Universidade Federal de Ouro Preto

CSI032 - Programação de Computadores II

Classes Abstratas

Professor: Dr. Rafael Frederico Alexandre
Contato: rfalexandre@decea.ufop.br
Colaboradores: Eduardo Matias Rodrigues
Contato: eduardo.matias@aluno.ufop.edu.br

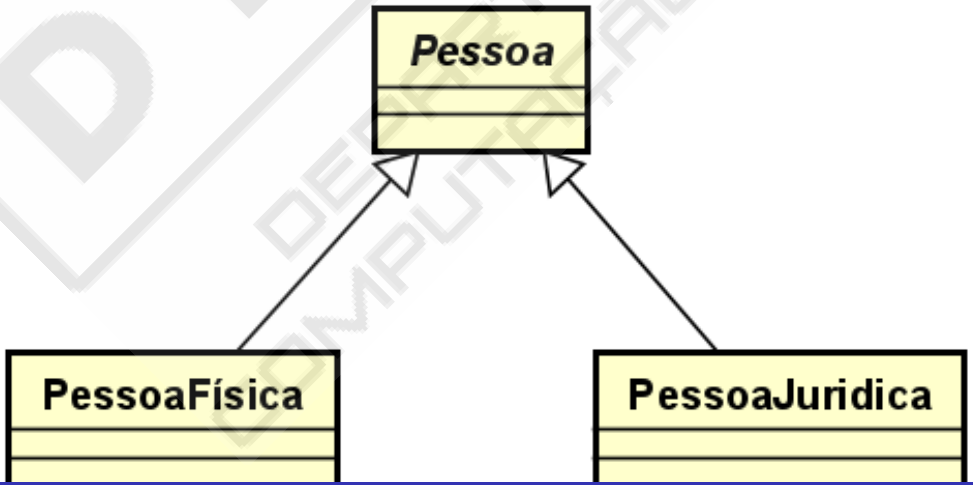
Tópicos

Tópicos

Classe Abstrata

Introdução

1. Em muitos casos há a necessidade de declararmos uma classe de maneira geral, mas que não faz sentido possuir uma instância, exemplo:
   1. Um sistema em que são armazenadas as informações de pessoas físicas e jurídicas. Não faz sentido para esse sistema existir uma instância da classe Pessoa, logo, podemos torná-la **abstrata**.
   2. Obs: no diagrama de classes, uma classe abstrata pode ser representada como uma classe com o nome em *itálico*.



Classe Abstrata

Sintaxe

1. Podemos tornar uma classe abstrata utilizando a palavra reservada **abstract**.

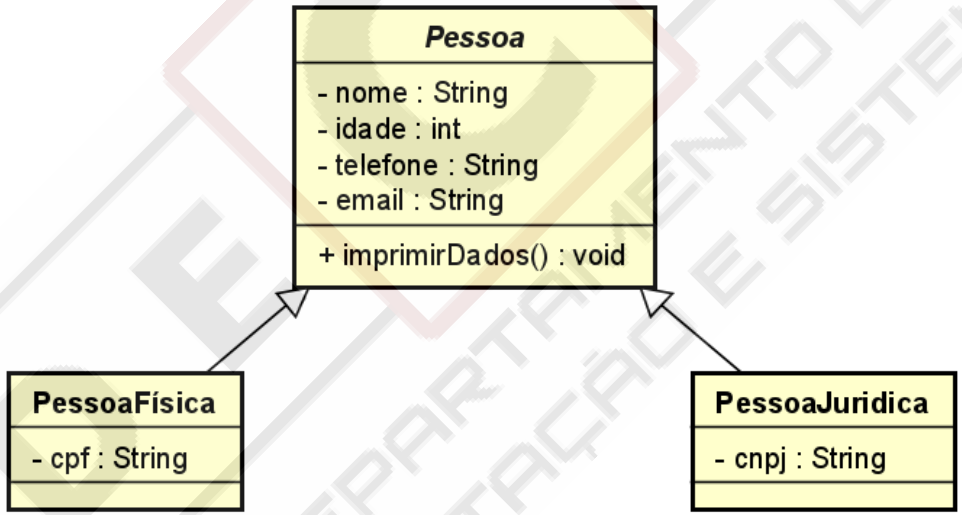
```
public abstract class Pessoa { /* ... */ }
```

1. Agora a classe Pessoa não pode ser instânciada.

Classe Abstrata

Utilidade: herança

1. Se uma classe abstrata não pode ser instânciada, então qual a sua utilidade?
   1. Herança dos atributos e métodos;



1. Os atributos (nome, idade, telefone, email) e métodos (imprimirDados()) são herdados por PessoaFisica e PessoaJuridica.

Tópicos

Classe Abstrata

Utilidade: polimorfismo

1. Podemos também utilizar uma classe abstrata para declarar variáveis que recebem a referência a um objeto da classe filha;

```
Pessoa joao = new PessoaFisica();
Pessoa cocaCola = new PessoaJuridica();
```

Classe Abstrata

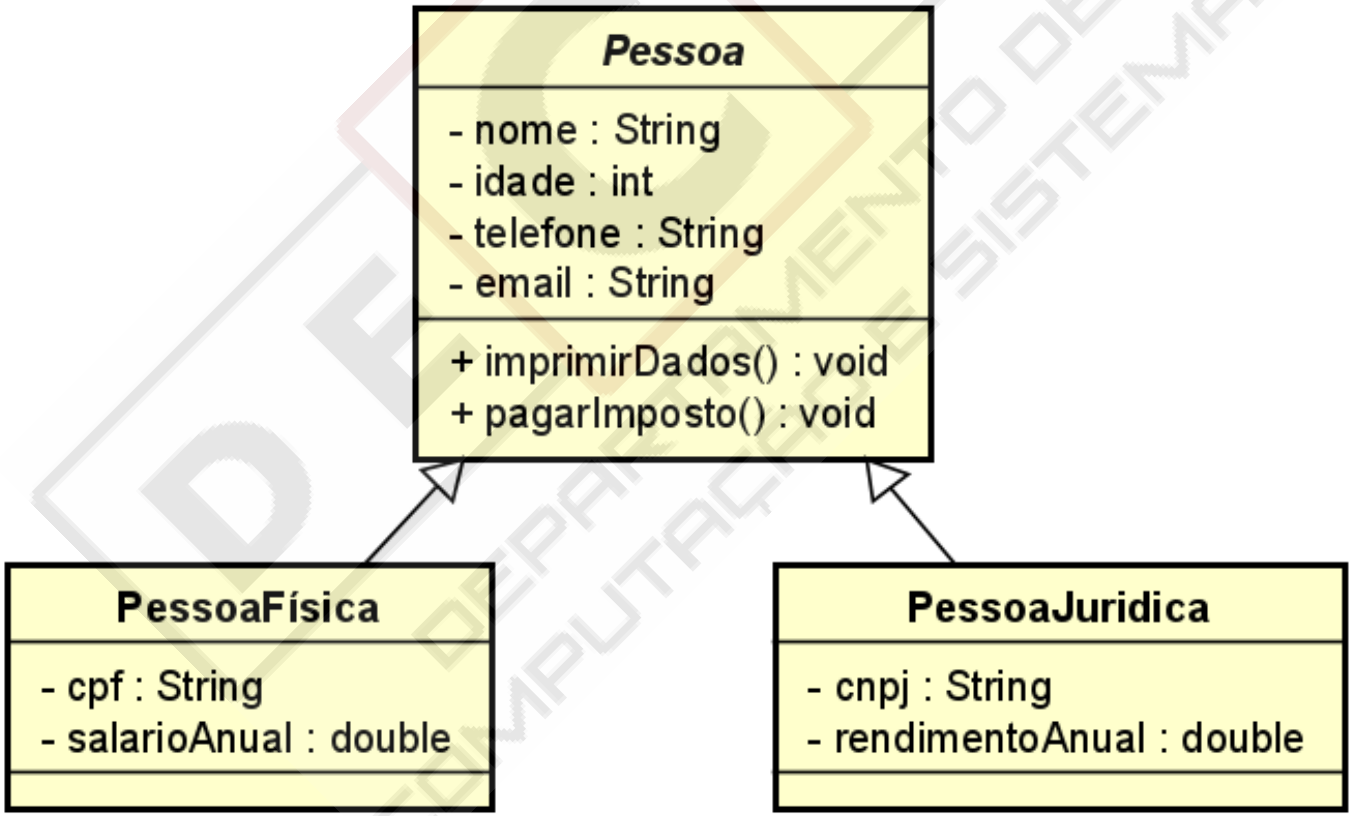
Pagar impostos

1. Supondo agora, que em nosso sistema de pessoas físicas e jurídicas, cada pessoa deverá pagar seus impostos:
   1. Uma pessoa física deve pagar 10% do seu salário anual em impostos;
   2. Uma empresa deve pagar 7% do seu rendimento anual para o governo.
2. Como resolver este problema? Vamos a algumas possíveis soluções...

Classe Abstrata

Solução 1

1. Implementar um método concreto pagarImposto() na classe Pessoa:



Classe Abstrata

Solução 1, continuação ...

1. Método pagarImposto() na classe Pessoa:

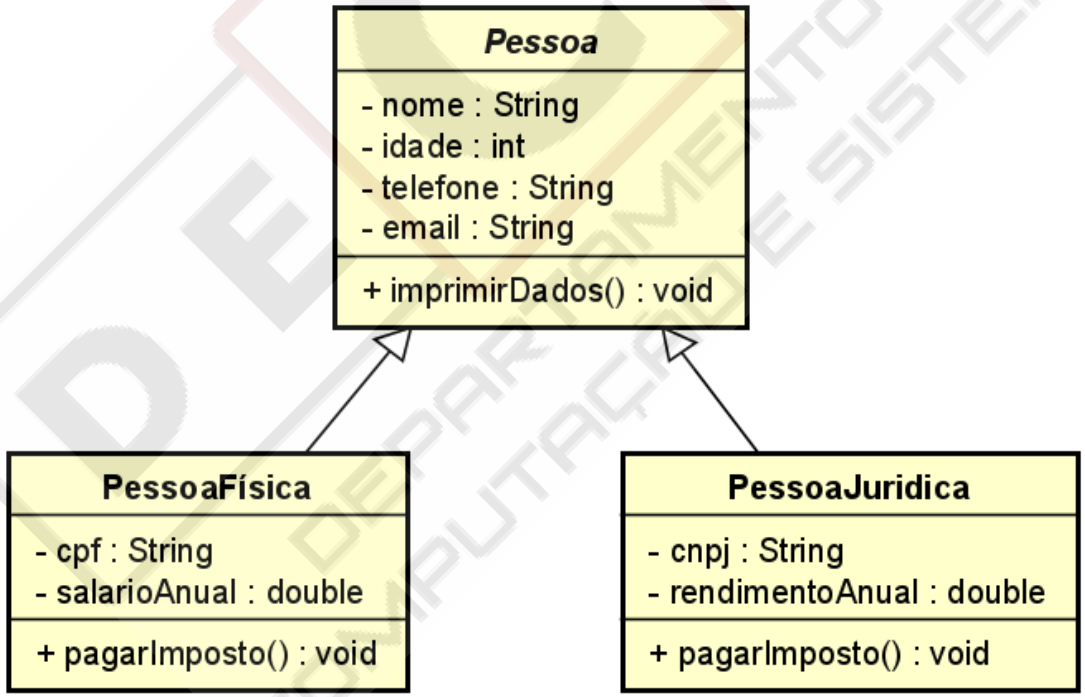
```
public void pagarImposto() {
    System.out.println("Pagando imposto ...");
}
```

1. O método pagarImposto() implementado na classe Pessoa deve ser reescrito nas classes PessoaFisica e PessoaJuridica para satisfazer as condições do problema.
2. Logo, temos uma implementação desnecessária na classe Pessoa.

Classe Abstrata

Solução 2

1. Uma outra solução seria não implementar o método pagarImposto() na classe Pessoa e implementá-lo apenas nas classes filhas:



Classe Abstrata

Solução 2, continuação ...

1. Implementação do método pagarImposto() na classe PessoaFisica:

```
@Override
public void pagarImposto() {
    double imposto = this.salarioAnual * 0.1;
    System.out.println("Pessoa física pagando imposto ...");
    System.out.println("Imposto pago, no valor de " + imposto + " reais");
}
```

1. Saída do programa supondo um salário anual de R$ 100000.

```
Pessoa física pagando imposto ...
Imposto pago, no valor de 10000.0 reais
```

Classe Abstrata

Solução 2, continuação ...

1. Implementação do método pagarImposto() na classe PessoaJuridica:

```
@Override
public void pagarImposto() {
    double imposto = this.rendimentoAnual * 0.07;
    System.out.println("Pessoa jurídica pagando imposto ...");
    System.out.println("Imposto pago, no valor de " + imposto + " reais");
}
```

1. Saída do programa supondo um rendimento anual de R$ 99999999.

```
Pessoa jurídica pagando imposto ...
Imposto pago, no valor de 6999999.930000001 reais
```

Classe Abstrata

Solução 2, continuação ...

1. Mas se tentarmos chamar o método pagarImposto() com uma referência à Pessoa, teremos um erro:

```
Pessoa joao = new PessoaFisica("João", 48, "384155290", "joao@hotmail.com", 100000);
joao.pagarImposto();
```

1. Saída do programa:
   1. Exception in thread "main"java.lang.RuntimeException: Uncompilable source code - Erroneous sym type: testes.Pessoa.pagarImposto at testes.Testes.main(Testes.java:20)

Classe Abstrata

Métodos abstratos

Caelum [Caelum, 2019]

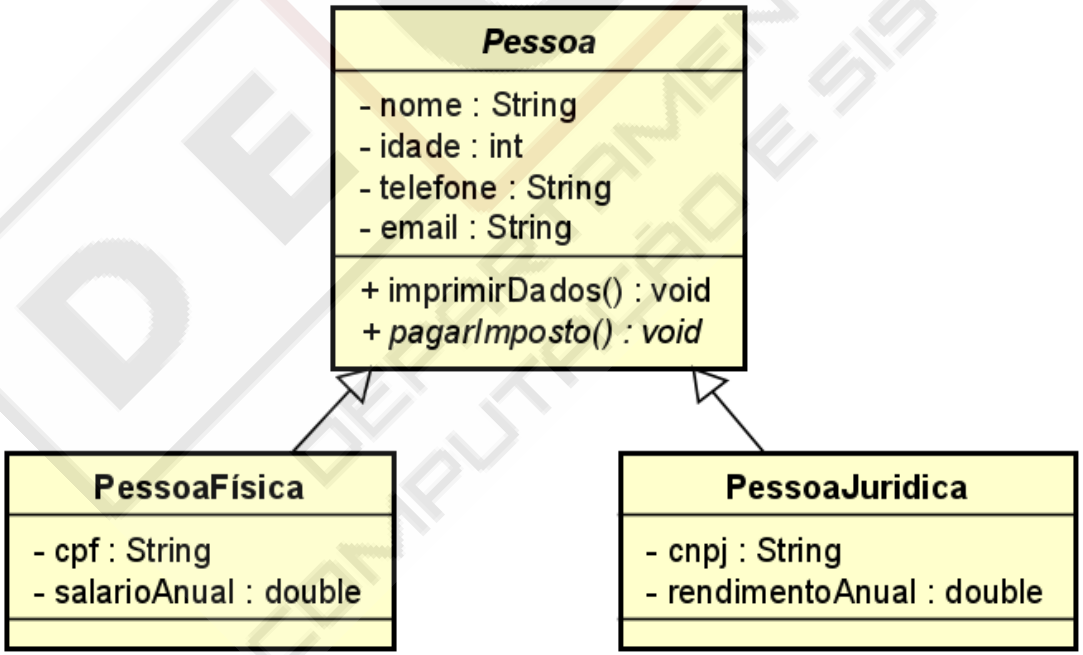
Existe um recurso em Java que nos permite escrever que determinado método **sempre** será reescrito pelas classes filhas, isto é, um método **abstrato**;

1. Um método abstrato não possui corpo e a palavra reservada **abstract** é usada para torná-lo abstrato.
2. Toda classe que possui um método abstrato **deve** também ser abstrata.

Classe Abstrata

Solução 3

1. Uma terceira alternativa para resolver o problema do pagamento de impostos é criar um método abstrato pagarImposto() na classe Pessoa. No diagrama de classes da UML, um método abstrato pode ser representado colocando sua assinatura em *itálico*.



Classe Abstrata

Solução 3, continuação ...

1. Método pagarImposto() na classe Pessoa:

```
public abstract void pagarImposto();
```

1. As classes PessoaFisica e PessoaJurica são obrigadas a reescreverem o método pagarImposto() e cada uma implementa o método seguindo as regras de negócio para cada tipo de pessoa;
2. Segue um exemplo de implementação...

Tópicos

Classe Abstrata

Classe Pessoa

```java
public abstract class Pessoa {

    private String nome;
    private int idade;
    private String telefone;
    private String email;

    public Pessoa(String nome, int idade, String telefone, String email) {
        this.nome = nome;
        this.idade = idade;
        this.telefone = telefone;
        this.email = email;
    }

    public void imprimirDados() {
        System.out.println("Nome: " + this.nome);
        System.out.println("Idade: " + this.idade + " anos");
        System.out.println("Telefone: " + this.telefone);
        System.out.println("e-mail: " + this.email);
    }

    // Implementação nas classes filhas
    public abstract void pagarImposto();
}
```

Classe Abstrata

Classe PessoaFisica

```
public class PessoaFisica extends Pessoa {

    private String cpf;
    private double salarioAnual;

    public PessoaFisica(String cpf, double salarioAnual, String nome, int idade, String telefone, S
        super(nome, idade, telefone, email);
        this.cpf = cpf;
        this.salarioAnual = salarioAnual;
    }

    @Override
    public void pagarImposto() {
        double imposto = this.salarioAnual * 0.1;
        System.out.println("Pessoa física pagando imposto...");
        System.out.println("Imposto pago, no valor de " + imposto + " reais\n");
    }

    @Override
    public void imprimirDados() {
        super.imprimirDados();
        System.out.println("CPF: " + this.cpf);
        System.out.println("Salário anual: " + this.salarioAnual);
    }
```

Classe Abstrata

Classe PessoaJuridica

```
public class PessoaJuridica extends Pessoa {

    private String cnpj;
    private double rendimentoAnual;

    public PessoaJuridica(String cnpj, double rendimentoAnual, String nome, int idade, String telef
        super(nome, idade, telefone, email);
        this.cnpj = cnpj;
        this.rendimentoAnual = rendimentoAnual;
    }

    @Override
    public void pagarImposto() {
        double imposto = this.rendimentoAnual * 0.07;
        System.out.println("Pessoa jurídica pagando imposto...");
        System.out.println("Imposto pago, no valor de " + imposto + " reais\n");
    }

    @Override
    public void imprimirDados() {
        super.imprimirDados();
        System.out.println("CNPJ: " + this.cnpj);
        System.out.println("Rendimento anual: " + this.rendimentoAnual);
    }
}
```

Classe Abstrata

Classe Principal

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

    ArrayList<Pessoa> pessoas = new ArrayList<>();

    PessoaFisica joao = new PessoaFisica("117748521089", 100000,
                                         "João Rodrigues", 58, "38475214", "joao@gmail.com");

    PessoaFisica maria = new PessoaFisica("117748521089", 50000,
                                          "Maria Justino", 50, "38475214", "maria@gmail.com");

    PessoaJuridica cocaCola = new PessoaJuridica("45.997.418/0001-53", 985215.1,
                                                 "Coca-Cola", 128, "0800251412",
                                                 "coca_cola@gmail.com");

    pessoas.add(maria);
    pessoas.add(cocaCola);
    pessoas.add(joao);

    for (Pessoa p : pessoas) {
        p.imprimirDados();
        p.pagarImposto();
    }
}
```

Classe Abstrata

Saída do programa

```
Nome: Maria Justino
Idade: 50 anos
Telefone: 38475214
e-mail: maria@gmail.com
CPF: 117748521089
Salário anual: 50000.0
Pessoa física pagando imposto...
Imposto pago, no valor de 5000.0 reais

Nome: Coca-Cola
Idade: 128 anos
Telefone: 0800251412
e-mail: coca_cola@gmail.com
CNPJ: 45.997.418/0001-53
Rendimento anual: 985215.1
Pessoa jurídica pagando imposto...
Imposto pago, no valor de 68965.057 reais
```

Classe Abstrata

Saída do programa, continuação...

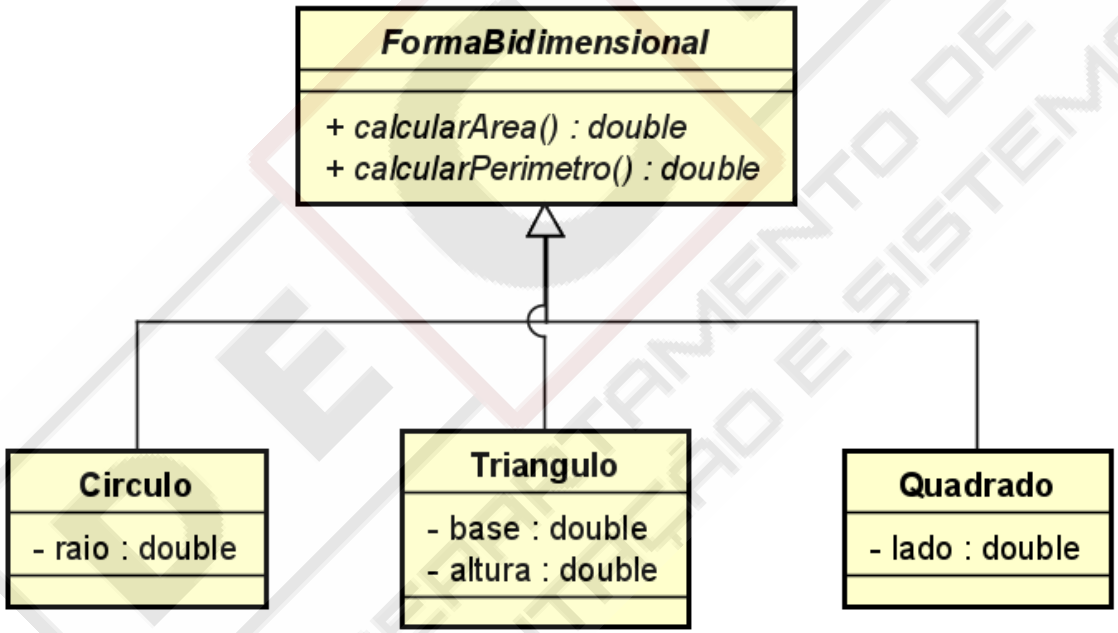
```
Nome: João Rodrigues
Idade: 58 anos
Telefone: 38475214
e-mail: joao@gmail.com
CPF: 117748521089
Salário anual: 100000.0
Pessoa física pagando imposto...
Imposto pago, no valor de 10000.0 reais
```

Classe Abstrata

Forma Geométrica Bidimensional

1. Crie uma classe FormaBidimensional, cada forma bidimensional deve ser capaz de calcular sua área e seu perímetro;
2. Crie classes que representem: círculos, triângulos e quadrados;
3. Crie um driver para testar sua aplicação. No teste deve conter um ArrayList de formas bidimensionais e para cada forma deverá ser impresso o tipo (Círculo, Quadrado ou Triângulo) e sua respectiva área e volume.

Exemplo
Diagrama de Classes



Exemplo
Classe FormaBidimensional

```
public abstract class FormaBidimensional {

    public abstract double calcularArea();

    public abstract double calcularPerimetro();

}
```

Exemplo
Classe Circulo

```
public class Circulo extends FormaBidimensional {

    private double raio;

    public Circulo(double raio) {
        this.raio = raio;
    }

    @Override
    public double calcularArea() {
        return Math.PI*Math.pow(this.raio, 2);
    }

    @Override
    public double calcularPerimetro() {
        return 2*Math.PI*this.raio;
    }

}
```

Exemplo
Classe Quadrado

```
public class Quadrado extends FormaBidimensional {

    private double lado;

    public Quadrado(double lado) {
        this.lado = lado;
    }

    @Override
    public double calcularArea() {
        return this.lado * this.lado;
    }

    @Override
    public double calcularPerimetro() {
        return 4 * this.lado;
    }
}
```

Exemplo
Classe Triangulo

```
public class Triangulo extends FormaBidimensional{

    private double base;
    private double altura;

    public Triangulo(double base, double altura) {
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return (this.base*this.altura)/2;
    }

    @Override
    public double calcularPerimetro() {
        return 3*this.base; // Triangulo equilatero
    }
```

Exemplo
Main

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

    ArrayList<FormaBidimensional> formas = new ArrayList<>();

    Circulo c = new Circulo(10);
    Quadrado q = new Quadrado(4);
    Triangulo t = new Triangulo(2, 5);

    formas.add(c);
    formas.add(q);
    formas.add(t);

    for (int i = 0; i< formas.size(); i++) {
        FormaBidimensional corrente = formas.get(i);
        System.out.println("Tipo: " + corrente.getClass().getSimpleName());
        System.out.println("Área: " + corrente.calcularArea());
        System.out.println("Perímetro: " + corrente.calcularPerimetro() + "\n");
    }

}
```

Exemplo
Saída do programa

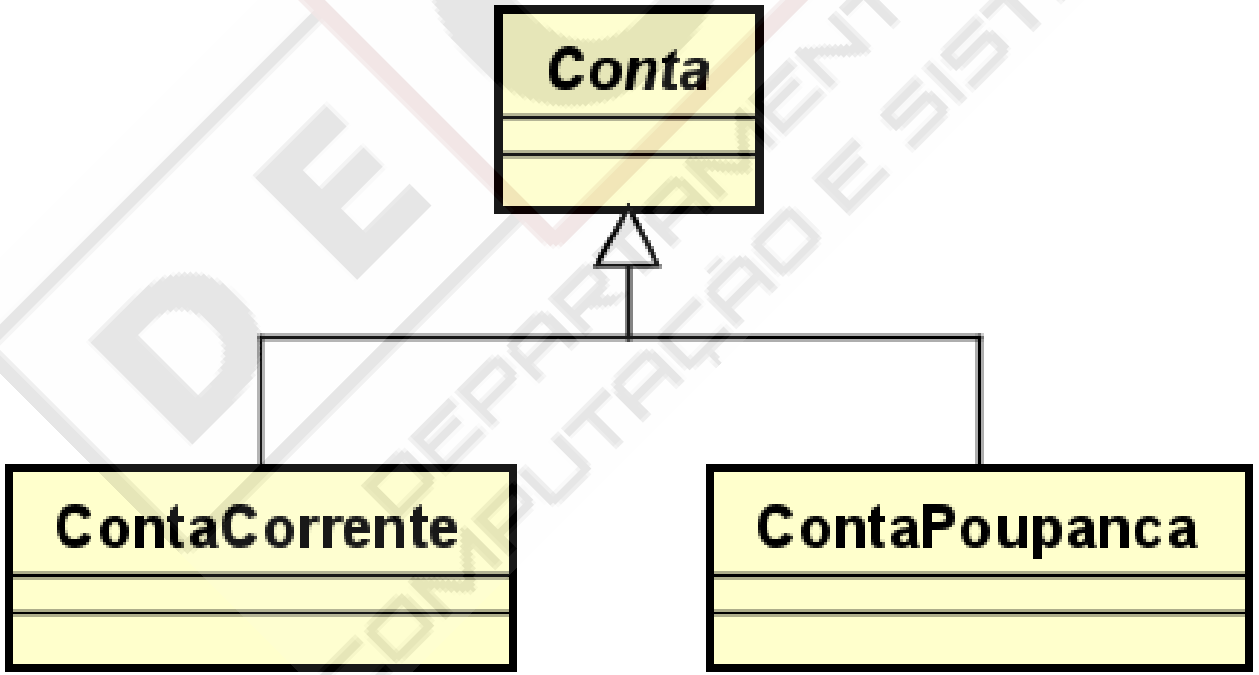
```
Tipo: Circulo
Área: 314.1592653589793
Perímetro: 62.83185307179586

Tipo: Quadrado
Área: 16.0
Perímetro: 16.0

Tipo: Triangulo
Área: 5.0
Perímetro: 6.0
```

Tópicos

Classe Abstrata

Exercício [Malaquias, 2015]

1. Crie uma hierarquia de herança que um banco possa utilizar para represetar dois tipos de conta: poupança e conta corrente. Todos os clientes deste banco podem depositar e sacar dinheiro de suas contas.



Classe Abstrata

Exercício [Malaquias, 2015], continuação ...

1. A classe **Conta** deve possuir um atributo que represente o saldo da conta. Este atributo deve ser inicializado através de um construtor parametrizado que valide o valor enviado como parâmetro. Devem ser criados métodos para mostrar o saldo, para crédito e para débito na conta. Crie um getter e um setter para o atributo.
2. A classe **ContaPoupanca** deve possuir um atributo relacionado à variação (rendimento), com métodos getter, setter e construtor. Crie também um método calcularRendimento(), que informa o valor do saldo multiplicado pela taxa de rendimento.

Classe Abstrata

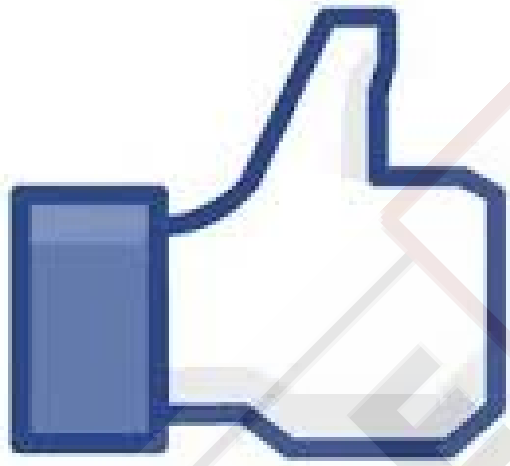
Exercício [Malaquias, 2015], continuação ...

1. A classe **ContaCorrente** deve incluir um atributo que represente a taxa cobrado por cada transação de crédito/débito, com getter, setter e construtor. Sobrescreva os métodos de crédito e débito para descontar o valor de tal taxa a cada transação bem sucedida.

Classe Abstrata

Exercício [Malaquias, 2015], continuação ...

1. Adicione o comportamento polimórfico conforme descrito:
   1. Os métodos de crédito, débito e saldo em conta devem ser abstratos na classe base;
   2. Crie um drive com um vetor de deferências para Conta, cada referência deve ser relativa a um dos objetos de cada classe concreta da hierarquia.
      1. Invoque os métodos de débito e crédito de cada objeto apontado.
   3. Percorra o vetor e determine em tempo de execução qual é o tipo de cada conta:
      1. Se for uma poupança, calcule seu rendimento através do método calculaRendimento();
      2. Se for um conta corrente, apenas mostre seu saldo.



MUITO OBRIGADO!!!

Referências Bibliográficas I

Caelum (2019).
Java e orientação a objeto.
https://www.caelum.com.br/download/caelum-java-objetos-fj11.pdf.

Malaquias, J. R. (2015).
Programação orientada a objetos.
http://www.decom.ufop.br/romildo/2015-2/bcc221/x1-1300012.