



Programação Orientada a Objetos I

Prof^a. Angela Abreu Rosa de Sá, Dr^a.

Contato: angelaabreu@gmail.com

Ementa

Fundamentos da orientação a objetos

Histórico e introdução à orientação a objetos
Conceitos básicos de orientação a objetos
Construtores e sobrecarga

1

Estruturas de programação orientadas a objetos

Estruturas de decisão e controle em Java
Estruturas de repetição em Java
Reutilização de classes em Java

2

Exceções, classes abstratas e interfaces

Definição e tratamento de exceções
Definição e uso de classes abstratas
Definição e uso de interfaces

3

Aplicações orientadas a objetos

Arrays em Java
Strings em Java
Coleções e arquivos

4

Material Didático

Programação Orientada a Objetos

Sumário

Unidade 1 Fundamentos da orientação a objetos	7
Seção 1.1 - Histórico e introdução à orientação a objetos	9
Seção 1.2 - Conceitos básicos de orientação a objetos	22
Seção 1.3 - Construtores e sobrecarga	37
Unidade 2 Estruturas de programação orientadas a objetos	59
Seção 2.1 - Estruturas de decisão e controle em Java	61
Seção 2.2 - Estruturas de repetição em Java	76
Seção 2.3 - Reutilização de classes em Java	93
Unidade 3 Exceções, classes abstratas e interfaces	111
Seção 3.1 - Definição e tratamento de exceções	113
Seção 3.2 - Definição e uso de classes abstratas	126
Seção 3.3 - Definição e uso de interfaces	141
Unidade 4 Aplicações orientadas a objetos	155
Seção 4.1 - Arrays em Java	157
Seção 4.2 - Strings em Java	173
Seção 4.3 - Coleções e arquivos	188

Exceções



Assimile

“Uma exceção é uma indicação de um problema que ocorre durante a execução de um programa. O nome “exceção” significa que o problema não ocorre frequentemente” (DEITEL; DEITEL, 2010, p. 336).

Erros que podem ser gerados **durante a execução** de um programa.

Não ocorre com **frequência**, mas **PODE acontecer**.

E quando acontece, **“derruba” (interrompe)** a execução do programa!

Exceções

Problems @ Javadoc Declaration Console

<terminated> DivisaoPorZeroSemTratamento [Java Application] C:\Program Files (x86)\Java\jre1.8.0_144\bin\javaw.exe

Informe o numerador: 75
Informe o denominador: 5

Resultado: 75 / 5 = 15

Problems @ Javadoc Declaration Console

<terminated> DivisaoPorZeroSemTratamento [Java Application] C:\Program Files (x86)\Java\jre1.8.0_144\bin\javaw.exe

Informe o numerador: doze

Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:864)
at java.util.Scanner.next(Scanner.java:1485)
at java.util.Scanner.nextInt(Scanner.java:2117)
at java.util.Scanner.nextInt(Scanner.java:2076)
at DivisaoPorZeroSemTratamento.main(DivisaoPorZeroSemTratamento.java:9)

Exceções

As **exceções** ocorrem quando algo **imprevisto** acontece.

Elas podem ser provenientes de **erros de lógica, leitura de dados incorretos ou acesso a recursos que talvez não estejam disponíveis.**

Alguns possíveis motivos externos para ocorrer uma exceção são:

- Leitura de um tipo de dado incorreto
(Exemplo: leitura de uma string ao invés de um inteiro)
- Acesso a região de memória inexistente
(Exemplo: acessar objeto que foi declarado mas não foi instanciado)
- Erro de lógica em operações
(Exemplo: divisão por zero | utilizar variáveis não inicializadas)

Como solucionar este problema?



Tratamento de Exceções

O **tratamento da exceção** serve justamente para que o programa possa **continuar sendo executado em vez de ser encerrado repentinamente**, o que confere confiabilidade e robustez às aplicações.

Exceções

try

{

// são escritas as linhas de código que podem acontecer uma exceção

}

catch (tipo de exceção)

{

// é descrita a ação que vai ser realizada quando a exceção acontecer

}

Exceções

```
/**
 * Classe utilizada para demonstrar o bloco try / catch.
 */
public class ExemploExcecao {
    public static void main(String[] args) {
        try {
            /* Trecho de código no qual uma
             * exceção pode acontecer.
             */
        } catch (Exception ex) {
            /* Trecho de código no qual uma
             * exceção do tipo "Exception" será tratada.
             */
        }
    }
}
```


Exceções

```
import java.util.Scanner;

/**
 * Classe que demonstra o uso do try / catch.
 */
public class ExemploTryCatch {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        try {
            System.out.print("Digite um valor inteiro..:");
            int numero1 = s.nextInt();
            System.out.print("Digite um valor inteiro..:");
            int numero2 = s.nextInt();

            System.out.println(numero1+ " + " + numero2 + " = " + (numero1+numero2));
        } catch (Exception ex) {
            System.out.println("ERRO - Valor digitado nao e um numero inteiro!");
        }
    }
}
```

Java Class

 The use of the default package is discouraged.



Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:

Interfaces:

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



```
import java.util.Scanner;

public class TratamentoExcecao {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);
        try
        {
            System.out.print("Digite um valor inteiro..");
            int numero1 = s.nextInt();
            System.out.print("Digite um valor inteiro..");
            int numero2 = s.nextInt();

            System.out.println(numero1+ " + " + numero2 + " = " + (numero1+numero2));

        } catch (Exception ex)
        {

            System.out.println("ERRO - Valor digitado nao e um numero inteiro!");

        }

    }
}
```

Problems @ Javadoc Declaration Console ×

<terminated> TratamentoExcecao [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (5 de out. de 2022 16:02:15 – 16:02:19) [pid: 10572]

Digite um valor inteiro..:dois

ERRO - Valor digitado nao e um numero inteiro!

```

4 public class TratamentoExcecao {
5
6
7     public static void main(String[] args) {
8
9         int contador = 0;
10        Scanner s = new Scanner(System.in);
11
12        while (contador < 5)
13        {
14
15            try
16            {
17                System.out.print("Digite um valor inteiro..");
18                int numero1 = s.nextInt();
19                System.out.print("Digite um valor inteiro..");
20                int numero2 = s.nextInt();
21
22                System.out.println(numero1 + " + " + numero2 + " = " + (numero1+numero2));
23
24            } catch (Exception ex)
25            {
26
27                System.out.println("ERRO - Valor digitado nao e um numero inteiro!");
28
29            }
30
31            contador++;
32
33            .....
34        }
35    }
36 }

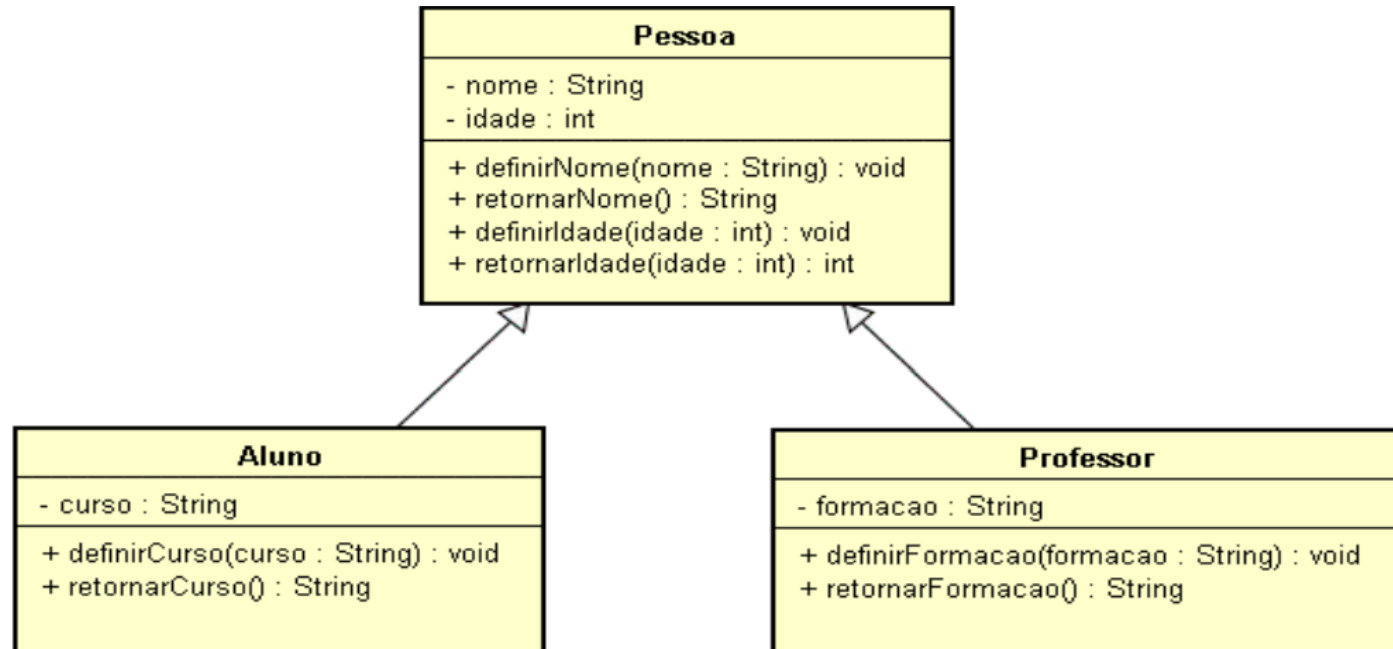
```

<terminated> TratamentoExcecao [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (5 de out. de 2022 16:14:55 – 16:14:57) [pid: 11604]
 Digite um valor inteiro...dois
 ERRO - Valor digitado nao e um numero inteiro!
 Digite um valor inteiro...ERRO - Valor digitado nao e um numero inteiro!
 Digite um valor inteiro...ERRO - Valor digitado nao e um numero inteiro!
 Digite um valor inteiro...ERRO - Valor digitado nao e um numero inteiro!
 Digite um valor inteiro...ERRO - Valor digitado nao e um numero inteiro!

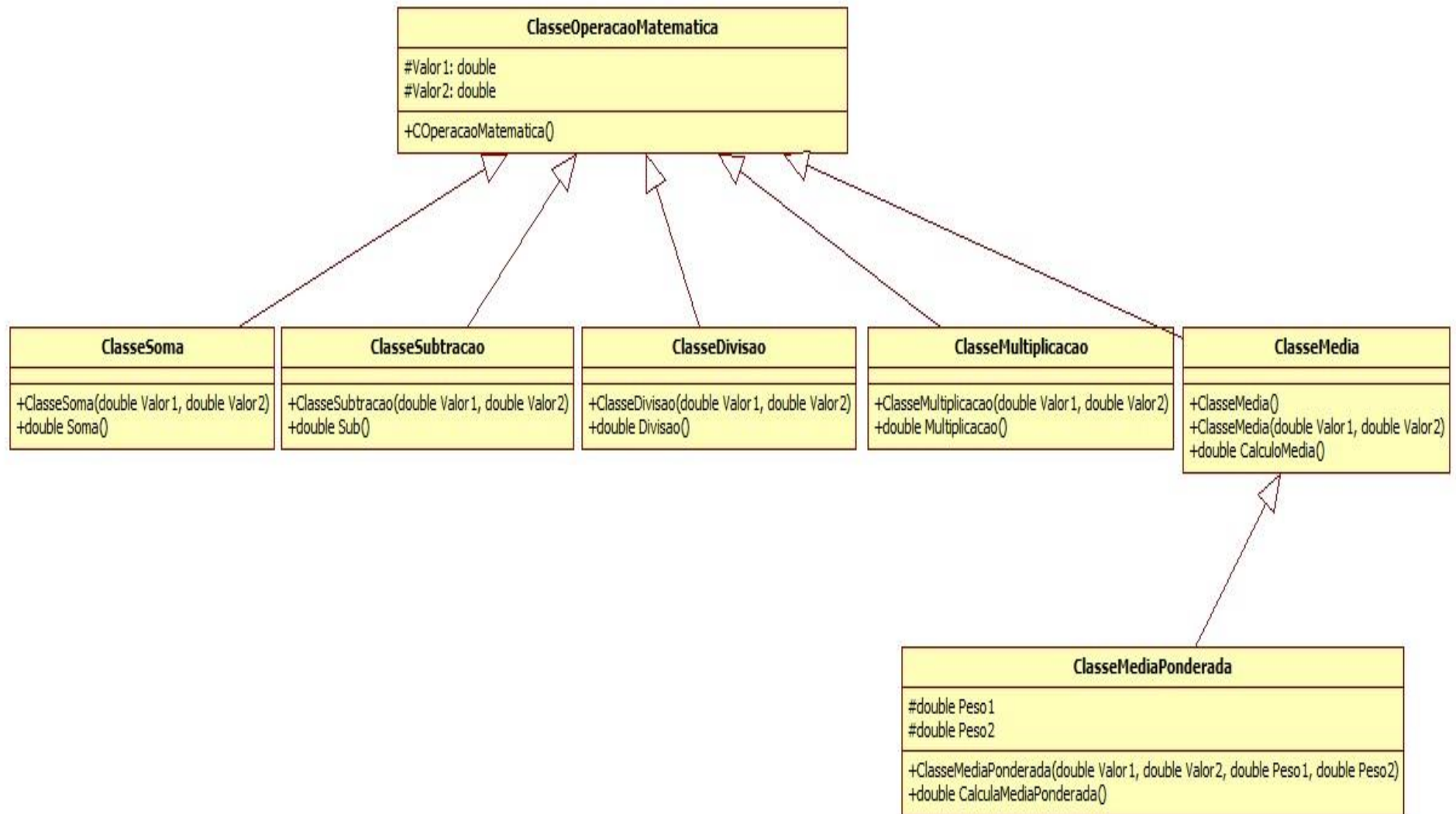
```
public class TratamentoExcecao {  
  
    public static void main(String[] args) {  
  
        int contador = 0;  
        Scanner s = new Scanner(System.in);  
  
        while (contador < 5)  
        {  
  
            try  
            {  
  
                System.out.print("Digite um valor inteiro..");  
                int numero1 = s.nextInt();  
                System.out.print("Digite um valor inteiro..");  
                int numero2 = s.nextInt();  
  
                System.out.println(numero1 + " + " + numero2 + " = " + (numero1+numero2));  
  
            } catch (Exception ex)  
            {  
  
                System.out.println("ERRO - Valor digitado nao e um numero inteiro!");  
                break;  
  
            }  
  
            contador++;  
  
        }  
  
    }  
}
```

Digite um valor inteiro..:dois
ERRO - Valor digitado nao e um numero inteiro!

Herança



Herança



Classes Abstratas

A classe abstrata é uma classe que **não permite a criação de instâncias a partir dela**, isto é, **não permite que sejam criados objetos**; ao contrário, uma classe concreta permite a geração de instâncias (FURGERI, 2013).



Usando classes abstratas o desenvolvedor pode declarar classes que definem **somente parte de uma implementação**, deixando para as classes estendidas o oferecimento de implementações específicas.

Classes Abstratas



Assimile

Uma classe abstrata não se destina a ser instanciada, logo não precisa fornecer uma implementação completa. Em vez disso, funciona como um modelo ou padrão a partir do qual outras variáveis e métodos podem ser adicionados em subclasses (RUSSEL; ROBERTS, 2009).

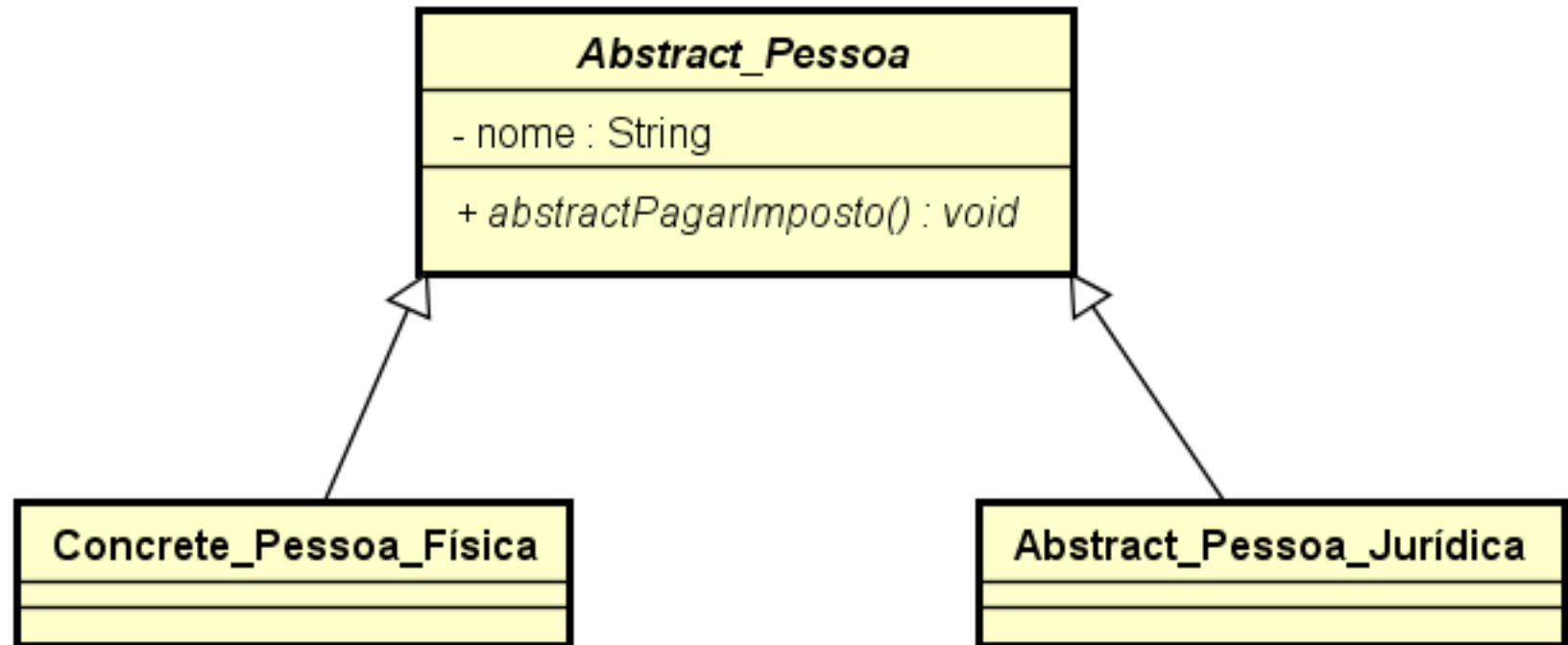
Classes Abstratas

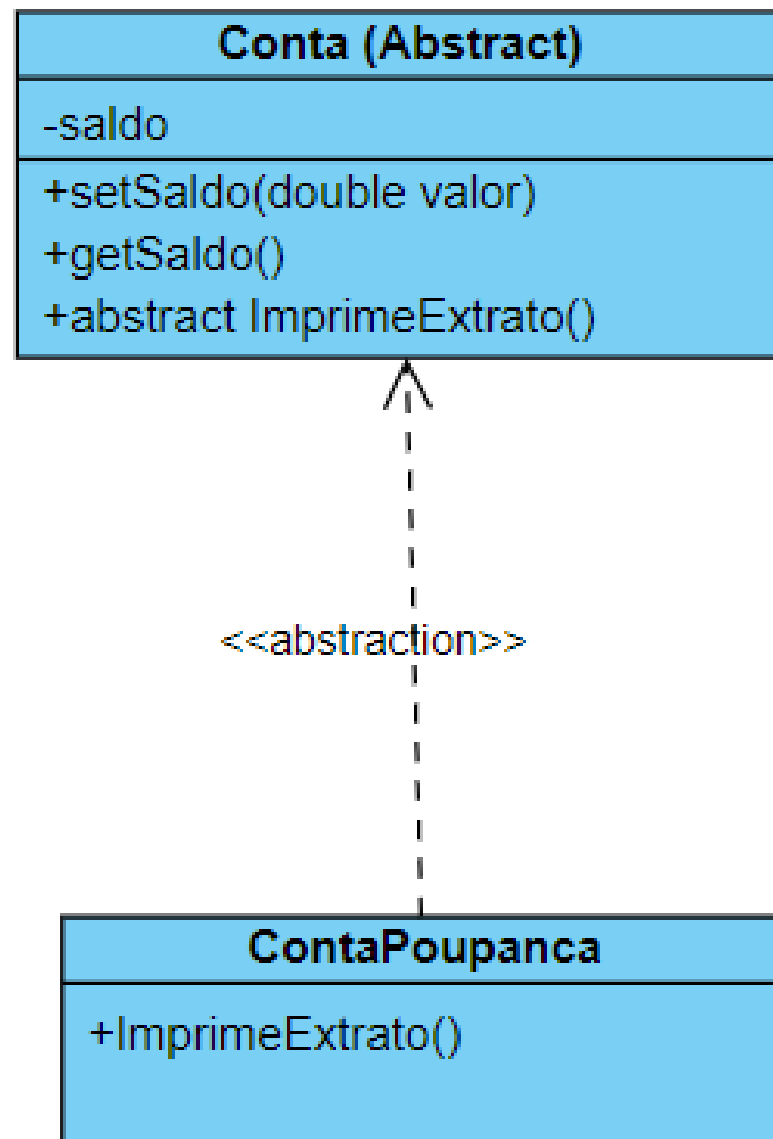
Existe a possibilidade de que sejam criados também **métodos abstratos**, que **compartilham comportamentos** como outros objetos.



Cada **método** não implementado na classe abstrata também é indicado como **abstract**, embora isso também pode ser realizado por meio das **interfaces**.

Classes Abstratas





Classes Abstratas

New Java Class

Java Class

⚠ The use of the default package is discouraged.

Source folder: ClasseAbstrata/src Browse...

Package: (default) Browse...

☐ Enclosing type: Browse...

Name: Conta

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☒ abstract ☐ final ☐ static
☒ none ☐ sealed ☐ non-sealed ☐ final

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

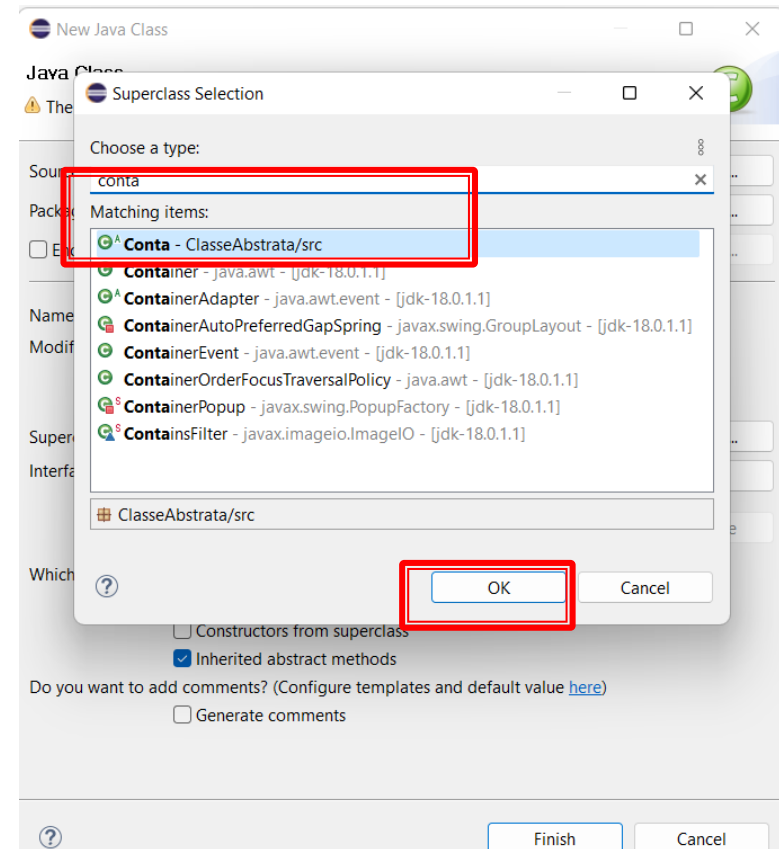
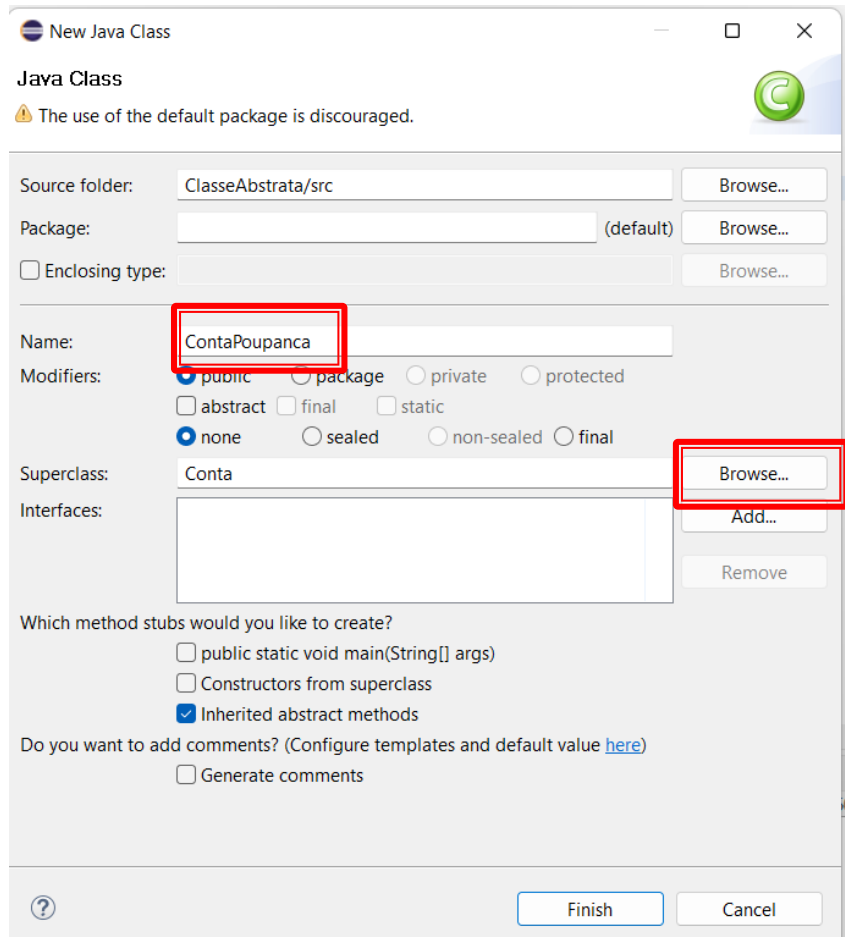
Finish Cancel

Classes Abstratas

*Conta.java ×

```
1
2 public abstract class Conta {
3
4     private double saldo;
5
6     public void setSaldo(double saldo) {
7         this.saldo = saldo;
8     }
9
10    public double getSaldo() {
11        return saldo;
12    }
13
14    public abstract void imprimeExtrato();
15
16 }
17
```

Classes Abstratas




```
public class ContaPoupanca extends Conta {  
  
    @Override  
    public void imprimirExtrato() {  
        // TODO Auto-generated method stub  
  
    }  
  
}
```





Alterar

```
public class ContaPoupanca extends Conta {  
  
    public ContaPoupanca(double valor)  
    {  
        this.setSaldo(valor);  
    }  
  
    @Override  
    public void imprimeExtrato() {  
  
        System.out.println("### Extrato da Conta ###");  
  
        System.out.println("Saldo: "+this.getSaldo());  
  
    }  
}
```

New Java Class

Java Class



 The use of the default package is discouraged.

Source folder: ClasseAbstrata/src

Browse...

Package: (default)

Browse...

☐ Enclosing type:

Browse...

Name: Principal

Modifiers:

☒ public

☐ package

☐ private

☐ protected

☐ abstract

☐ final

☐ static

☒ none

☐ sealed

☐ non-sealed

☐ final

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

```
public class Principal {  
    public static void main(String[] args) {  
  
        Conta cp = new ContaPoupanca(1520);  
  
        cp.imprimeExtrato();  
  
    }  
}
```

```
<terminated> Principal [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (5 de out  
### Extrato da Conta ###  
Saldo: 1520.0
```

Interfaces

Se a classe abstrata **não possuir nenhum método concreto (não abstrato)**, então podemos declará-la como uma **interface**. Uma interface é como uma classe, mas **contém apenas declarações vazias de seus métodos**.



O desenvolvedor de uma interface **declara os métodos que devem ser oferecidos** pelas classes que implementam a interface e **declara o que esses métodos devem fazer**.

Interfaces

Costuma-se dizer que uma interface permite estabelecer um "contrato" entre as classes; funciona de maneira bastante similar as classes abstratas, porém **não permite implementação de nenhum método, contendo apenas a especificação deste** (FURGERI, 2013).

Interfaces

Diferença básica entre classes abstratas e interfaces :

Classe Abstrata



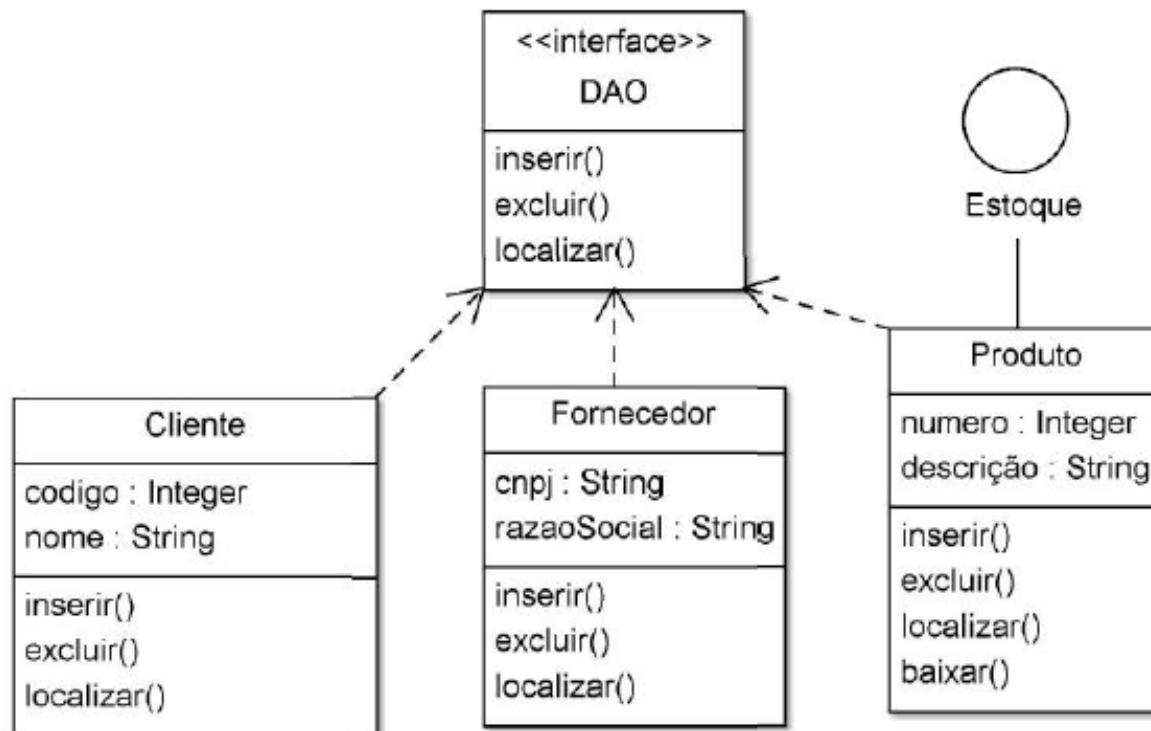
qualquer classe pode implementar várias interfaces simultaneamente.

Interface

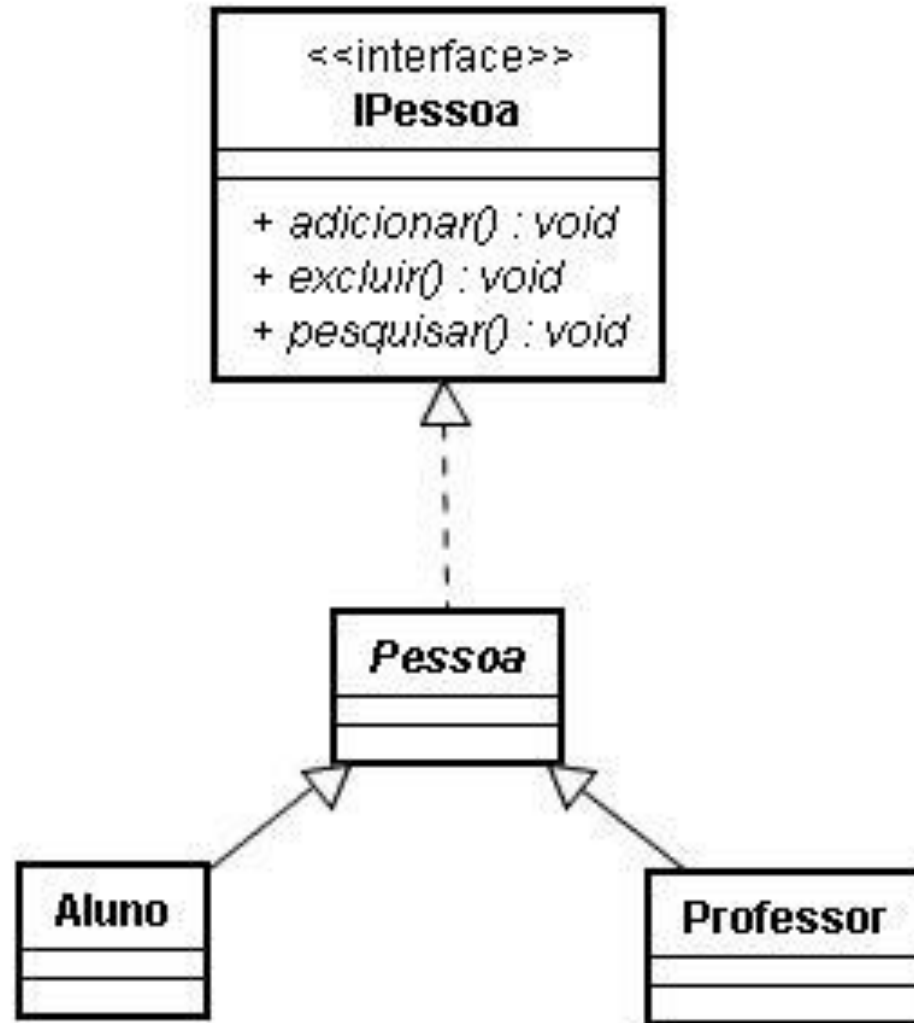


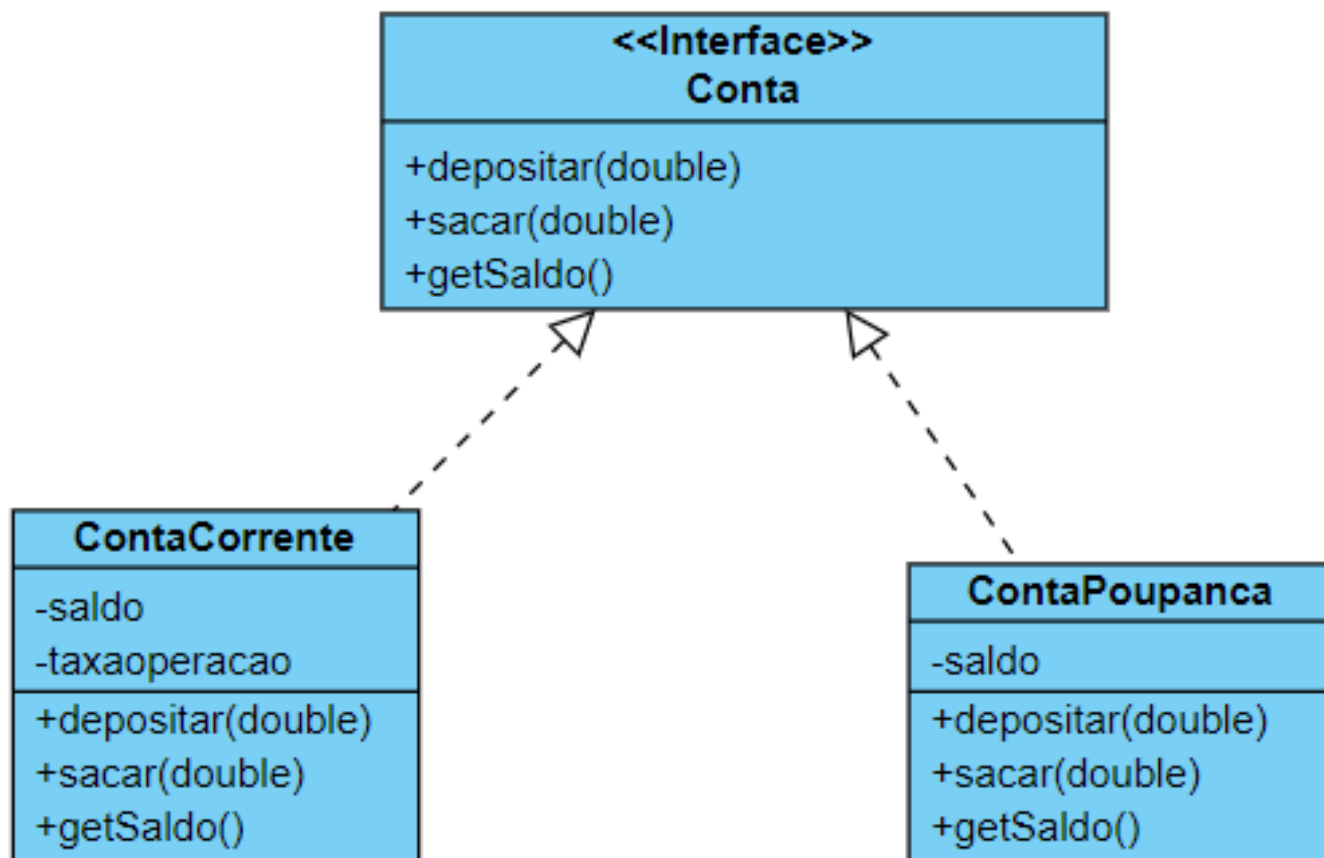
classe herdeira somente **pode herdar de uma única classe (independentemente de ser abstrata ou não),**

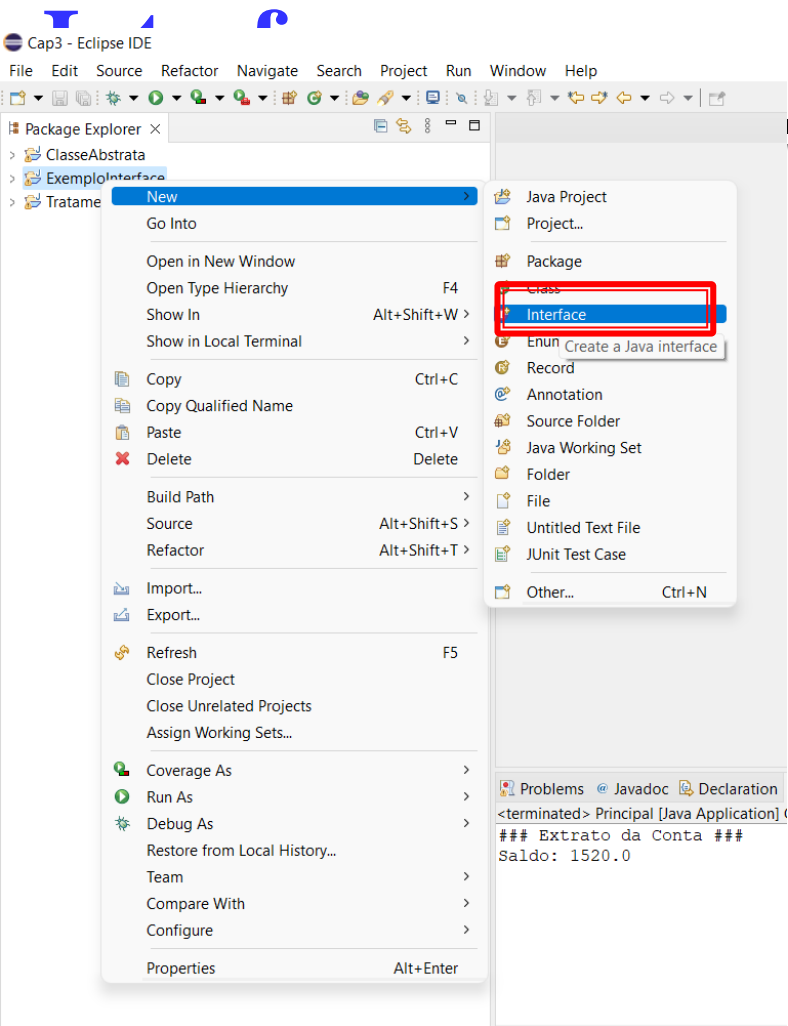
Interfaces



Interfaces







New Java Interface

Java Interface

⚠ The use of the default package is discouraged.



Source folder:

Browse...

Package: (default)

Browse...

☐ Enclosing type:

Browse...

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected

☒ none ☐ sealed ☐ non-sealed

Extended interfaces:

Add...

Remove

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

☐ Generate comments



Finish

Cancel

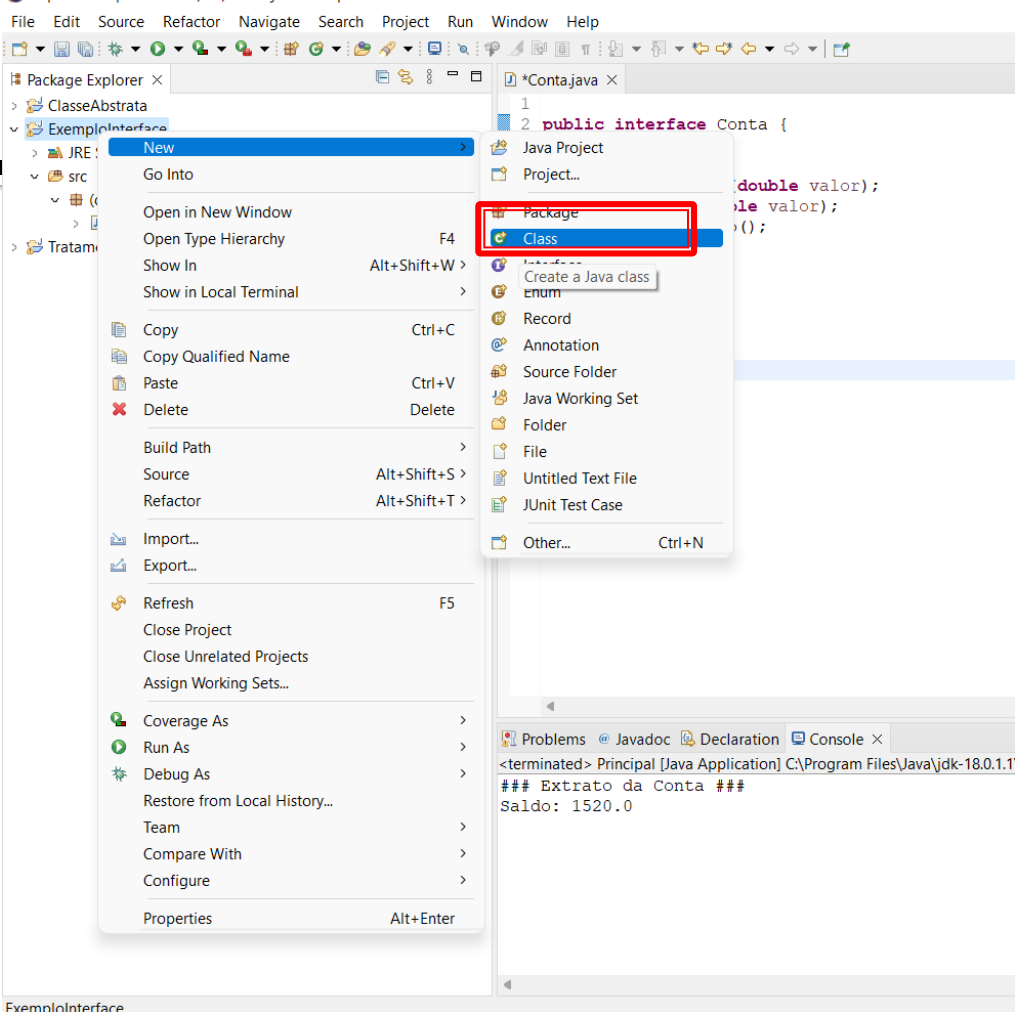
```
public interface Conta {
```

```
    void depositar(double valor);
```

```
    void sacar(double valor);
```


```
    double getSaldo();
```

```
}
```



New Java Class

Java Class

 The use of the default package is discouraged.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers:

☒ public ☐ package ☐ private ☐ protected

☐ abstract ☐ final ☐ static

☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:

Interfaces:

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

Implemented Interfaces Selection

Choose interfaces:

Matching items:

- ☒ **Conta** - ExemplolInterface/src
- ☐ ContainerListener - java.awt.event - [jdk-18.0.1.1]
- ☐ ContainerPeer - java.awt.peer - [jdk-18.0.1.1]

ExemplolInterface/src

Java Class

⚠ The use of the default package is discouraged.



Source folder:

Package:

☐ Enclosing type:

Name:


Modifiers: ☒ public ☐ package ☐ private ☐ protected

☐ abstract ☐ final ☐ static

☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:

Interfaces:

 Conta

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




```
public class ContaCorrente implements Conta {  
  
    @Override  
    public void depositar(double valor) {  
        // TODO Auto-generated method stub  
  
    }  
  
    @Override  
    public void sacar(double valor) {  
        // TODO Auto-generated method stub  
  
    }  
  
    @Override  
    public double getSaldo() {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
}
```

```
public class ContaCorrente implements Conta {
```

```
    private double saldo;
```

```
    private double taxaOperacao = 0.45;
```

```
    @Override
```

```
    public void depositar(double valor) {  
        this.saldo += valor - taxaOperacao;  
    }
```

```
    @Override
```


```
    public double getSaldo() {  
        return this.saldo;  
    }
```

```
    @Override
```

```
    public void sacar(double valor) {  
        this.saldo -= valor + taxaOperacao;  
    }
```

```
}
```

Java Class

 The use of the default package is discouraged.



Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:

Interfaces:

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



```
public class ContaPoupanca implements Conta {  
  
    @Override  
    public void depositar(double valor) {  
        // TODO Auto-generated method stub  
  
    }  
  
    @Override  
    public void sacar(double valor) {  
        // TODO Auto-generated method stub  
  
    }  
  
    @Override  
    public double getSaldo() {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
}
```

```
public class ContaPoupanca implements Conta {  
  
    private double saldo;  
  
    @Override  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
  
    @Override  
    public double getSaldo() {  
        return this.saldo;  
    }  
  
    @Override  
    public void sacar(double valor) {  
        this.saldo -= valor;  
    }  
  
}
```

Java Class

⚠ The use of the default package is discouraged.



Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass:

Interfaces:

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



```
public class Principal {  
  
    public static void main(String[] args) {  
  
        ContaCorrente cc = new ContaCorrente();  
        cc.depositar(1200.20);  
        cc.sacar(300);  
  
        System.out.println("Saldo Atual Conta Corrente: "+cc.getSaldo());  
  
        ContaPoupanca cp = new ContaPoupanca();  
        cp.depositar(500.50);  
        cp.sacar(25);  
        System.out.println("Saldo Atual Conta Poupança: "+cp.getSaldo());  
  
    }  
  
}
```

Problems @ Javadoc Declaration Console ×

<terminated> Principal (1) [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (5 de out. de 2022 17:03:49 – 17:03:50)

Saldo Atual Conta Corrente: 899.3

Saldo Atual Conta Poupança: 475.5

Interfaces

Classes

- Atributos
- Métodos

Interfaces

- Assinaturas dos métodos

Classes Abstratas

- Atributos
- Métodos
- Assinatura de Métodos



Exercícios 😊

Obs.: Em todos os programas, utilize o Tratamento de Exceção para a leitura de dados do teclado em qualquer outro trecho programa/operação que possa acontecer uma exceção.

- 1) Utilizando o conceito de Classe abstrata, implemente um programa que leia um vetor de 5 valores que serão armazenados em uma conta poupança. Em seguida, imprima um relatório com os dados armazenados.



- 2) Utilizando o conceito de Interface, implemente um programa que leia um vetor de 5 saldos para armazenar em Conta Corrente e outro vetor para armazenar 5 saldos para armazenar em uma Conta Poupança.

O programa deve apresentar um menu com as operações disponíveis: Depositar | Sacar | Mostrar saldo. Para cada operação, o usuário deverá escolher o tipo de conta (Corrente ou Poupança) e escolher o número do cliente para o qual será realizada a operação. Obs.: o número do cliente deve ser a posição ocupada nos vetores Conta Corrente e Conta Poupança.



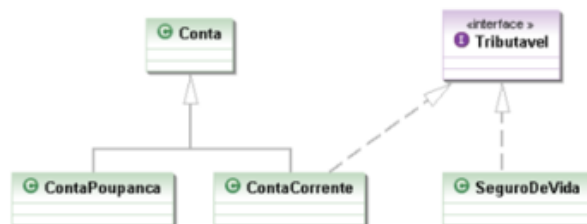
- 3) Considere a seguinte interface:

```

public interface IProduto {
    public String getNome();
    public float getCosto();
}
    
```

O método `getNome` retorna o nome do produto e o método `getCosto` retorna o valor de compra do produto (este valor representa seu custo). Implemente uma classe Produto dessa interface para armazenar o nome e o valor de um produto. Em seguida, no programa principal, solicite que o usuário cadastre 5 produtos e apresente um relatório dos dados.

- 4) Nosso banco precisa tributar dinheiro de alguns bens que nossos clientes possuem. Para isso, vamos criar um sistema para isso.



- a) Crie uma interface `Tributavel` que possui o método `calculaTributos()`, que retorna um `double`.
- b) Alguns bens são tributáveis e outros não, `ContaPoupanca` não é tributável, já para `ContaCorrente` você precisa pagar 1% da conta e o `SeguroDeVida` tem uma taxa fixa de 42 reais.
- c) As classes `ContaCorrente` e `ContaPoupanca` herdam de uma classe Abstrata `Conta`. Essa classe `Conta` possui um saldo e os métodos `sacar(double)`, `depositar(double)` e `obterSaldo()` que retorna o saldo da conta.
- d) Crie uma classe principal para testar o funcionamento das classes.
- e) Obs.: definir os métodos e os atributos que devem ser implementados!



Muito Obrigada!

Profª. Angela Abreu Rosa de Sá, Drª.

Contato: angelaabreu@gmail.com