import repositories.ProdutoRepositoryTXT;

public class Program {

    public static void main(String[] args) {

        try {

            //criando uma variável de instância para Produto
            Produto produto = new Produto();

            produto.setIdProduto(Integer.parseInt
                (JOptionPane.showInputDialog
                ("Entre com o ID do produto:")));

            produto.setNome(JOptionPane.showInputDialog
                ("Entre com o nome do produto:"));

            produto.setPreco(Double.parseDouble
                (JOptionPane.showInputDialog("Entre com o preço
                do produto:")));

            produto.setQuantidade(Integer.parseInt
                (JOptionPane.showInputDialog("Entre com
                a quantidade do produto:")));

            //definindo o status do produto (ENUM)
            if(produto.getQuantidade() > 0) {
                produto.setStatus(StatusProduto.Disponivel);
            }
            else {
                produto.setStatus(StatusProduto.Esgotado);
            }

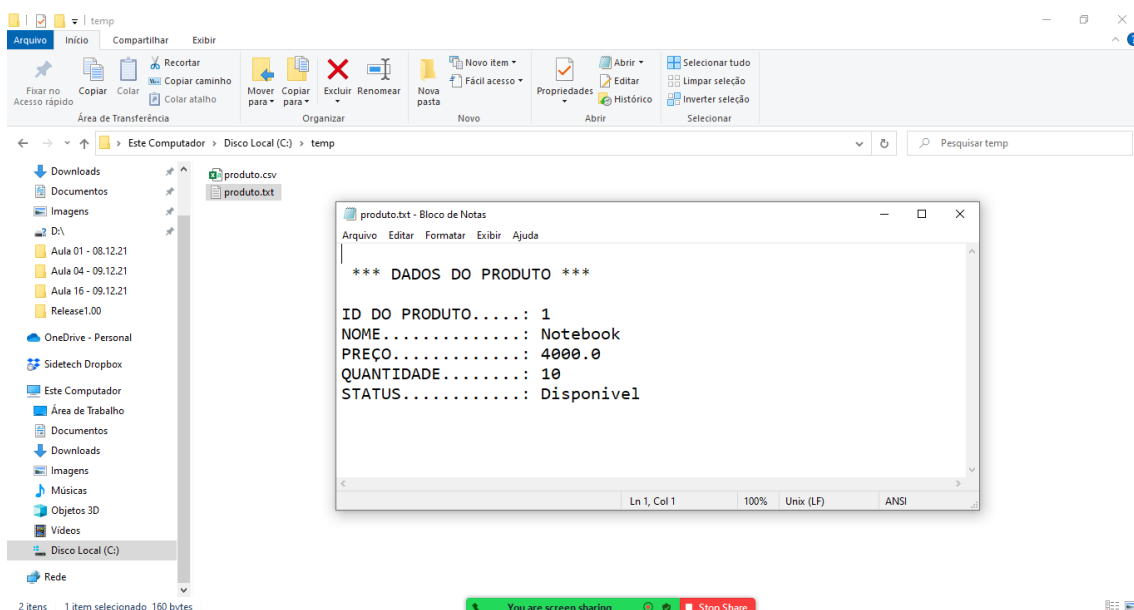
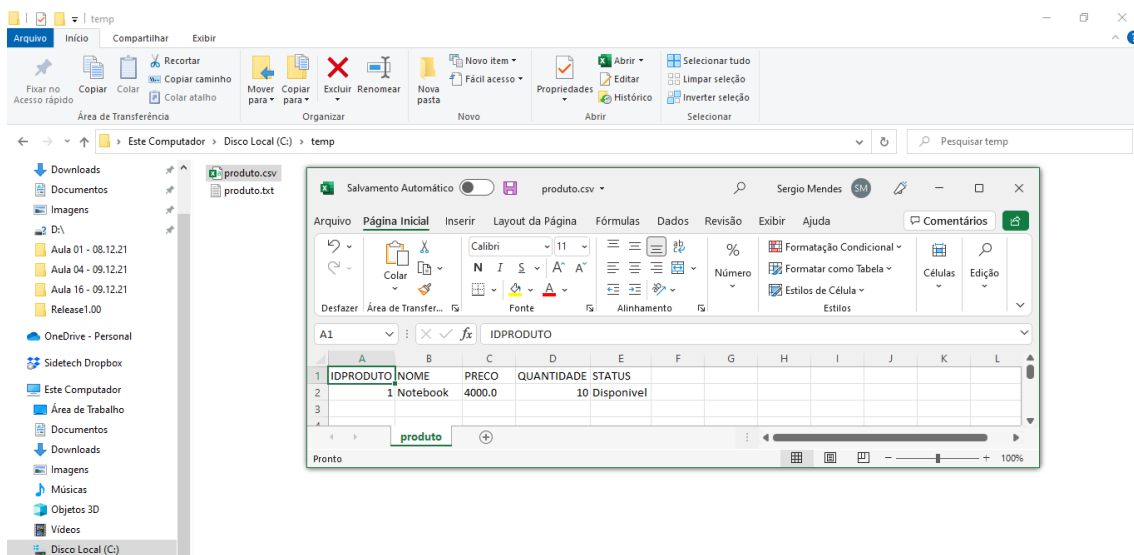
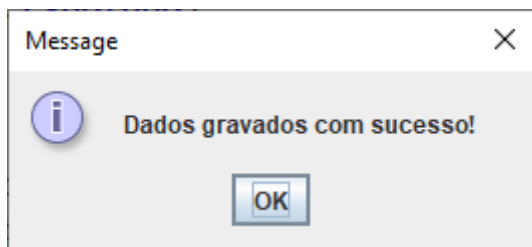
            //exportar os dados do produto para arquivo TXT
            ProdutoRepositoryTXT produtoRepositoryTXT
                = new ProdutoRepositoryTXT();
            produtoRepositoryTXT.exportarDados(produto);

            ProdutoRepositoryCSV produtoRepositoryCSV
                = new ProdutoRepositoryCSV();
            produtoRepositoryCSV.exportarDados(produto);

            //exibir mensagem de sucesso
            JOptionPane.showMessageDialog
                (null, "Dados gravados com sucesso!");
```

```
} catch (Exception e) {
    System.out.println("\nErro: " + e.getMessage());
}
}
```

Executando:



Polimorfismo

MUITAS FORMAS

Recurso de Programação Orientada a Objetos onde podemos modificar o comportamento de uma variável de instancia a partir da classe através do qual a variável é inicializada.

Exemplo:

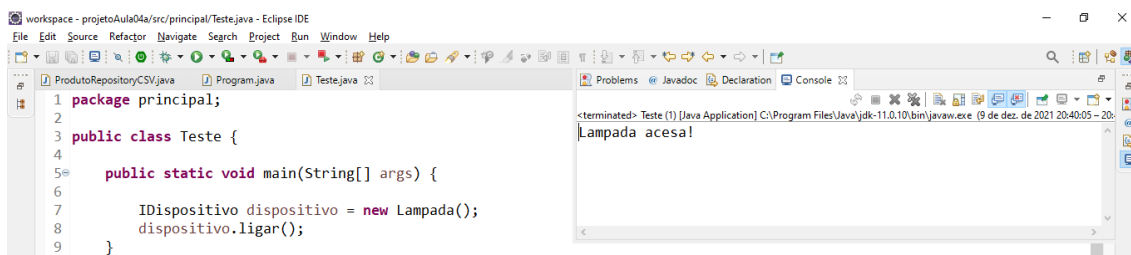
Considere o modelo de interfaces e classes abaixo:

```
interface IDispositivo {  
    void ligar(); //método abstrato  
}  
  
class Lampada implements IDispositivo {  
    public void ligar() {  
        System.out.println("Lampada acesa!");  
    }  
}  
  
class Alarme implements IDispositivo {  
    public void ligar() {  
        System.out.println("Alarme disparado!");  
    }  
}
```

Podemos criar um objeto do tipo **IDispositivo** que terá o seu comportamento modificado conforme a instância definida para ele, por exemplo:

```
package principal;  
  
public class Teste {  
  
    public static void main(String[] args) {  
        IDispositivo dispositivo = new Lampada();  
        dispositivo.ligar();  
    }  
}
```

O resultado é o seguinte:



Porém, se modificarmos a instancia da variável dispositivo podemos fazer também com que ela se comporte como Alarme, por exemplo:

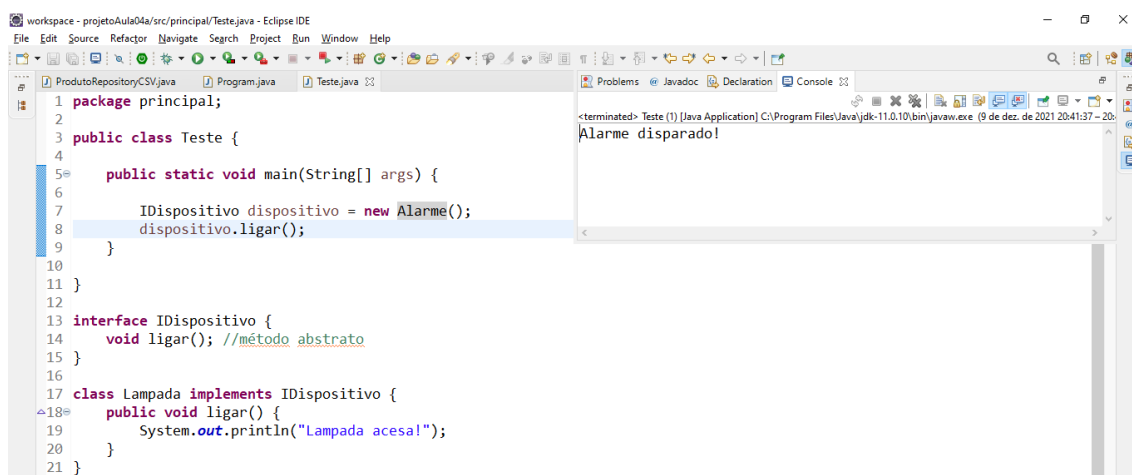
```
package principal;

public class Teste {

    public static void main(String[] args) {

        IDispositivo dispositivo = new Alarme();
        dispositivo.ligar();
    }
}
```

Resultado:



Para que possamos ter o polimorfismo, precisamos de uma abstração, ou seja de um tipo genérico que represente os demais, no exemplo acima foi a interface IDispositivo.

```
interface IDispositivo {
    void ligar(); //método abstrato
}
```

```
class Lampada implements IDispositivo {
    public void ligar() {
        System.out.println("Lampada acesa!");
    }
}
```

```
class Alarme implements IDispositivo {
    public void ligar() {
        System.out.println("Alarme disparado!");
    }
}
```


Fazendo um **polimorfismo** para que o usuário possa escolher que tipo de arquivo deseja exportar, se prefere **TEXT** ou **CSV**.

package principal;

import javax.swing.JOptionPane;

import entities.Produto;
import enums.StatusProduto;
import interfaces.IProdutoRepository;
import repositories.ProdutoRepositoryCSV;
import repositories.ProdutoRepositoryTXT;

public class Program {

 public static void main(String[] args) {

 try {

 //criando uma variável de instância para Produto
 Produto produto = new Produto();

 produto.setIdProduto(Integer.parseInt
 (JOptionPane.showInputDialog("Entre com
 o ID do produto:")));

 produto.setNome(JOptionPane.showInputDialog
 ("Entre com o nome do produto:"));

 produto.setPreco(Double.parseDouble
 (JOptionPane.showInputDialog("Entre com
 o preço do produto:")));

 produto.setQuantidade(Integer.parseInt
 (JOptionPane.showInputDialog("Entre com
 a quantidade do produto:")));

 //definindo o status do produto (ENUM)
 if(produto.getQuantidade() > 0) {
 produto.setStatus(StatusProduto.Disponivel);
 }
 else {
 produto.setStatus(StatusProduto.Esgotado);
 }

 //criando uma variavel para a interface,
 //sem inicializa-la
 IProdutoRepository produtoRepository = null; //vazio!

 String opcao = JOptionPane.showInputDialog
 ("Informe TXT ou CSV para exportar os dados:");

 //fazendo o polimorfismo..

```
switch(opcao) {

    case "TXT":
        produtoRepository
            = new ProdutoRepositoryTXT();
        //POLIMORFISMO!
        break;

    case "CSV":
        produtoRepository
            = new ProdutoRepositoryCSV();
        //POLIMORFISMO!
        break;

}

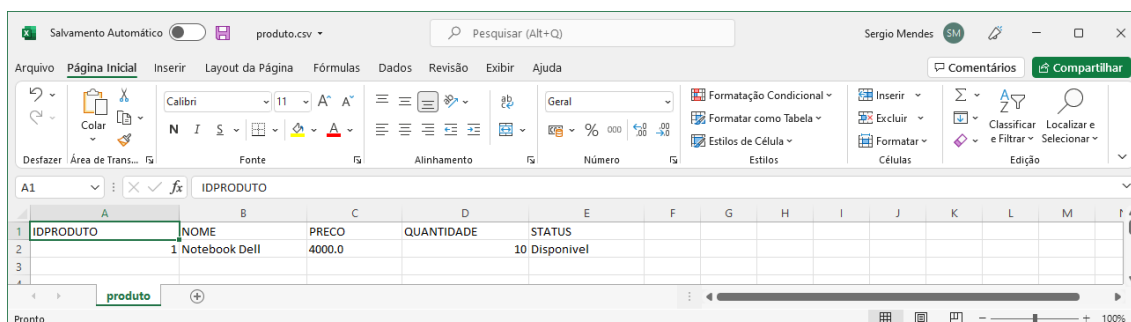
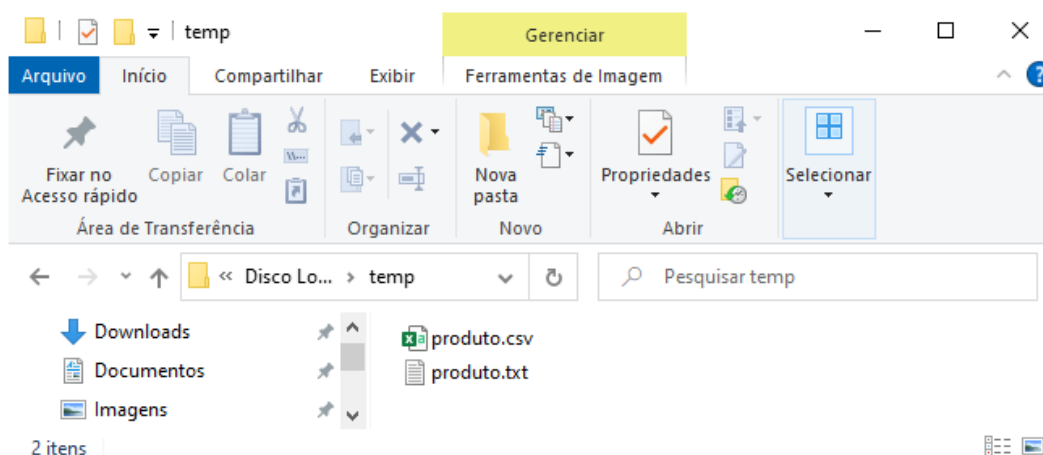
//gerando o arquivo
produtoRepository.exportarDados(produto);

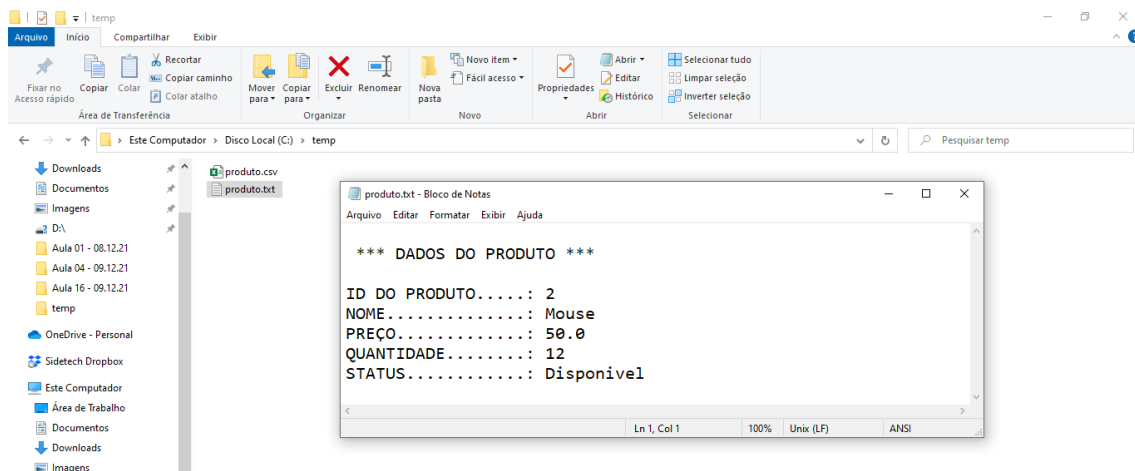
//exibir mensagem de sucesso
JOptionPane.showMessageDialog
    (null, "Dados gravados com sucesso!");

} catch (Exception e) {
    System.out.println("\nErro: " + e.getMessage());
}

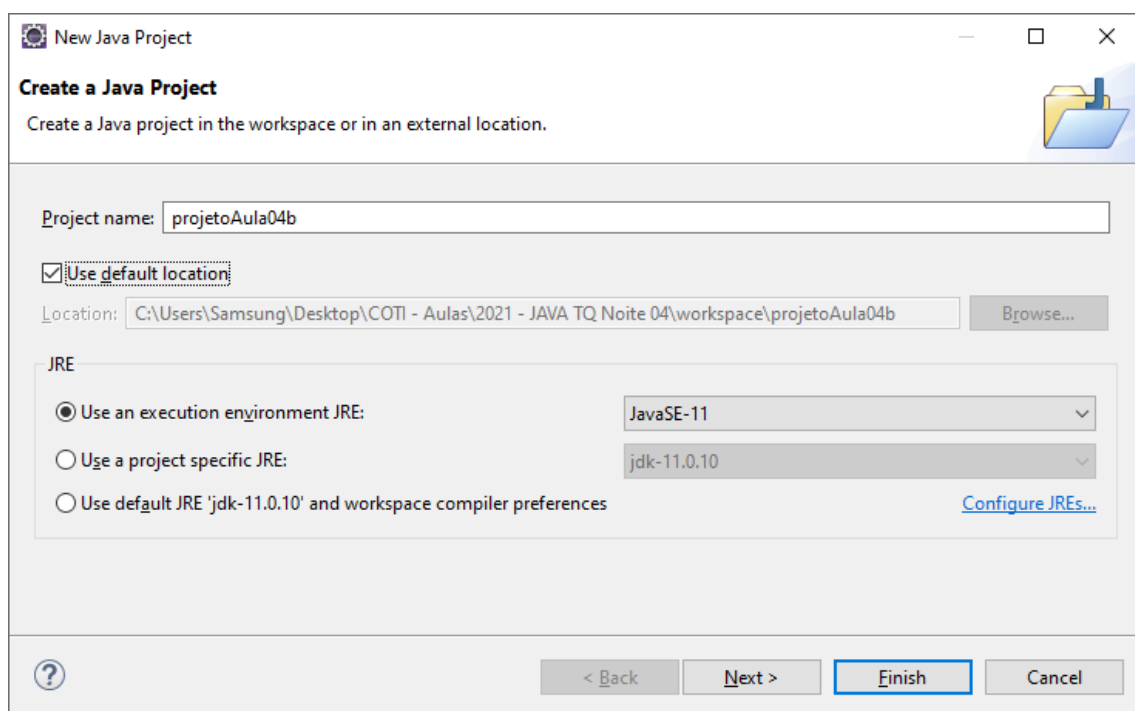
}
```

Arquivos gerados:

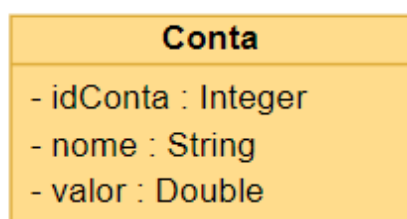




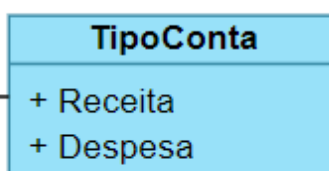
Novo projeto:



/entities



/enums



1

```
package enums;
```

```
public enum TipoConta {  
    Receita,  
    Despesa  
}
```

```
package entities;
```

```
import enums.TipoConta;
```

```
public class Conta {  
  
    private Integer idConta;  
    private String nome;  
    private Double valor;  
    private TipoConta tipo;  
  
    public Conta() {  
        // TODO Auto-generated constructor stub  
    }  
  
    public Conta(Integer idConta, String nome,  
        Double valor, TipoConta tipo) {  
        super();  
        this.idConta = idConta;  
        this.nome = nome;  
        this.valor = valor;  
        this.tipo = tipo;  
    }  
  
    public Integer getIdConta() {  
        return idConta;  
    }  
  
    public void setIdConta(Integer idConta) {  
        this.idConta = idConta;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public Double getValor() {  
        return valor;  
    }  
}
```

```

    public void setValor(Double valor) {
        this.valor = valor;
    }

    public TipoConta getTipo() {
        return tipo;
    }

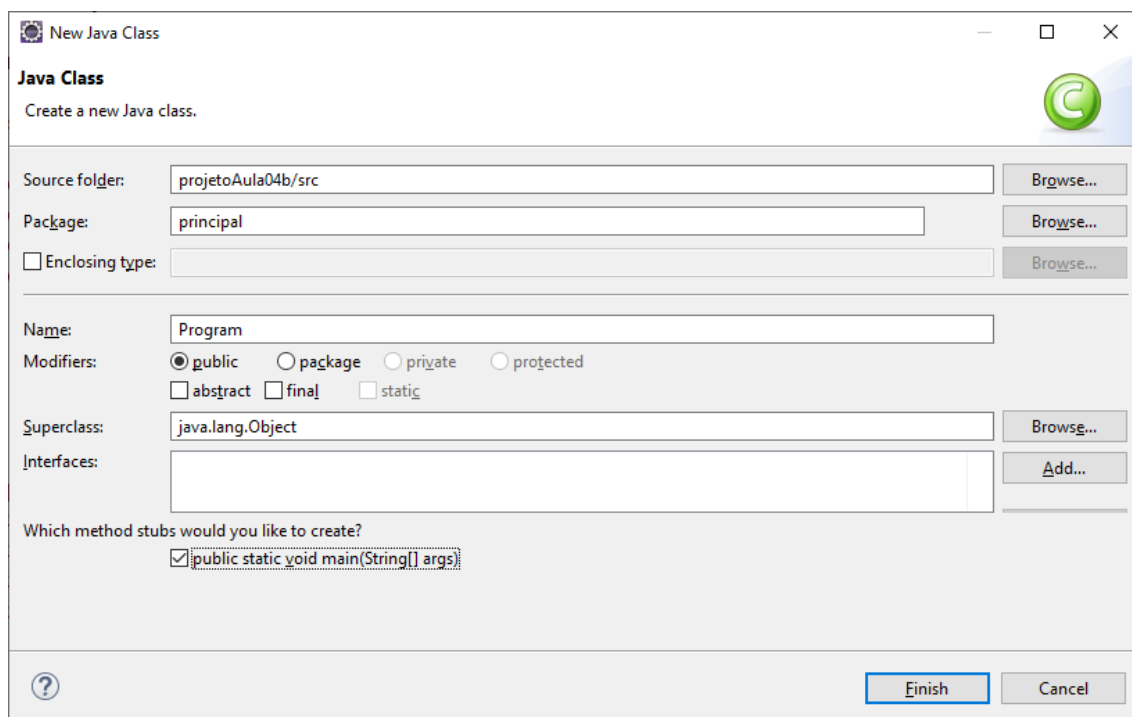
    public void setTipo(TipoConta tipo) {
        this.tipo = tipo;
    }

    @Override
    public String toString() {
        return "Conta [idConta=" + idConta + ", nome="
            + nome + ", valor=" + valor + ", tipo="
            + tipo + "]";
    }
}

```

/principal/**Program.java**

Executando os dados de Conta:



New Java Class

Create a new Java class.

Source folder: projetoAula04b/src Browse...

Package: principal Browse...

☐ Enclosing type: Browse...

Name: Program

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

Superclass: java.lang.Object Browse...

Interfaces: Add...

Which method stubs would you like to create?

☒ public static void main(String[] args)

? Finish Cancel

```
package principal;
```

```
import javax.swing.JOptionPane;
```

```
import entities.Conta;
```

```
import enums.TipoConta;
```

```
public class Program {

    public static void main(String[] args) {

        try {

            Conta conta = new Conta();

            conta.setIdConta(Integer.parseInt(
                JOptionPane.showInputDialog(
                    "Entre com o ID da Conta:")));

            conta.setNome(JOptionPane.showInputDialog(
                "Entre com o Nome da Conta:"));

            conta.setValor(Double.parseDouble(
                JOptionPane.showInputDialog(
                    "Entre com o Valor da Conta:")));

            Integer tipoConta = Integer.parseInt(JOptionPane
                .showInputDialog("Informe (1) Receita
                ou (2) Despesa:"));

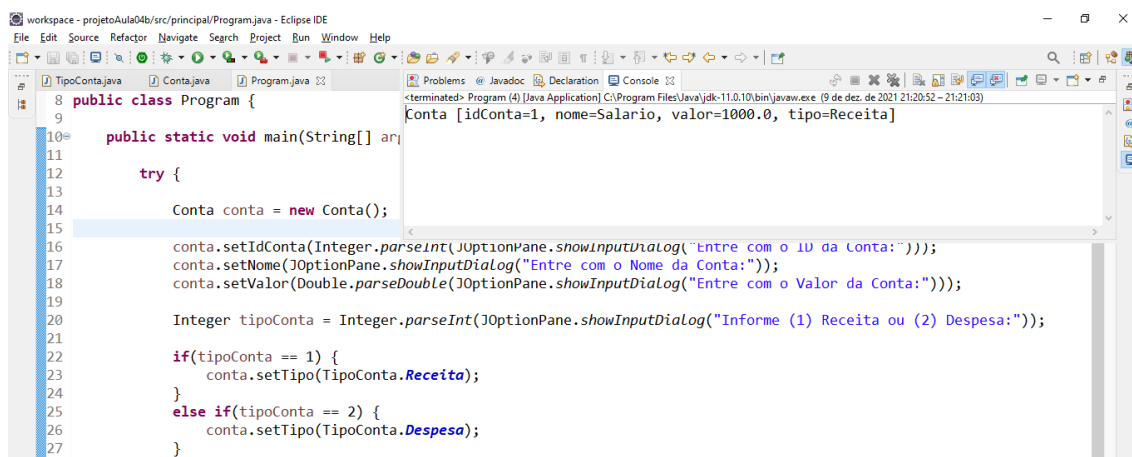
            if(tipoConta == 1) {
                conta.setTipo(TipoConta.Receita);
            }
            else if(tipoConta == 2) {
                conta.setTipo(TipoConta.Despesa);
            }

            System.out.println(conta.toString());

        }
        catch(Exception e) {
            System.out.println("\nErro: " + e.getMessage());
        }

    }

}
```



Classes Abstratas

São classes em Programação Orientada a Objetos que, assim como interfaces, podem ter métodos abstratos. Ou seja, métodos que deverão ser implementados pelas classes que herdarem a classe abstrata.

Exemplo:

<< CLASSE ABSTRATA >>

ResumoConta

+ *imprimirDados(conta : Conta) : void*

/abstractions/**ResumoConta.java**

Criando uma classe abstrata.

** Uma classe é abstrata quando a declaramos com a palavra reservada **abstract**.

```
package abstractions;
```

```
public abstract class ResumoConta {  
}
```

Uma classe abstrata pode ter todo o conteúdo de uma classe comum (atributos, construtores, métodos etc.) mas também pode ter **métodos abstratos**.

Estes métodos abstratos deverão ser implementados pelas classes que HERDAREM a classe abstrata.

```
package abstractions;
```

```
import entities.Conta;
```

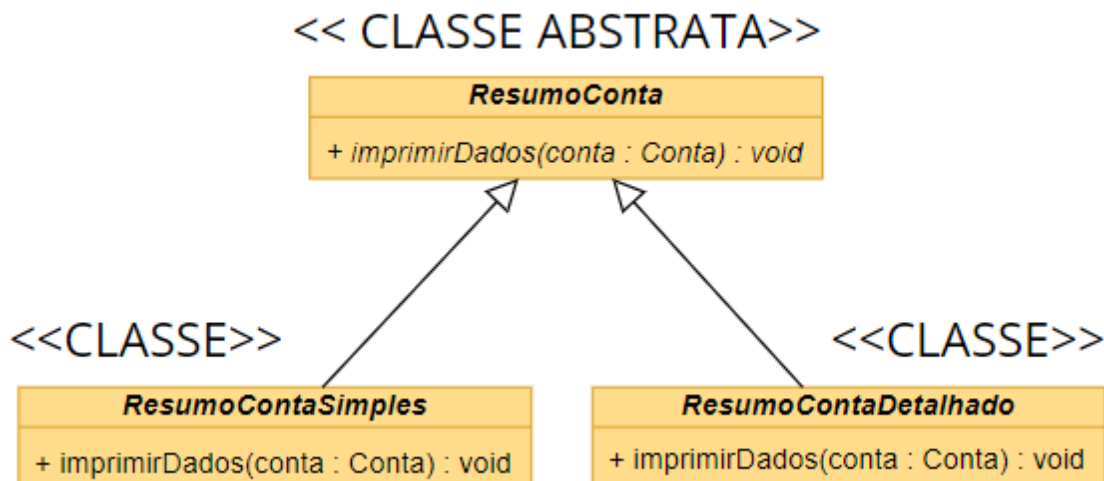
```
public abstract class ResumoConta {  
    public abstract void imprimirDados(Conta conta);  
}
```

Regra: A classe que HERDAR esta classe abstrata deverá implementar (fornecer corpo) todos os métodos abstratos

Exemplo:

Iremos HERDAR a Classe Abstrata de forma a implementar o método imprimirDados de forma simples ou detalhada.

Este tipo de cenário configura o uso de **POLIMORFISMO**.



```

package services;

import abstractions.ResumoConta;
import entities.Conta;

public class ResumoContaSimples extends ResumoConta {

    @Override
    public void imprimirDados(Conta conta) {
        // TODO Auto-generated method stub
    }
}
  
```

Implementando o método **imprimirDados**:

```

package services;

import abstractions.ResumoConta;
import entities.Conta;

public class ResumoContaSimples extends ResumoConta {

    @Override
    public void imprimirDados(Conta conta) {

        System.out.println
            ("\n *** RESUMO DE CONTA SIMPLES *** \n");
    }
}
  
```

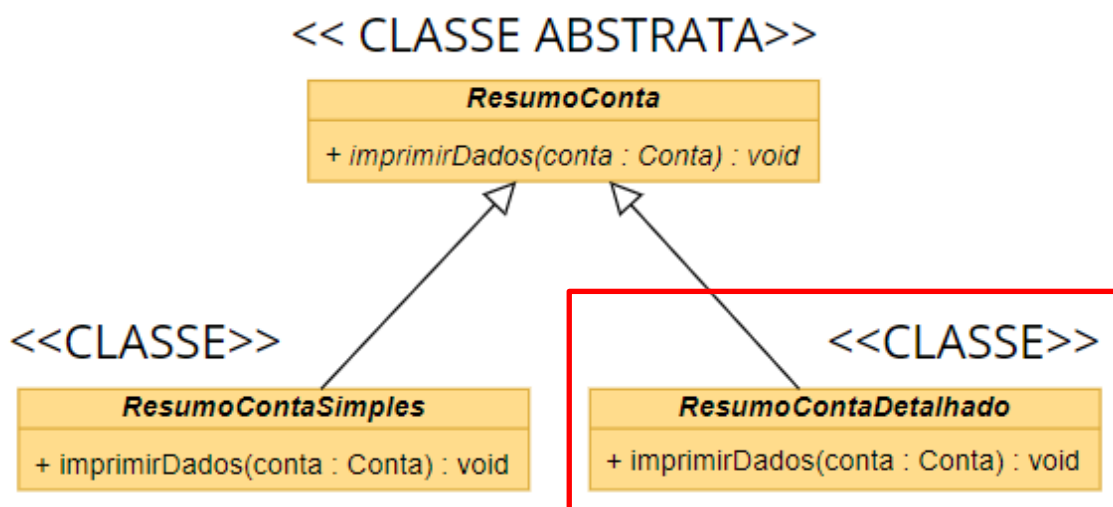


```

        System.out.println("ID.....: " + conta.getIdConta());
        System.out.println("NOME....: " + conta.getNome());
        System.out.println("VALOR...: " + conta.getValor());
        System.out.println("TIPO....: " + conta.getTipo());
    }
}

```

Criando mais uma implementação para a classe abstrata:



```

package services;

import java.util.Date;

import abstractions.ResumoConta;
import entities.Conta;

public class ResumoContaDetalhado extends ResumoConta {

    @Override
    public void imprimirDados(Conta conta) {

        System.out.println
            ("\n *** RESUMO DE CONTA DETALHADO *** \n");

        System.out.println("ID.....: " + conta.getIdConta());
        System.out.println("NOME.....: " + conta.getNome());
        System.out.println("VALOR.....: " + conta.getValor());
        System.out.println("TIPO.....: " + conta.getTipo());
        System.out.println("GERADO EM.: " + new Date());
        System.out.println("USUÁRIO...: COTI INFORMÁTICA" );
    }
}

```

POLIMORFISMO

Vamos fazer com que o usuário do sistema possa escolher que tipo de impressão ele deseja fazer (SIMPLES ou DETALHADO). Para isso iremos fazer um POLIMORFISMO da classe abstrata `ResumoConta`.

Exemplo:

```
package principal;

import javax.swing.JOptionPane;

import abstractions.ResumoConta;
import entities.Conta;
import enums.TipoConta;
import services.ResumoContaDetalhado;
import services.ResumoContaSimples;

public class Program {

    public static void main(String[] args) {

        try {

            Conta conta = new Conta();

            conta.setIdConta(Integer.parseInt
                (JOptionPane.showInputDialog
                ("Entre com o ID da Conta:")));

            conta.setNome(JOptionPane.showInputDialog
                ("Entre com o Nome da Conta:"));

            conta.setValor(Double.parseDouble
                (JOptionPane.showInputDialog
                ("Entre com o Valor da Conta:")));

            Integer tipoConta = Integer.parseInt
                (JOptionPane.showInputDialog
                ("Informe (1) Receita ou (2) Despesa:"));

            if (tipoConta == 1) {

                conta.setTipo(TipoConta.Receita);

            } else if (tipoConta == 2) {

                conta.setTipo(TipoConta.Despesa);

            }

        }

    }

}
```

```
Integer tipoImpressao = Integer
    .parseInt(JOptionPane.showInputDialog
        ("(1) Impressão Simples,
         (2) Impressão detalhada: "));

ResumoConta resumoConta = null; // vazio

switch (tipoImpressao) {

    case 1:
        resumoConta = new ResumoContaSimples();
        //POLIMORFISMO!
        break;

    case 2:
        resumoConta = new ResumoContaDetalhado();
        //POLIMORFISMO!
        break;

}

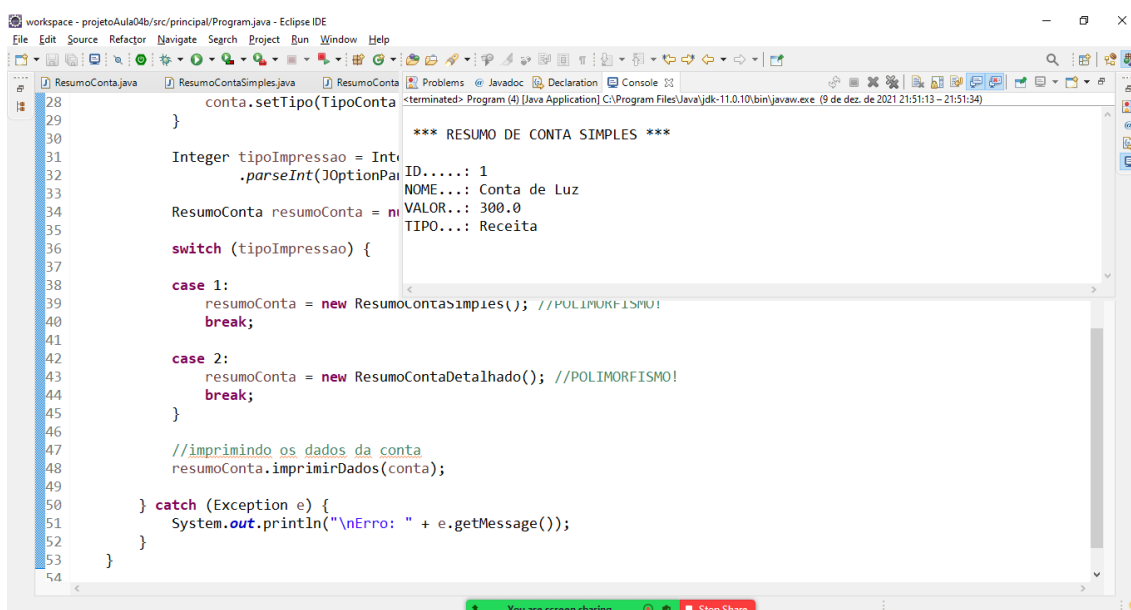
//imprimindo os dados da conta
resumoConta.imprimirDados(conta);

} catch (Exception e) {
    System.out.println("\nErro: " + e.getMessage());
}

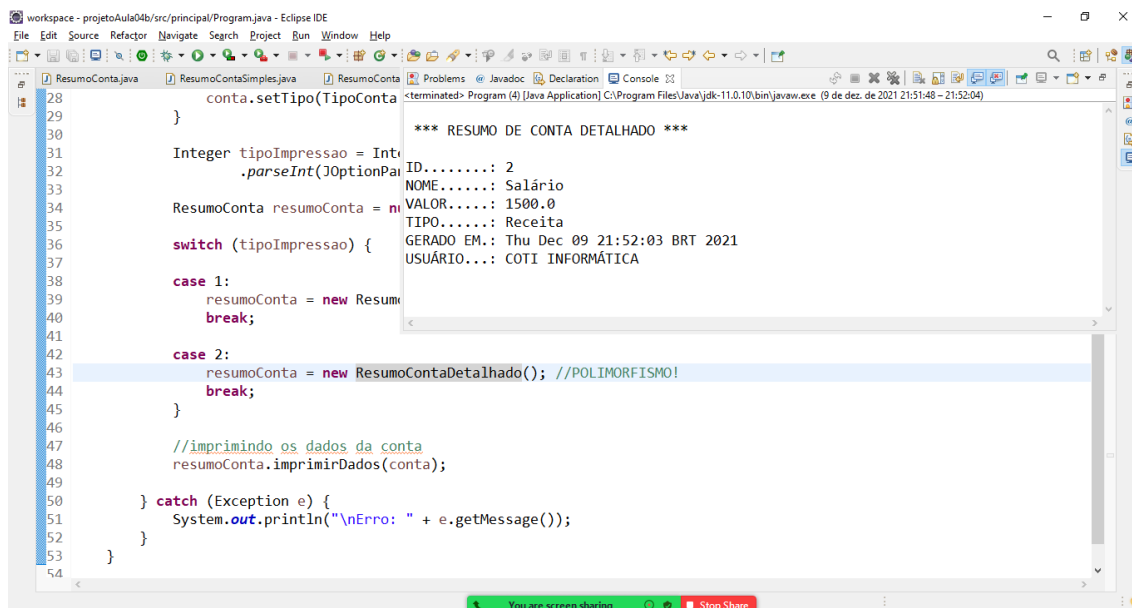
}

}
```

Executando:



```
workspace - projetoAula04b/src/principal/Program.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
*** RESUMO DE CONTA SIMPLES ***
ID.....: 1
NOME.....: Conta de Luz
VALOR....: 300.0
TIPO.....: Receita
```



```

workspace - projetoAula04b/src/principal/Program.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
ResumoConta.java ResumoContaSimples.java ResumoConta
28      conta.setTipo(TipoConta
29      }
30
31      Integer tipoImpressao = Integer.parseInt(optionPai
32      ,parseInt(30ptionPai
33
34      ResumoConta resumoConta = new ResumoConta(tipoImpressao,
35
36      switch (tipoImpressao) {
37
38      case 1:
39          resumoConta = new ResumoContaSimple();
40          break;
41
42      case 2:
43          resumoConta = new ResumoContaDetalhado(); //POLIMORFISMO!
44          break;
45      }
46
47      //imprimindo os dados da conta
48      resumoConta.imprimirDados(conta);
49
50      catch (Exception e) {
51          System.out.println("\nErro: " + e.getMessage());
52      }
53  }
54
*** RESUMO DE CONTA DETALHADO ***
ID.....: 2
NOME.....: Salário
VALOR.....: 1500.0
TIPO.....: Receita
GERADO EM.: Thu Dec 09 21:52:03 BRT 2021
USUÁRIO...: COTI INFORMÁTICA
  
```

- ▼ projetoAula04a
 - > JRE System Library [JavaSE-11]
 - ▼ src
 - ▼ entities
 - > Produto.java
 - ▼ enums
 - > StatusProduto.java
 - ▼ interfaces
 - > IProdutoRepository.java
 - ▼ principal
 - > Program.java
 - ▼ repositories
 - > ProdutoRepositoryCSV.java
 - > ProdutoRepositoryTXT.java
- ▼ projetoAula04b
 - > JRE System Library [JavaSE-11]
 - ▼ src
 - ▼ abstractions
 - > ResumoConta.java
 - ▼ entities
 - > Conta.java
 - ▼ enums
 - > TipoConta.java
 - ▼ principal
 - > Program.java
 - ▼ services
 - > ResumoContaDetalhado.java
 - > ResumoContaSimples.java