PROGRAMAÇÃO ORIENTADA A OBJETOS

AULA 11 – PILARES DA PROGRAMAÇÃO ORIENTADA A OBJETOS

WALTER TRAVASSOS SARINHO

@PROF.WALTERTRAVASSOS

WALTER.TRAVASSOS@SECTRAS.EDU.BR

Dúvidas sobre a aula passada

Mapa de Estudos

AULA 01
Paradigmas de Programação e história do Java.
IDEs e Hello Universe

AULA 02
Manipulação de datas.
Sintaxe da Linguagem Java e variáveis primitivas e de referência.

AULA 03
Visibilidade de variáveis, classe String e casting.

AULA 04
Classes Wrapper.
Operadores e métodos de conversão.

AULA 05
Formatar a saída de dados numérica.
Operador condicional if.
Métodos para comparar Strings

Avaliação 1

Java

Mapa de Estudos

AULA 06
Manipulação de Strings. Operador condicional switch.

AULA 08
Vetores (for each) e matrizes. Classe ArrayList

AULA 10
Construindo métodos e Métodos construtores

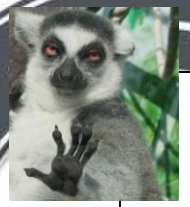
Avaliação 2

AULA 07
Operadores de atribuição compostos. Estruturas de repetição: while, do while e for.

AULA 09
O paradigma Orientado a Objetos

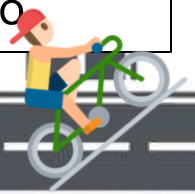
Java

Mapa de Estudos



AULA 12
Classes Abstratas e Interfaces

AULA 11
Pilares da POO: Abstração, Encapsulamento, Herança e Polimorfismo



Avaliação 3

Java

Aula de Hoje

Apresentar os pilares da programação orientada o objetos:

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

Introdução

- Em orientação a objeto, uma classe é uma estrutura que abstrai um conjunto de objetos com características similares.
- Uma classe define o **comportamento** de seus objetos através de **métodos** e os **estados** possíveis destes objetos através de **atributos (variáveis de instância)**.
- Em outros termos, uma classe descreve **os serviços providos** por seus objetos e **quais informações** eles podem armazenar.

Introdução

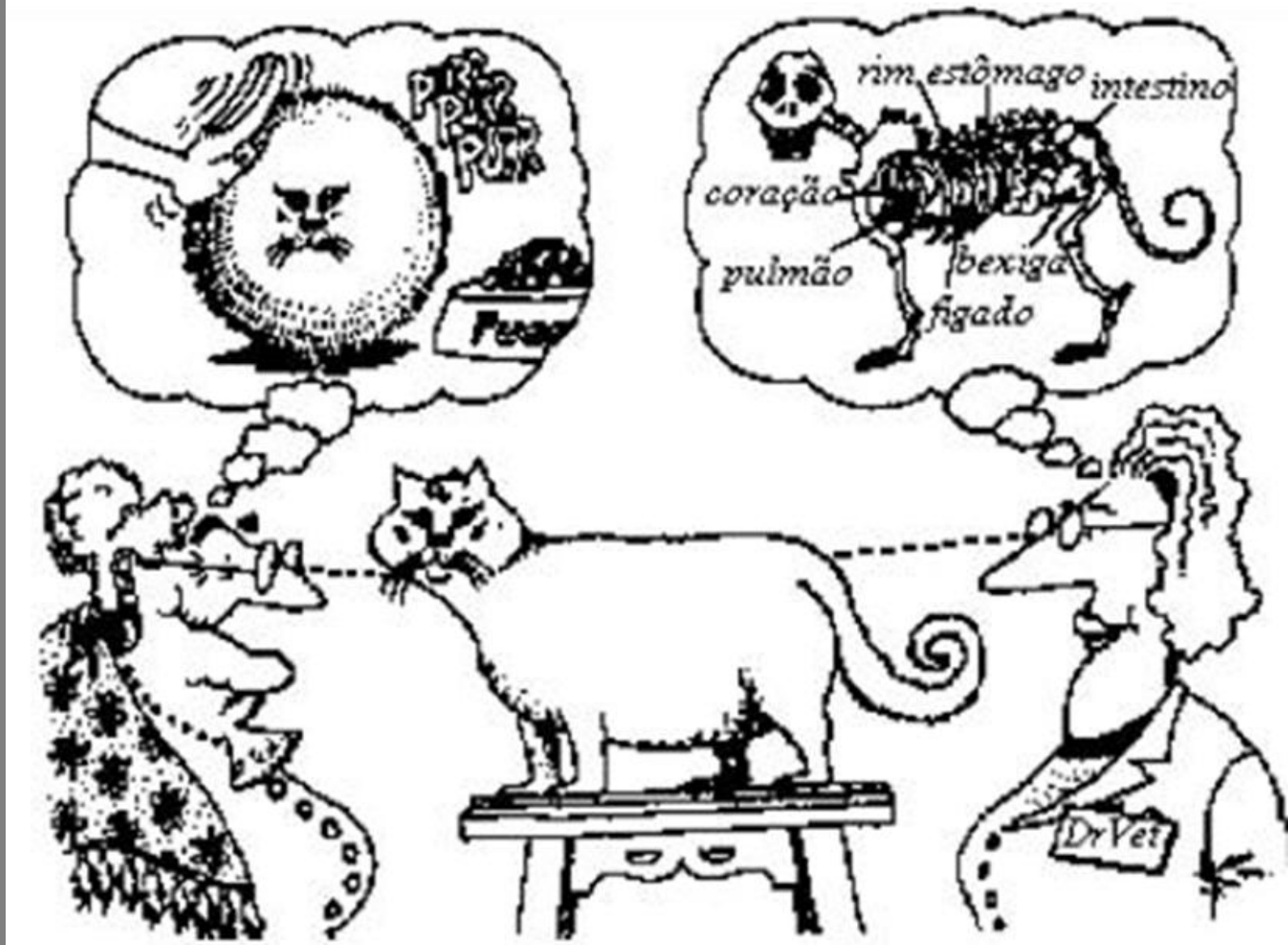
- O conceito de classes não é diretamente suportado em **todas** as linguagens. Este conceito é necessário para que uma linguagem seja orientada a objetos.
- Alguns autores mencionam que a programação orientada a objeto tem **três** pilares: **Encapsulamento, Herança e Polimorfismo.**

- Outras referências citam também a **Abstração** como sendo o primeiro pilar da POO.

Pilares da Programação Orientada a Objetos

1. Abstração

É possível perceber diferentes níveis de abstração a partir da visão da dona do gato e de uma veterinária.



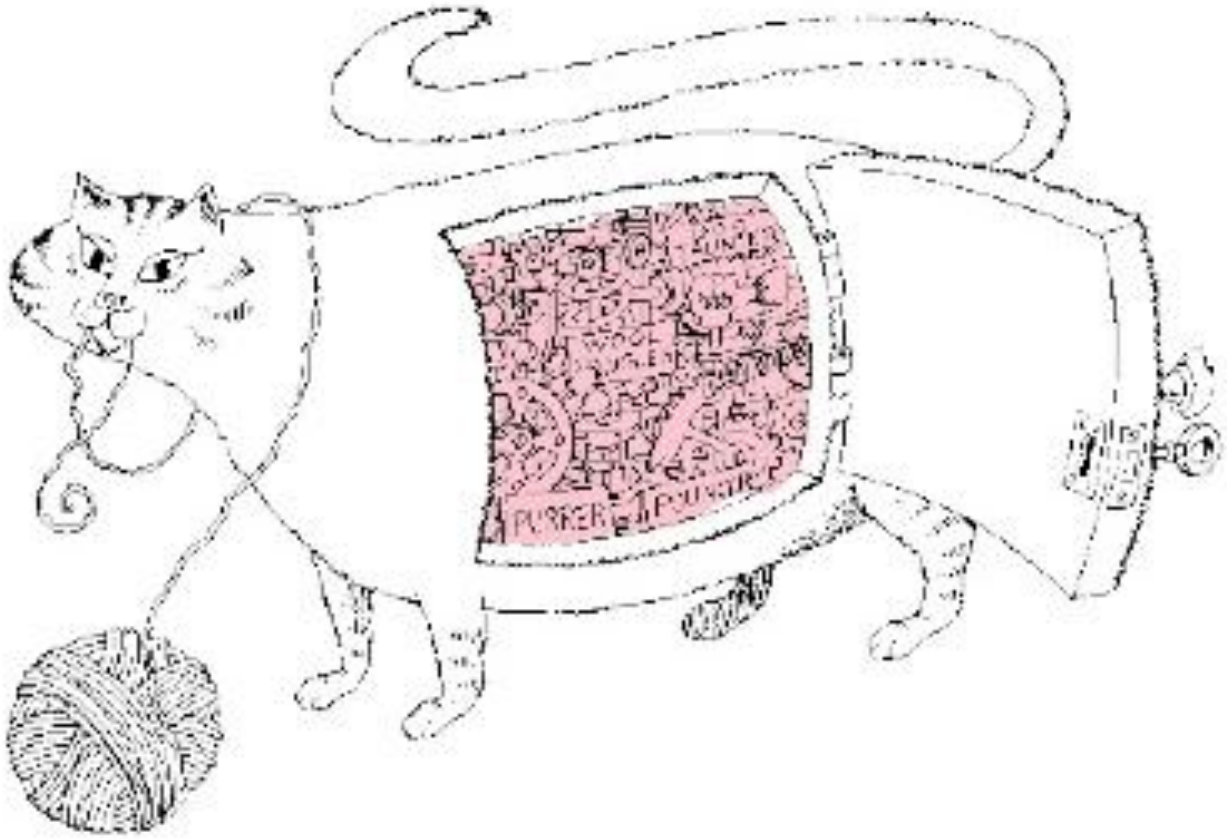
1. Abstração

Nesta etapa “imaginamos” o nosso objeto. Aqui definimos a identidade, as propriedades e seus métodos (eventos):

- **Identidade**: O nome do objeto a ser criado.
- **Propriedades**: São as características do objeto.
- **Métodos**: São os eventos, as ações que esse objeto irá executar.

- É a representação de algo real. Na abstração analisamos o que é relevante conter em um objeto.

2. Encapsulamento



O que é encapsulamento?

É separar a interface de um objeto dos detalhes de seu funcionamento interno.

Definição 1

- Encapsulamento vem de encapsular, que em programação orientada a objetos significa **juntar o programa em partes,** o mais isoladas possível.
- A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações.

Definição 2

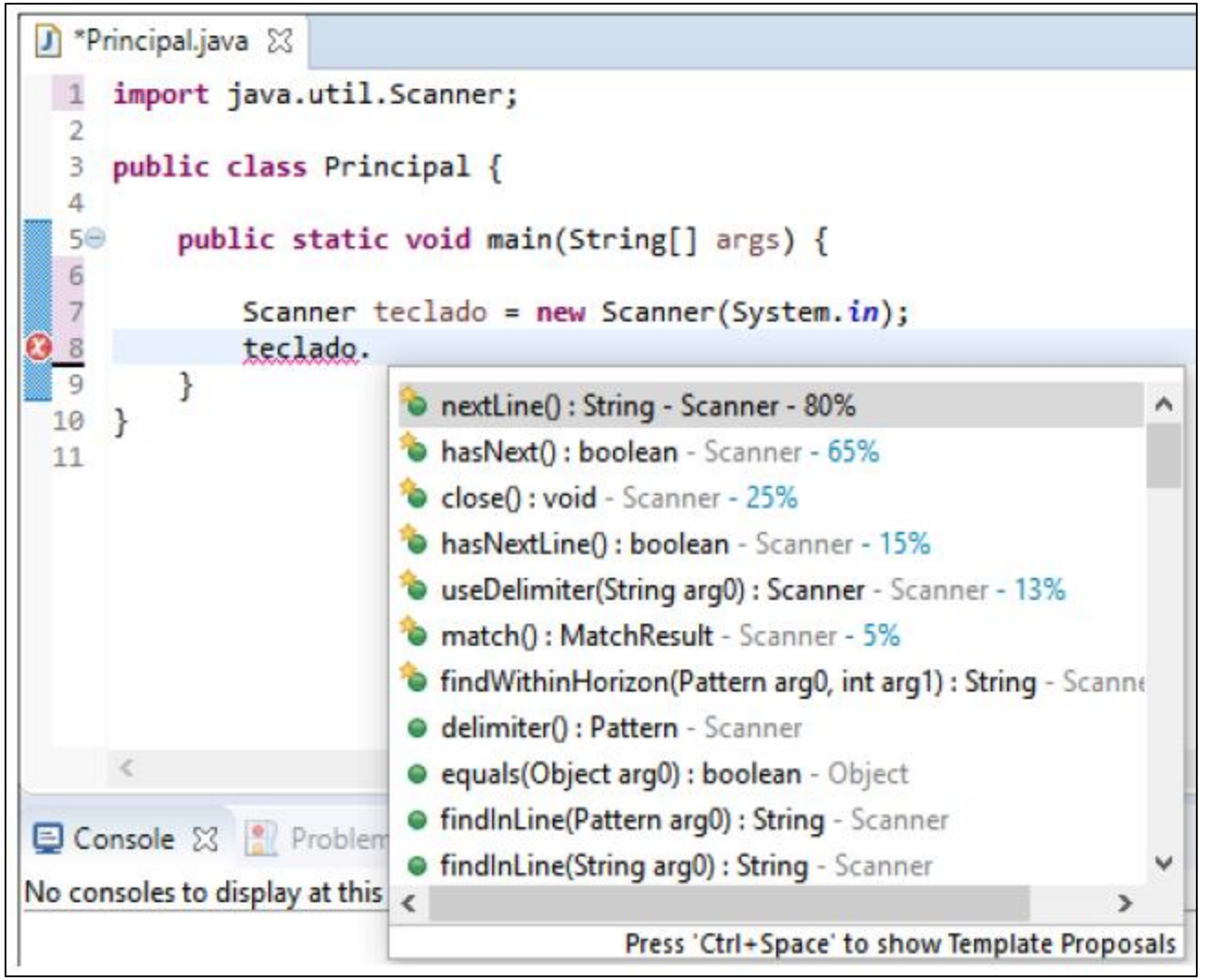
- Encapsulamento é a técnica que faz com que **detalhes internos do funcionamento** dos **métodos** de uma classe permaneçam ocultos para os objetos.
- Por conta dessa técnica, o conhecimento a respeito da implementação interna da classe é desnecessário do ponto de vista do objeto, uma vez que isso passa a ser responsabilidade dos métodos internos da classe.

Encapsular tem a ver com...

- Combinar variáveis de instância e métodos de uma classe.
- Esconder os membros (privados) de uma classe.
- Esconder como funcionam os métodos de uma classe.

Encapsulamento

- O acesso às informações encapsuladas em um objeto se dá a partir da sua **interface de classe**.
- **Interface de classe** = Conjunto de **métodos públicos** de uma classe.



Encapsulamento

Quem utiliza os recursos da **classe** só precisa tomar conhecimento **do que ela é capaz de fazer**, e não necessariamente como ela faz.

Por exemplo:

- Você sabe o que o método **nextInt()** da classe **Scanner** faz. Mas você sabe como ele foi codificado?
- O fato de não conhecer sua codificação impossibilitou seu uso?

COMO

encapsulamos uma classe?

Com os Modificadores de Acesso public e private

Marque suas variáveis de instância com *private* e forneça métodos de captura e configuração *public*, para ter controle sobre o acesso (SIERRA; BATES, 2010).

Resumindo

- Marque as variáveis de instância com *private*.
- Marque os métodos de configuração e captura com *public*.

Exemplo

```
public class Aluno {
        private String nome;
        private int matricula;

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

3. Herança

O que é Herança?

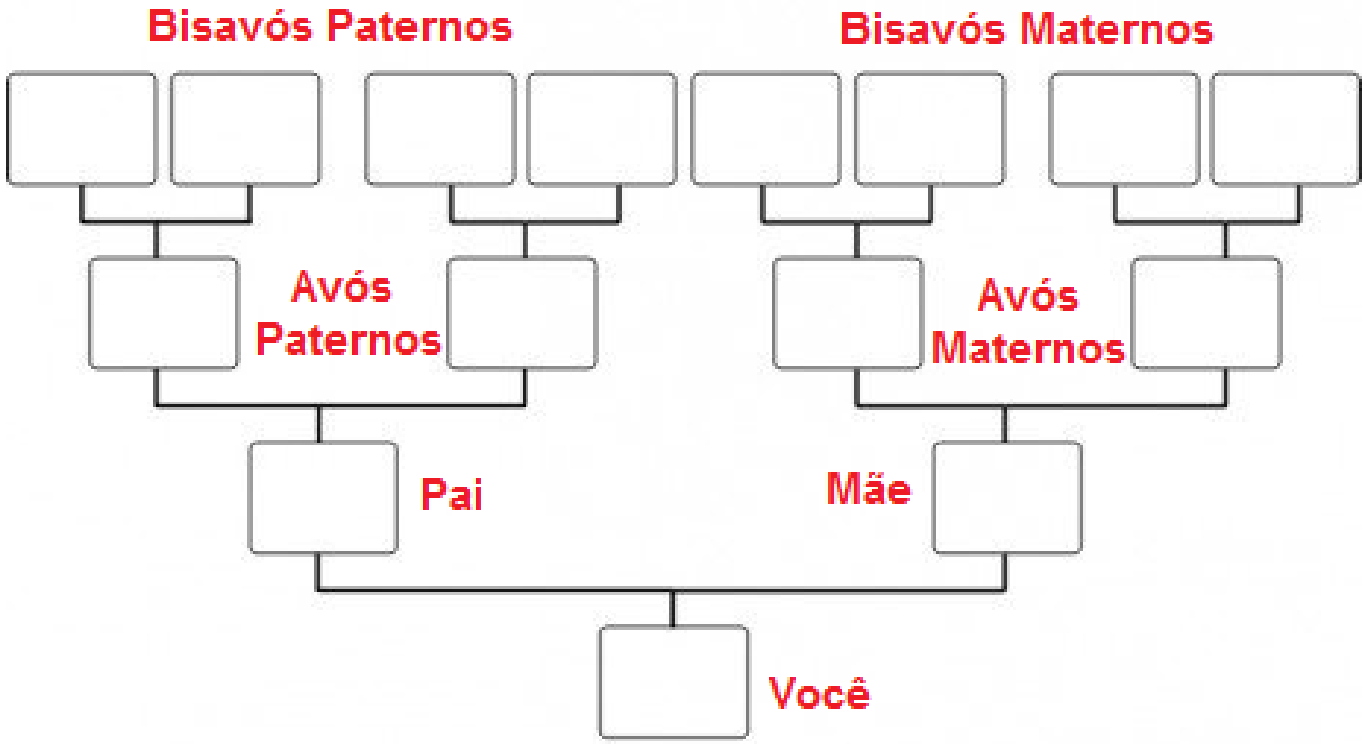
Significa herdar algo de alguém!

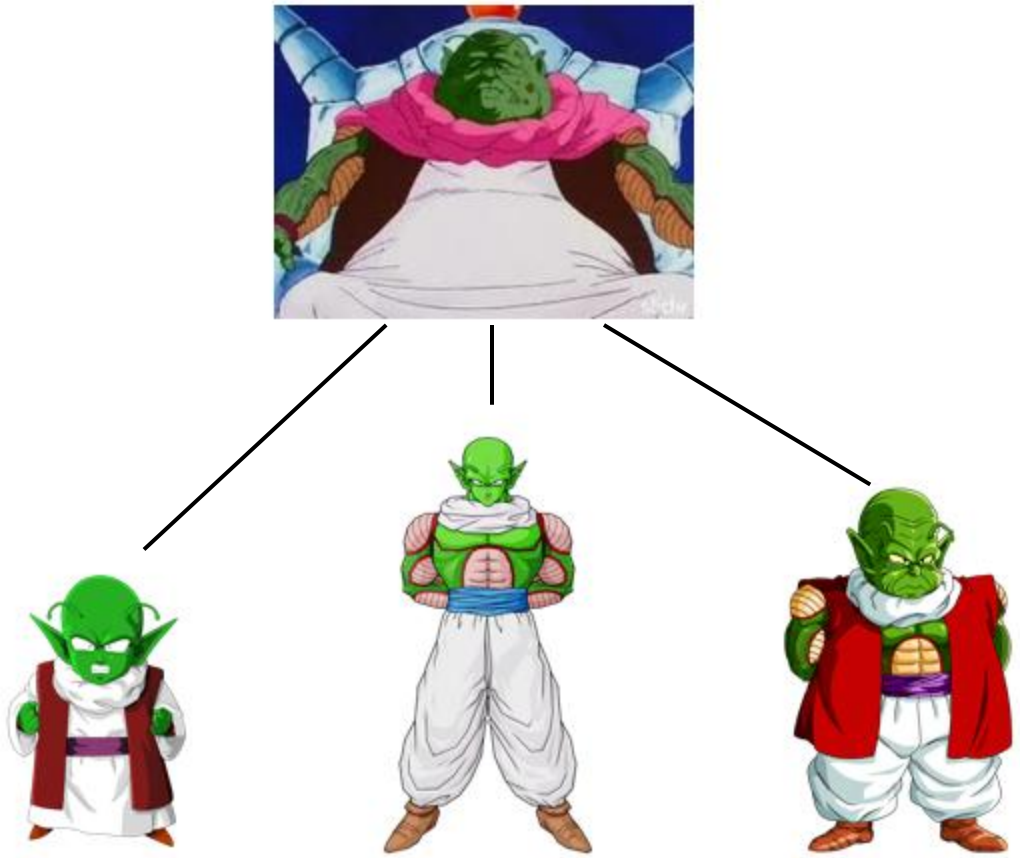
Herança em POO – Classe Object

http://www.devmedia.com.br/java-object-class-entendendo-a-classe-object/30513

- Em POO, significa herdar algo de outra classe. Esse "algo", são os membros da classe definidos com o modificador de acesso que não foram definidos com **private** (**public** e **protected** serão herdados).
- **Object** é a classe raiz da hierarquia de classes do Java.
- Toda classe em Java, mesmo as que não pertencem a API, são filhas de **Object**.
- **Ela é a superclasse de todas as classes, direta ou indiretamente.**
- Toda instância de uma classe implementada em Java é um **Object** e herda os **métodos declarados** nesta superclasse.

A herança pode acontecer a partir de duas classes?





A herança pode acontecer a partir de duas classes? Em Java, Não! Java não permite herança de múltiplas classes.

Java não permite herança múltipla!
Java não permite herança múltipla!
Java não permite herança múltipla!
Java não permite herança múltipla!

Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!
Java nÃo permite heranÇa mÚltipla!

Entre Classes!!!

Herança Múltipla?

- Em Java não é possível que uma **classe** faça herança de múltiplas classes.
- Imagine que uma classe seja herdeira de outras duas classes, entretanto essas duas classes herdadas possuem métodos com a mesma assinatura porém com implementações diferentes.
- Seria algo difícil de lidar, pois como a subclasse (classe filha) saberia qual implementação do método ela deveria usar? Por isso não é possível que uma classe **estenda** de mais de uma classe em Java.

Estender é um neologismo para representar a herança.

A classe filha estende a classe pai.

Está associada a palavra-chave que implementa a herança: extends

Herança

Em Java, podemos herdar variáveis de instância e métodos de uma classe (criar uma **extensão de classe**, **derivar** a classe) a partir da palavra reservada **extends**.

```
public class Pessoa {
  private String nome;

  public Pessoa() {
    nome = “Fulano”;
  }

  public void escreverNome() {
    System.out.println(nome);
  }
}
```

Super classe

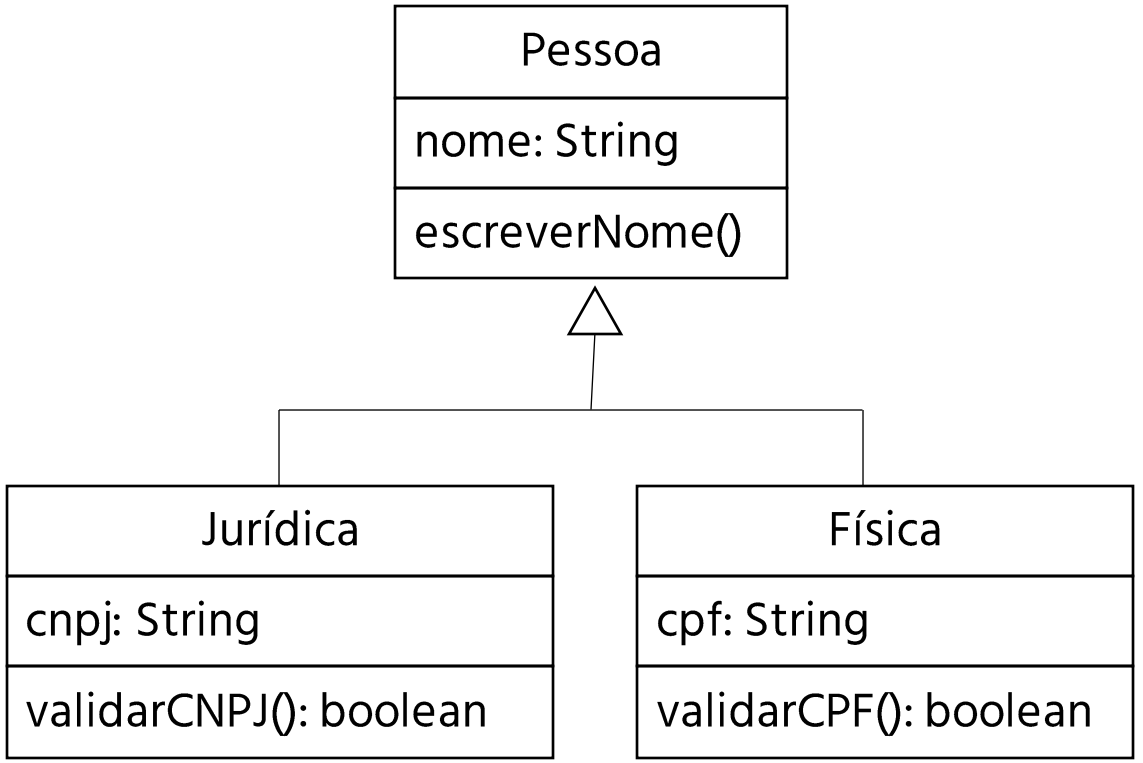
```
public class Física extends Pessoa {
  private String cpf;

  public boolean validarCPF() {

    // Código para validação

    return true;
  }
}
```

Sub classe

Herança no Diagrama de Classes

- Uma subclasse pode herdar todos os componentes de uma classe pai.
- Permite a criação de novas classes sem duplicação de código.
- Existe reaproveitamento de código da classe pai.

Superclasse: Características comuns



Subclasses: Características específicas

Regra de instância de objetos na herança

- Uma variável da superclasse comporta um objeto da subclasse.

Pessoa p = new Juridica();

- No entanto, uma variável da subclasse não comporta um objeto da superclasse.

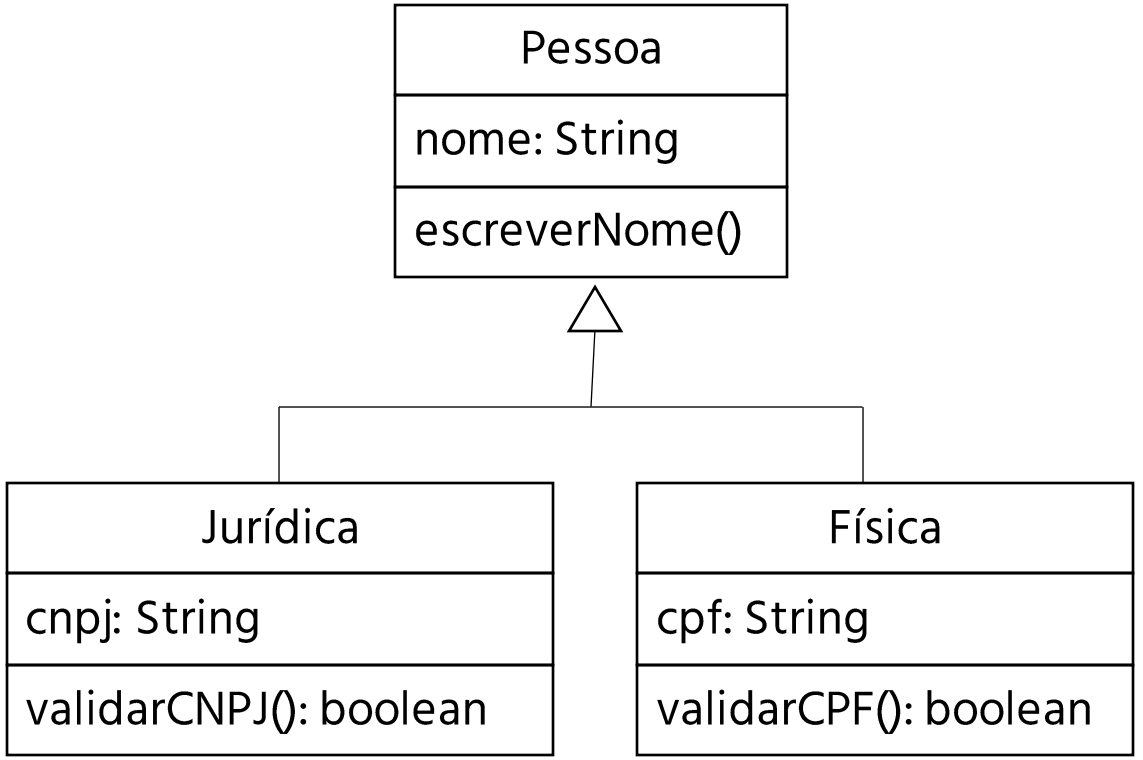
~~**Juridica j = new Pessoa();**~~

- Uma variável da classe **Object** suporta qualquer objeto de outra classe porque está no topo da hierarquia.

```
Object o = new Jurídica();        // ou new Pessoa(); ou new Física();
                                  // ou new QualquerClasse();
```

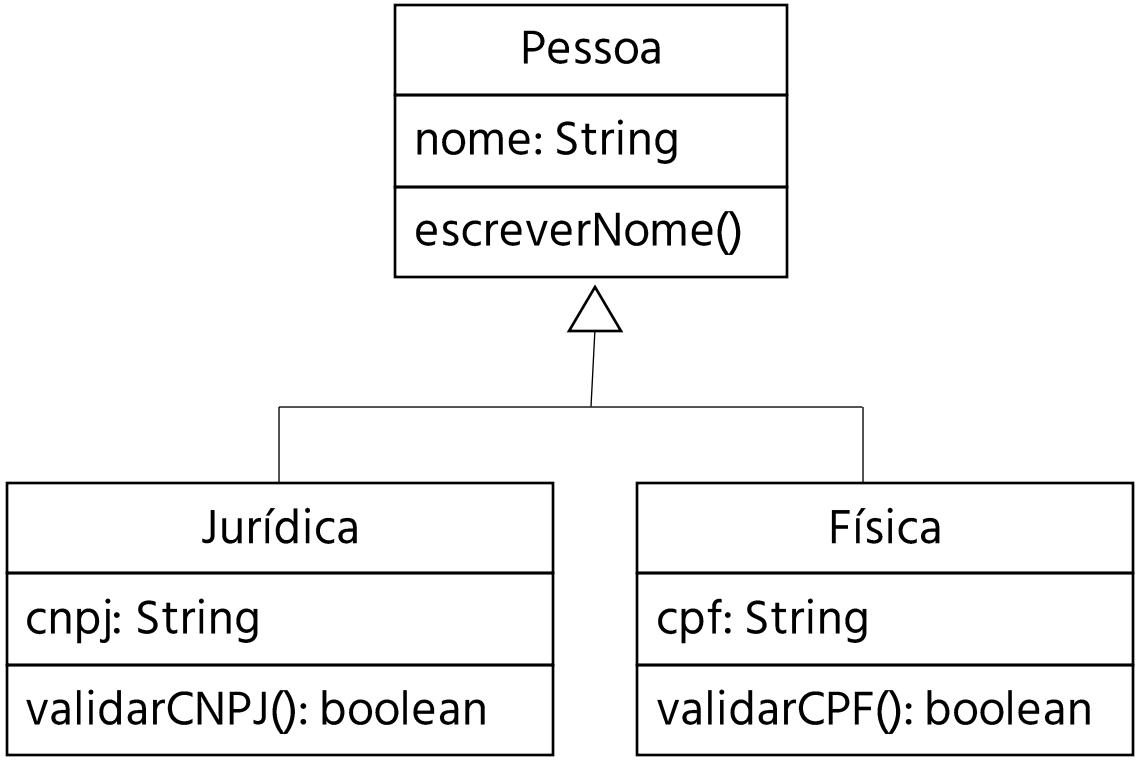
Quais instâncias funcionam?

```
Pessoa p1 = new Pessoa();
Pessoa p2 = new Jurídica();
Pessoa p3 = new Física();
Jurídica p4 = new Jurídica();
Física p5 = new Física();
Jurídica p6 = new Pessoa();
Física p7 = new Pessoa();
Jurídica p8 = new Física();
Física p9 = new Jurídica();
```



Quais instâncias funcionam?

```
Pessoa p1 = new Pessoa();
Pessoa p2 = new Jurídica();
Pessoa p3 = new Física();
Jurídica p4 = new Jurídica();
Física p5 = new Física();
Jurídica p6 = new Pessoa();
Física p7 = new Pessoa();
Jurídica p8 = new Física();
Física p9 = new Jurídica();
```



p6, p7, p8 e p9 não instanciam!

Pensem em "forma"!

- A classe Object é a "maior", mais sem forma (mais genérica, com forma pouco definida)!
- As classes filhas de Object são "menores" (mais especializadas, com formatos mais específicos).
- Assim, uma superclasse (mais genérica) comporta qualquer classe filha (mais especializada).

Resposta

- **p1** até **p5** funcionam! **p6** até **p9** não instanciam!
- **p6** e **p7** não funcionam porque variável da **subclasse** não suporta um objeto da **superclasse.**

```
Jurídica p6 = new Pessoa();
Física p7 = new Pessoa();
```

- **p8** e **p9** são classes irmãs, mas com formatos diferentes. Não se comportam.

```
Jurídica p8 = new Física();
    Física p9 = new Jurídica();
```

Mas existe uma desvantagem...

- É interessante essa possibilidade de programação, mas ao declarar um objeto nesse formato...

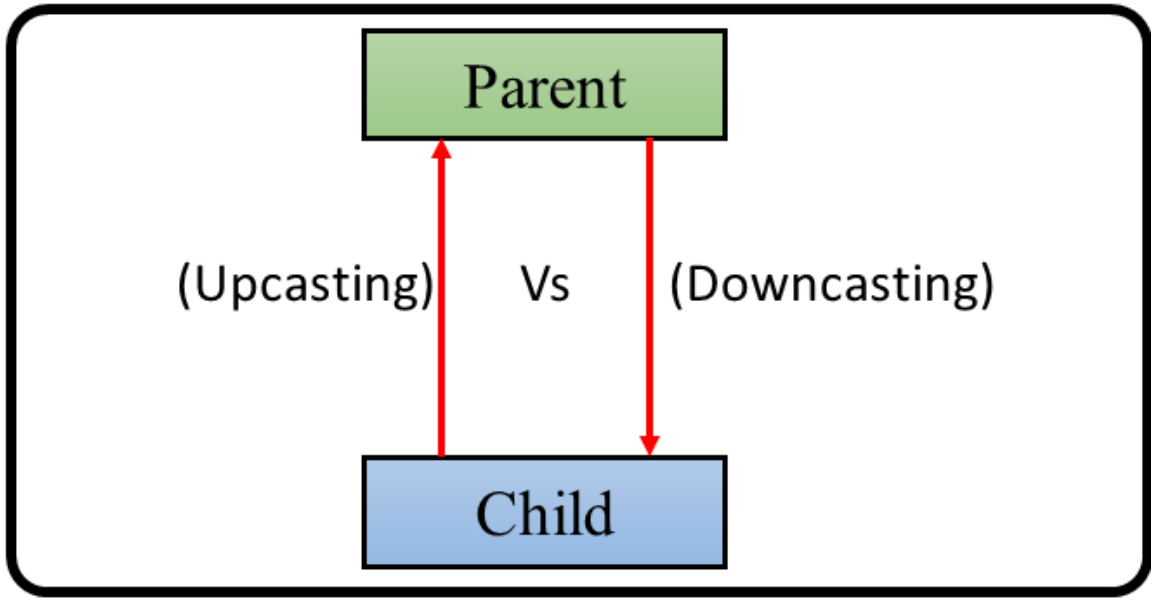
Pessoa p2 = new Jurídica();

Pessoa p3 = new Física();

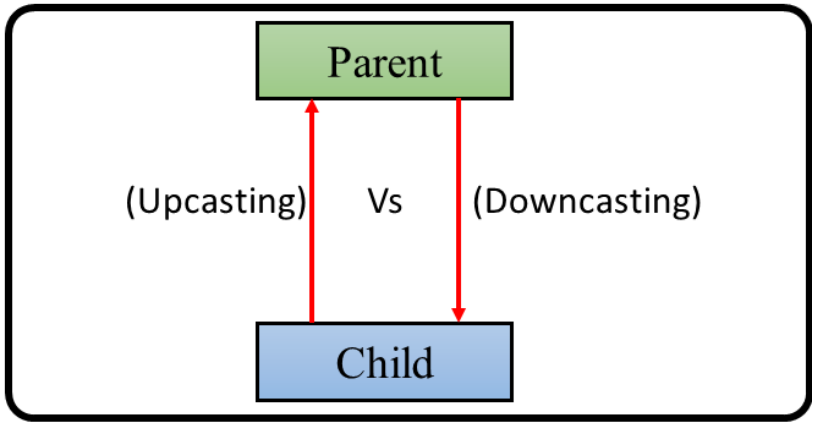
- Só ficará disponível na interface dos objetos p2 e p3 os métodos que também existam na superclasse Pessoa.
- Assim, caso as classes Jurídica e Física possuam métodos específicos (que não estão presentes na classe Pessoa), eles não conseguirão ser acessados neste formato de criação de objeto.

Upcasting e Downcasting

- **Upcasting** ocorre quando você converte um objeto de uma subclasse para uma superclasse.
- **Downcasting** ocorre quando você converte um **objeto de uma** superclasse de volta para uma subclasse.



Upcasting
Conversão Implícita

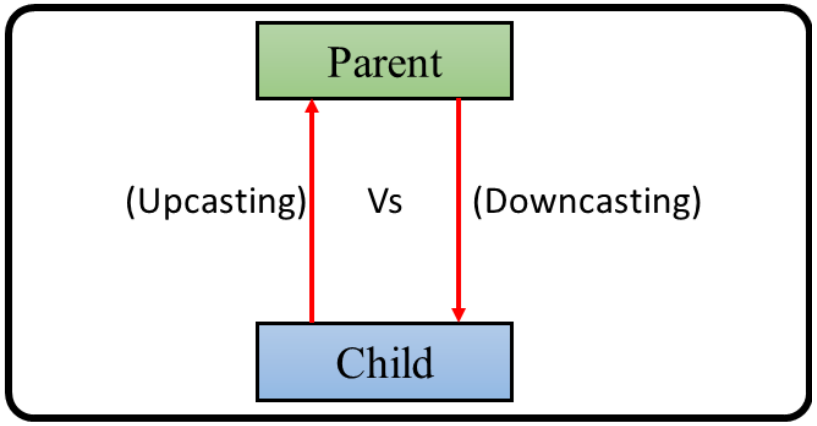


A sintaxe para upcasting é simplesmente atribuir uma instância da subclasse a uma variável da superclasse.

Exemplo:

- Superclasse **obj** = new Subclasse();
- Pessoa p = new Juridica();
- Aqui, **obj** é uma variável do tipo Superclasse, mas está referenciando um objeto da Subclasse.

Downcasting
Conversão Explícita



A sintaxe para downcasting é a utilização de parênteses e o tipo da subclasse.

Exemplo:

- Subclasse obj = (Subclasse) superobj;
- Juridica j = (Juridica) p;
- Aqui, superobj é uma variável do tipo Superclasse, mas é downcasted para o tipo Subclasse.

Leitura obrigatória do capítulo 7 do livro Use a cabeça! Java

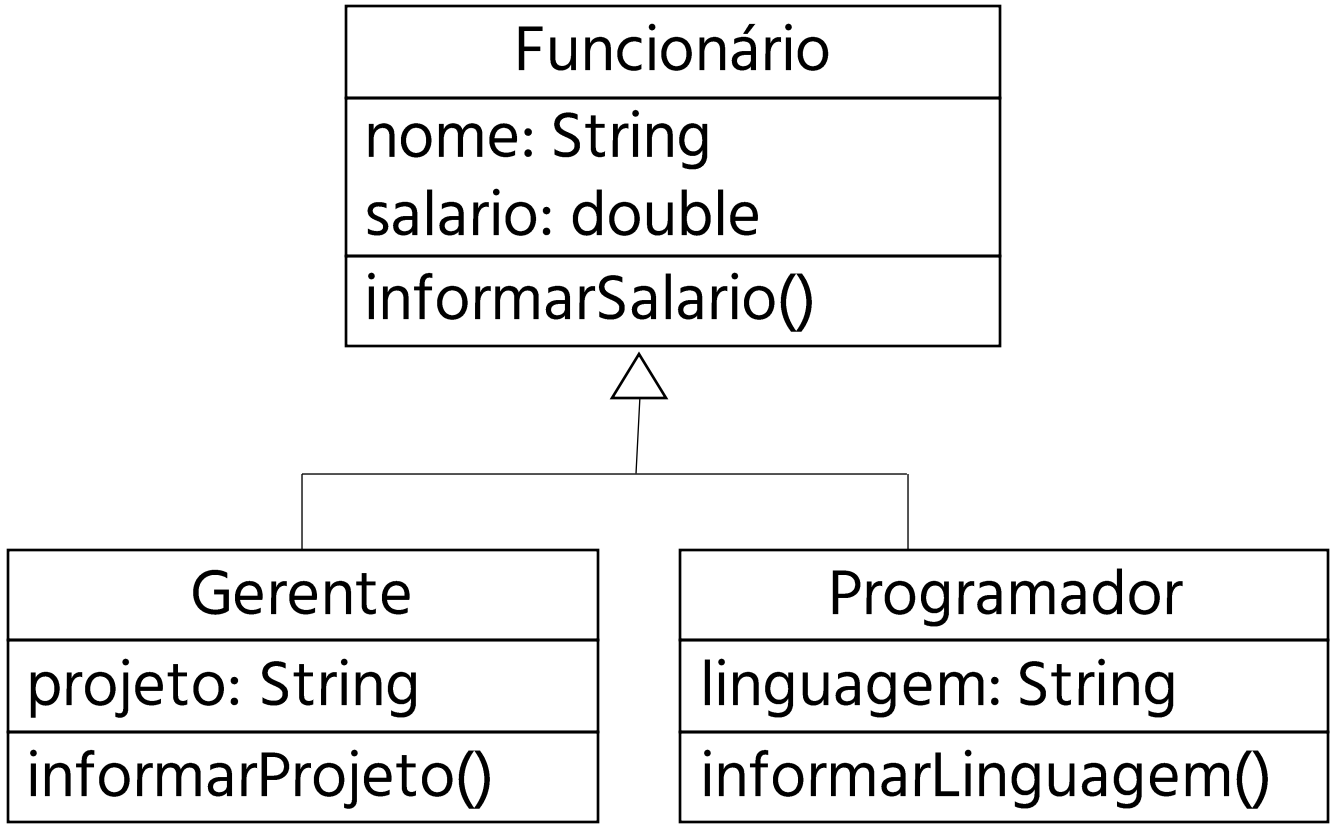
Disponível na sala virtual.

Exercício

Exercício – parte 1

Encapsule as classes!
Crie os métodos
get e set necessários.

Crie as seguintes classes:



Exercício - parte 2

Crie uma quarta classe Java chamada Principal, que...

a) Terá o método main implementado.
b) Irá instanciar as classes Scanner (para receber as entradas), Gerente e Programador.
c) **Receberá o nome e salário de cada um dos 2 funcionários (programador e gerente).**
d) Receberá o nome do projeto do gerente e o nome da linguagem utilizada pelo programador.
e) Irá enviar os dados para cada objeto das classes Gerente e Programador
f) Irá exibir todos os dados do gerente e do programador.
g) Experimente criar um objeto da classe Funcionário e armazenar um objeto da classe Programador ou Gerente nele. É possível?

Observação

Atente que na letra **c** do exercício, você estará com objetos das subclasses (programador e gerente) e irá utilizar o método da superclasse para *setar* nome e salário.

Classe Principal

*Principal.java | Funcionário.java | Gerente.java | Programador.java

```java
import java.util.Scanner;

public class Principal {

    public static void main(String[] args) {

        Gerente g = new Gerente();
        Programador p = new Programador();
        Scanner teclado = new Scanner(System.in);
        System.out.println("Qual o projeto do gerente?");
        g.setProjeto(teclado.nextLine());
        System.out.println("Qual a linguagem do programador?");
        p.setLinguagem(teclado.nextLine());

        g.informarProjeto();
        p.informarLinguagem();

        Funcionário f1 = new Gerente();
        Funcionário f2 = new Programador();

    }

}
```

Problems | Javadoc | Declaration | Console

```
<terminated> Principal (11) [Java Application] C:\Program Files
Qual o projeto do gerente?
SOS Drones
Qual a linguagem do programador?
Java
O projeto é SOS Drones
A linguagem é Java
```

Classe Funcionário

*Principal.java | *Funcionário.java | Gerente.java | Programador.j

```
1
2  public class Funcionário {
3      private String nome;
4      private double salário;
5
6      public String getNome() {
7          return nome;
8      }
9      public void setNome(String nome) {
10         this.nome = nome;
11     }
12     public double getSalário() {
13         return salário;
14     }
15     public void setSalário(double salário) {
16         this.salário = salário;
17     }
18
19     public void informarSalário() {
20         System.out.println("O salário é $"+getSalário());
21     }
22 }
23
```

Classe Gerente

*Principal.java | *Funcionário.java | *Gerente.java | Progr

```
1
2  public class Gerente extends Funcionário {
3      private String projeto;
4
5      public String getProjeto() {
6          return projeto;
7      }
8
9      public void setProjeto(String projeto) {
10         this.projeto = projeto;
11     }
12
13     public void informarProjeto() {
14         System.out.println("O projeto é "+getProjeto());
15     }
16 }
```

Classe Programador

*Principal.java | *Funcionário.java | *Gerente.java | *Programador.java

```
1
2  public class Programador extends Funcionário {
3      private String linguagem;
4
5
6      public String getLinguagem() {
7          return linguagem;
8      }
9
10
11     public void setLinguagem(String linguagem) {
12         this.linguagem = linguagem;
13     }
14
15
16     public void informarLinguagem() {
17         System.out.println("A linguagem é "+getLinguagem());
18     }
19
20 }
21
```

Exercício 1

Com relação à orientação a objetos em JAVA, é INCORRETO afirmar que JAVA permite:

a) herança múltipla

b) polimorfismo

c) sobrecarga

d) métodos de classe

e) a implementação de múltiplas interfaces

Exercício 1 - Resposta

A

Exercício 2

Na linguagem de programação Java, para indicar que uma classe X é derivada de Y, utiliza-se, na declaração da classe X um modificador. Assinale a alternativa que contém esse modificador.

a) imports
b) extends
c) inherits
d) subclass
e) superclass

Exercício 2 - Resposta

B

Exercício 3
O que vai aparecer no console?

```
public class A  {
        int n = 3;
        public int soma(int i) {
                return (this.n+i);
        }
}
```

```
public class B extends A{

        public int soma(int i) {
                return (i+4);
        }
}
```

```
public class Principal {
        public static void main(String[] args) {
                A a = new A();
                System.out.print(a.soma(3));
                B b = new B();
                System.out.print(" e "+ b.soma(8));
        }
}
```

Exercício 3 - Resposta

6 e 12

Exercício 4

Com relação a herança em programação orientada a objetos com Java, analise as alternativas a seguir e marque a opção INCORRETA.

a) Uma subclasse herda os métodos da superclasse, entretanto, pode ter seus próprios métodos.

b) Quando se instancia um objeto da subclasse, podem ser passados valores para os atributos da superclasse.

c) Um objeto da subclasse pode ser um objeto da superclasse.

d) Em uma superclasse, para acessar métodos da subclasse deve ser usada a instrução super.

e) Para definir que a subclasse herda as características da superclasse utiliza-se a instrução extends na declaração da subclasse.

Exercício 4 – Resposta

D

Exercício 5

Palavras Cruzadas!

https://puzzel.org/pt/crossword/play?p=-ND2lJmiP1feJRfzo5Dc

Intervalo: 15 minutos

Relembrando
Herança Simples x Múltipla

- Herança simples herda apenas de uma única classe pai (ou superclasse).
- Herança múltipla, são herdadas de várias classes.
- **Java só permite herança simples entre classes.**

- Mas se ligue! É possível implementar uma espécie de herança múltipla utilizando as interfaces (na próxima aula veremos como isso é possível).

Modificador de acesso final na Herança

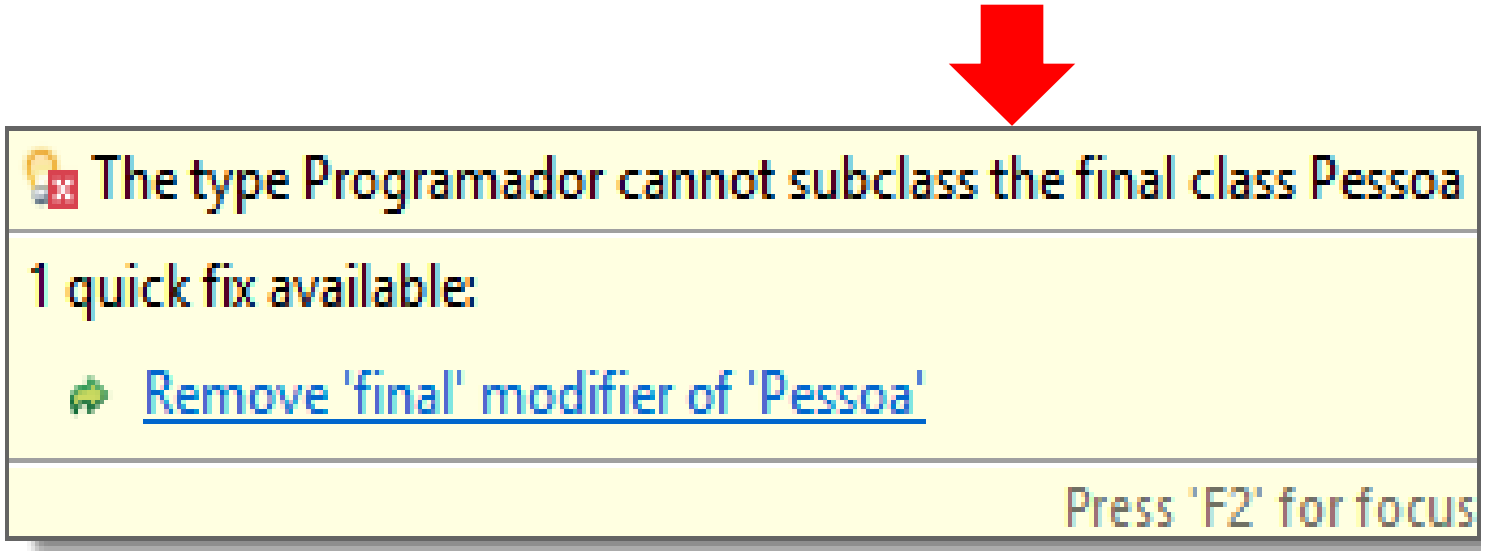
- É possível restringir uma classe para que **não seja possível criar classes filhas** a partir dela.
- Uma classe com este modificador não pode ser estendida, isto é, não podem ser implementadas classes que sejam herdeiras dela.
- É como se a classe fosse “esterilizada”!
- Diz-se que é impossível “estendê-la”!

Modificador de acesso final na Herança

No exemplo abaixo, torna-se impossível criar uma classe filha da classe Pessoa.

public final class Pessoa {...}

public class Programador extends Pessoa {... }



4. Polimorfismo

O que é Polimorfismo?

QUANDO VÁRIOS POWER RANGERS
MORFAM AO MESMO TEMPO

CHAMAMOS ISSO DE
POLIMORFISMO

Polimorfismo

- O termo polimorfismo é originário do grego e significa "muitas formas" (**poli** = muitas, **morphos** = formas).

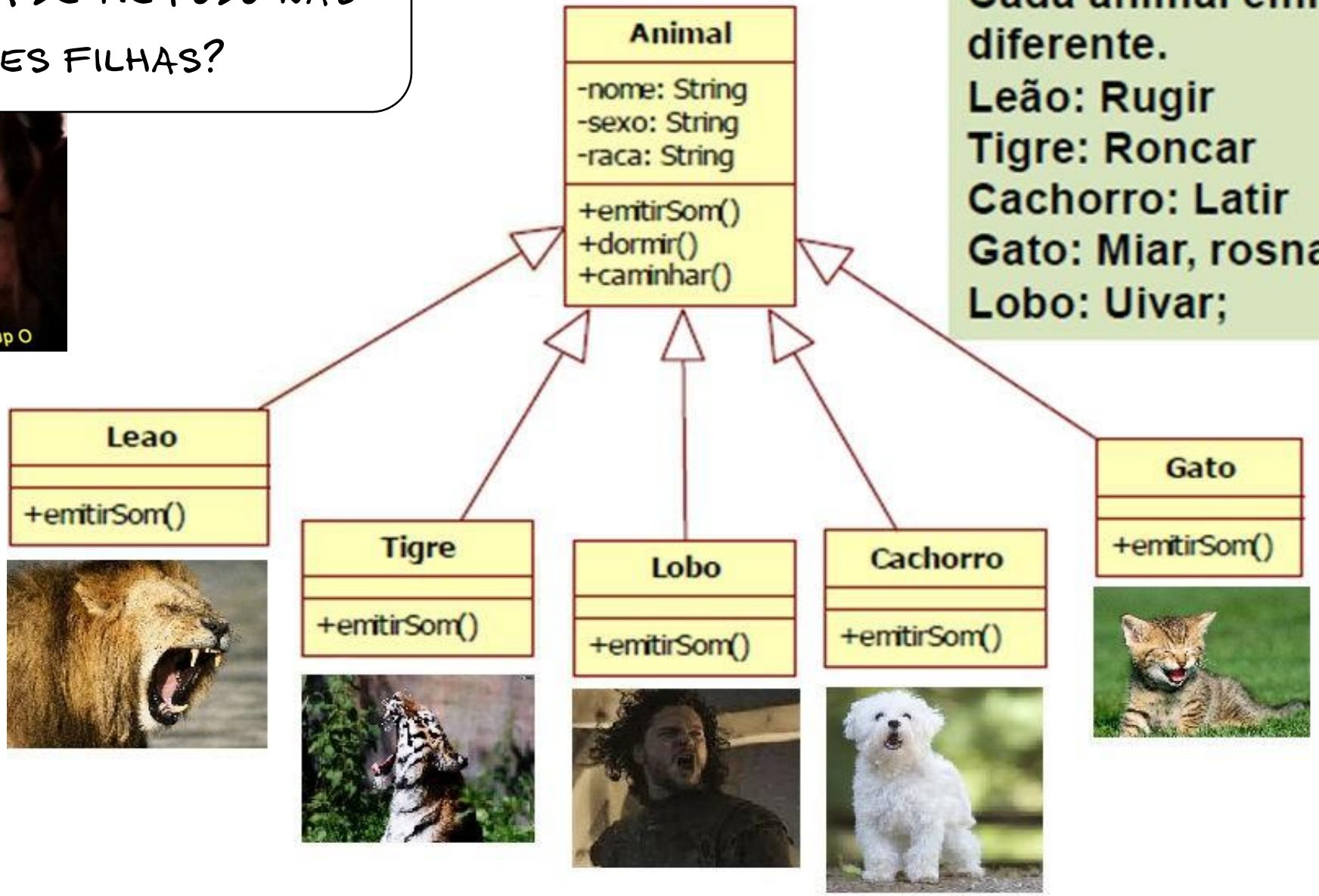
- Numa linguagem de programação, isso significa que pode haver várias formas de fazer uma “certa coisa”.
- Mas... O que é uma “certa coisa”?

Polimorfismo

- A “certa coisa” é uma chamada aos métodos.
- Portanto, em Java, o polimorfismo se manifesta **apenas em chamadas de métodos**.
- Polimorfismo significa que **uma chamada de método pode ser executada de várias formas** (ou polimorficamente).
- Quem decide “a forma” é o objeto que recebe a chamada!

PARECE QUE EXISTE UMA **SOBRESCRITA** DE MÉTODO NAS CLASSES FILHAS?

Cada animal emite sons diferente.
Leão: Rugir
Tigre: Roncar
Cachorro: Latir
Gato: Miar, rosnar
Lobo: Uivar;



Polimorfismo

- Na programação orientada a objetos, o polimorfismo permite que referências de tipos de classes mais abstratas (superclasses) representem o comportamento das classes concretas que referenciam (upcasting).
- Por exemplo: `Animal a1 = new Leao();  a1.emitirSom();`
- Assim, é possível tratar vários tipos de maneira homogênea (através da interface do tipo mais abstrato, a classe Animal).
- A superclasse (Animal) pode definir um método que é **sobrescrito** por suas subclasses (**sobrescrita** de método).
- **Em herança temos um relacionamento de sobrescrita de método.**

Consequências do polimorfismo

Sobrecarga e Sobrescrita

Sobrecarga e Sobrescrita

- A sobrescrita e a sobrecarga de métodos acompanham o Java desde o seu início.
- É bastante comum, principalmente para os iniciantes na linguagem, estranharem a existência de **métodos com mesmo nome** declarados em **uma mesma classe**, situação que certamente geraria erro em algumas linguagens de programação.

Sobrescrita ou Sobreposição

ou também Polimorfismo por Inclusão ou Polimorfismo de Herança

@Override

A Sobrescrita ou Sobreposição

- A sobrescrita (ou @Override) está diretamente relacionada à orientação a objetos, mais especificamente com a herança.
- Com a sobrescrita, conseguimos **especializar** os **métodos herdados das superclasses**, alterando o seu comportamento nas subclasses por um mais específico.

A Sobrescrita

- O conceito de sobrescrita, reside quando você declara um **método na classe filha** com a **mesma assinatura da classe pai**.
- Você está sobrescrevendo e substituindo o método da classe pai pelo mesmo método, no entanto, agora existe a possibilidade de uma implementação diferente da existente na classe pai.

Assinatura de Método

A assinatura de um método é composta apenas pela primeira linha do método. Ao invés de ter as chaves contendo o corpo do método, substituímos elas por um ponto e vírgula.

```
public double calcularArea (double n1, double n2) {
        return (n1 * n2);
}
```

Método Completo

```
public double calcularArea(double n1, double n2);
```

Assinatura do Método

Exemplo Sobrescrita

O que vai aparecer no console?

```
public class A {
        public void fazAlgo() {
        System.out.println("Método fazAlgo
          da classe A (Classe Pai)");
        }
}
public class B extends A {
        @Override
        public void fazAlgo() {
        System.out.println ("Método fazAlgo
        da classe B (Classe Filha)");
        }
}
```

```
public class Principal {
   public static void main(String args [] ) {
   A a = new A();
   a.fazAlgo();
   B b = new B();
   b.fazAlgo();
   b.super.fazAlgo();
   }
}
```

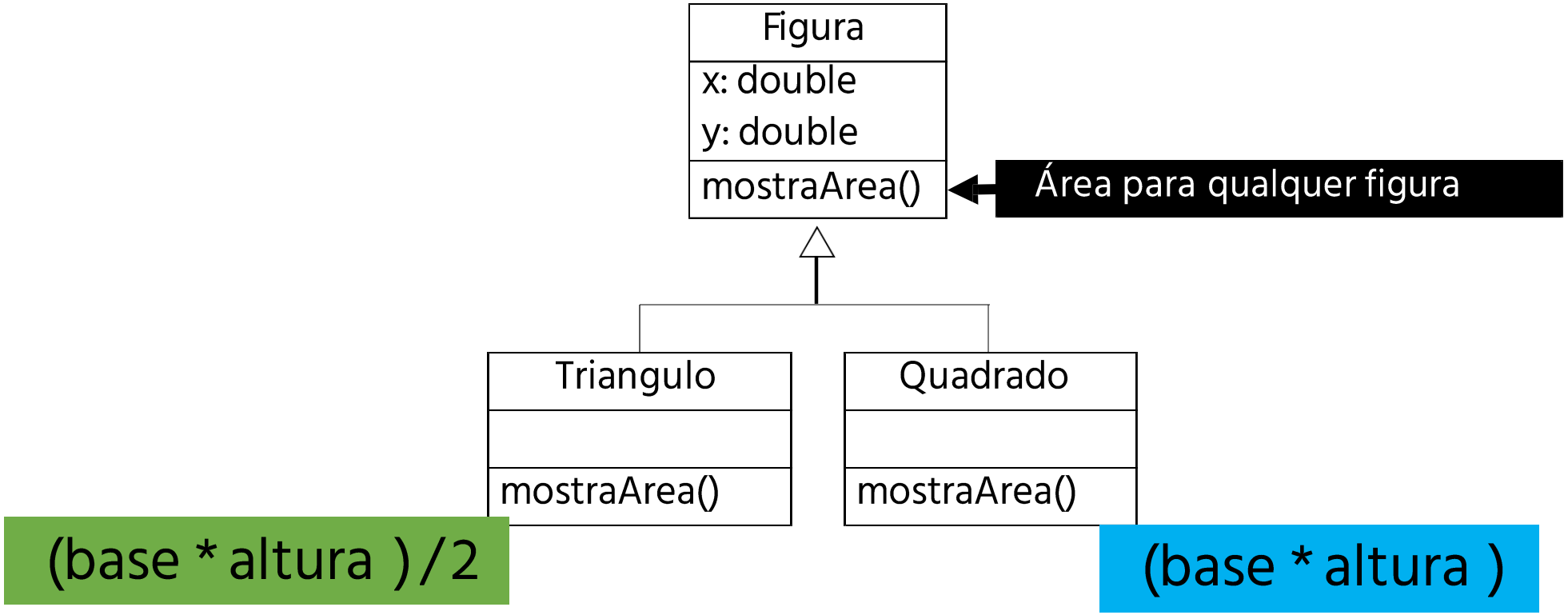
Retorno:
"Método fazAlgo da classe A (Classe Pai)"

Retorno:
"Método fazAlgo da classe B (Classe Filha)";

Retorno:
"Método fazAlgo da classe A (Classe Pai)"

Outro exemplo de Sobrescrita

Outro Exemplo de Sobrescrita



Modificador super

- Assim como o modificador **this** acessa as variáveis de instância da classe atual, o modificador **super** acessa as variáveis de instância ou métodos da superclasse.
- É possível no corpo de um método da classe filha chamar um método da classe pai: **super.MétodoDaSuperclasse();**

- Lembre-se que também é possível utilizar **super();** dentro do construtor da classe filha para chamar o construtor da classe pai.

Sobrecarga

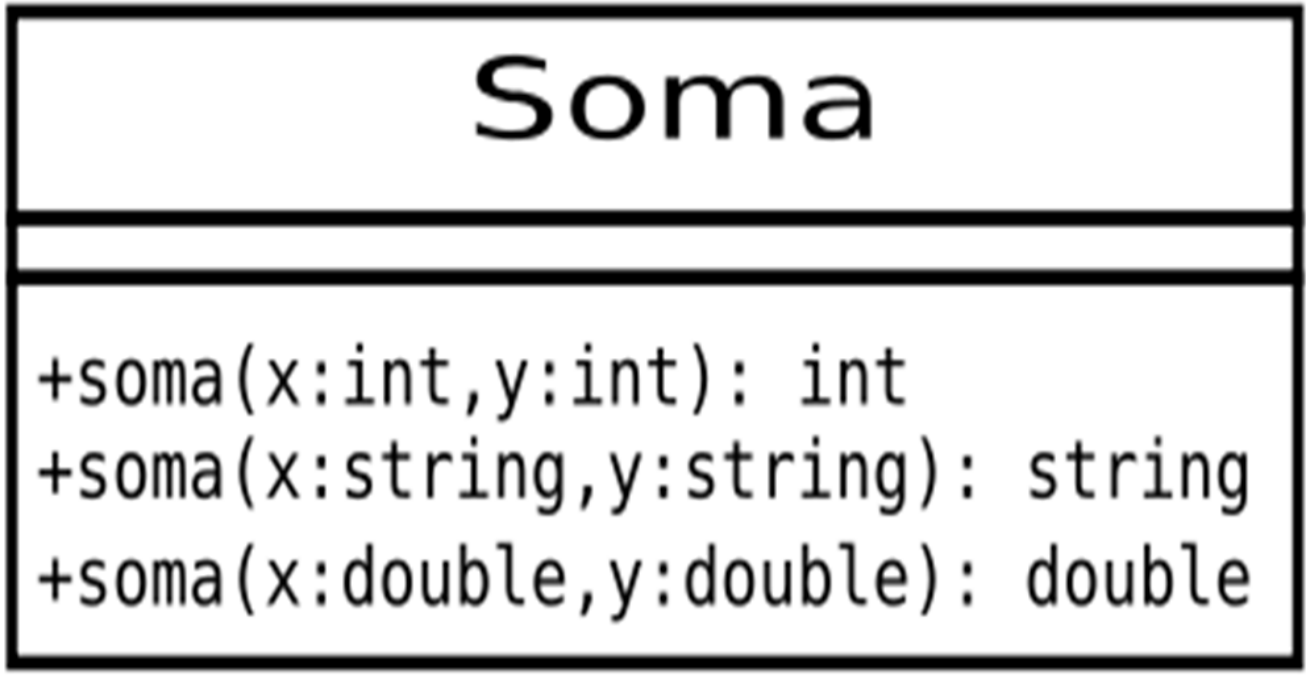
@Overload

A Sobrecarga

- A sobrecarga de método permite a existência de vários métodos com nomes iguais **na mesma classe**.
- No entanto, as assinaturas são levemente diferentes, ou seja, a assinatura pode variar na:

1. Quantidade de parâmetros
2. Tipo de parâmetros
3. Tipo e valor de retorno
4. Modificadores de acesso

A Sobrecarga



public int soma (int x, int y) { … }

public String soma (String x, String y) { … }

public double soma (double x, double y) { … }

Quando ocorre sobrescrita?
Quando ocorre sobrecarga?

Com a sobrescrita, conseguimos especializar os métodos herdados das superclasses, alterando o seu comportamento nas subclasses por um mais específico.

Exercício

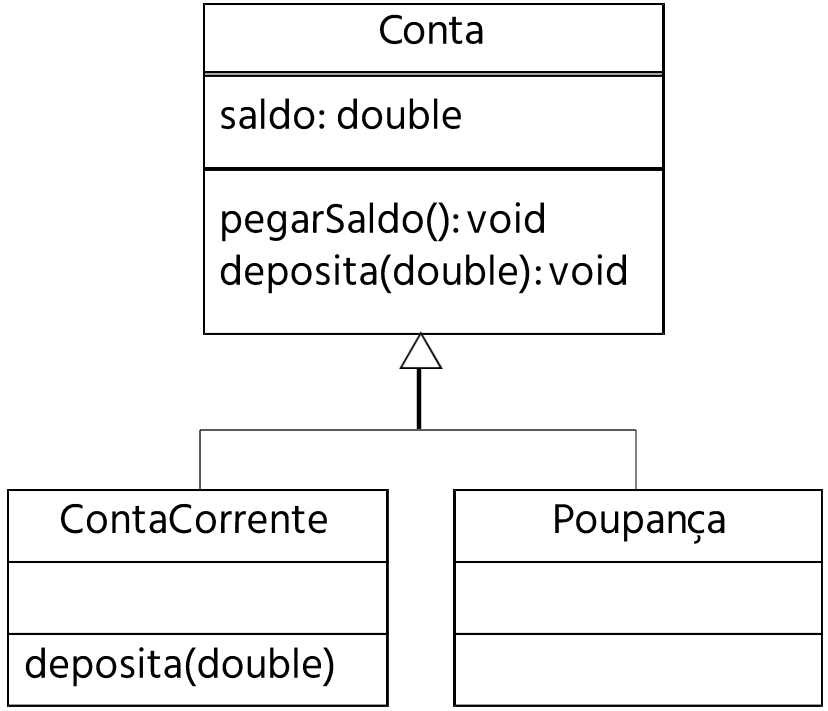
Exercício 1

Implemente as classes abaixo:

Todos os atributos são privados. Crie os métodos get e set necessários.

OBS: O método deposita da Subclasse ContaCorrente Deverá descontar uma taxa de 10 centavos em cada depósito.

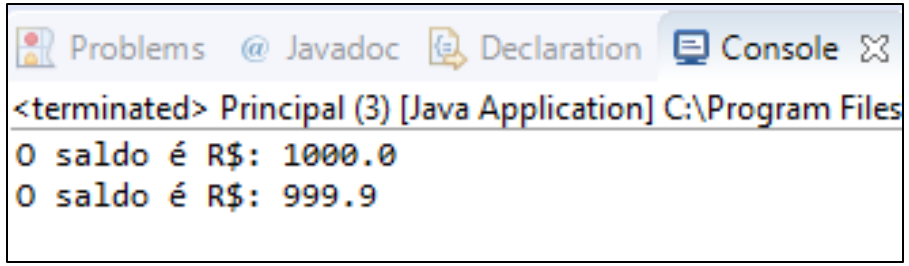
Para depositar na conta poupança será utilizado o método da superclasse.



Exercício 1

Crie uma classe Java chamada Principal, que...

1. Terá o método main implementado.
2. Crie dois objetos, um de ContaCorrente e outro de Poupança.
3. Deposite 1000,00 em cada objeto.
4. Exibir o saldo da conta corrente e o saldo da conta poupança.



Exercício 1
Resposta

*Conta.java | Poupança.java | ContaCorrente.java | *Princi

```
1
2  public class Conta {
3      private double saldo = 0;
4
5      public double getSaldo() {
6          return saldo;
7      }
8
9      public void setSaldo(double saldo) {
10         this.saldo = saldo;
11     }
12
13     public void pegarSaldo() {
14         System.out.println( "O saldo é R$: "+this.saldo);
15     }
16
17     public void deposita(double valor) {
18         this.saldo += valor;
19     }
20 }
```

Problems | Javadoc | Declaration | Console

```
<terminated> Principal (3) [Java Application] C:\Program Files
O saldo é R$: 1000.0
O saldo é R$: 999.9
```

*Conta.java | Poupança.java | ContaCo

```
1
2  public class Poupança extends Conta {
3
4  }
5
```

*Conta.java | Poupança.java | *ContaCorrente.java

```
1
2  public class ContaCorrente extends Conta{
3
4      public void deposita(double valor) {
5          double saldo = super.getSaldo();
6          double novosaldo = saldo + valor - 0.10;
7          super.setSaldo(novosaldo);
8
9      }
10
11 }
```

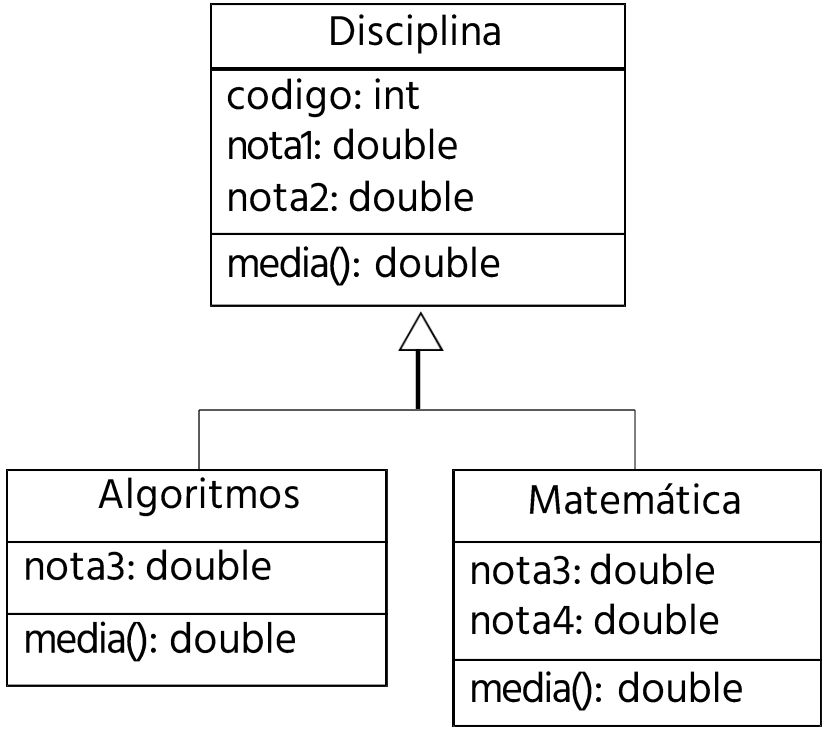
*Conta.java | Poupança.java | *ContaCorrente.java | *Principal.java

```
1
2  public class Principal {
3
4      public static void main(String[] args) {
5          Poupança p1 = new Poupança();
6          p1.deposita(1000);
7          p1.pegarSaldo();
8
9          ContaCorrente c1 = new ContaCorrente();
10         c1.deposita(1000);
11         c1.pegarSaldo();
12     }
13 }
14
```

Exercício 2

Implemente as classes abaixo:

OBS: Todos os atributos são privados. Crie os métodos get e set necessários.



Exercício 2

Crie uma classe Java chamada Principal, que...

1. Terá o método main implementado.
2. Irá instanciar as classes Scanner (para receber as entradas) e objetos da classe Algoritmos e Matemática.
3. Receberá um código (por exemplo, 001) para o aluno e as notas para cada disciplina.
4. Irá enviar os dados para cada objeto da classe Disciplina.
5. Irá calcular e exibir a média do aluno para cada disciplina.

Exercício 2 - Resposta

```java
public class Principal {

    public static void main(String[] args) {
        Algoritmos alg = new Algoritmos();
        alg.setCódigo(001);
        alg.setNota1(8);
        alg.setNota2(7);
        alg.setNota3(6);
        System.out.println(alg.media());

        Matemática mat = new Matemática();
        mat.setCódigo(002);
        mat.setNota1(5);
        mat.setNota2(6);
        mat.setNota3(7);
        mat.setNota4(8);
        System.out.println(mat.media());
    }
}
```

```
Console
<terminated> Pri
7.0
6.5
```

```java
public class Disciplina {
    private int código;
    private double nota1;
    private double nota2;

    public double media() {
        return ((this.nota1+this.nota2)/2);
    }

    public int getCódigo() {
        return código;
    }
    public void setCódigo(int código) {
        this.código = código;
    }
    public double getNota1() {
        return nota1;
    }
    public void setNota1(double nota1) {
        this.nota1 = nota1;
    }
    public double getNota2() {
        return nota2;
    }
    public void setNota2(double nota2) {
        this.nota2 = nota2;
    }
}
```

```java
public class Algoritmos extends Disciplina {
    private double nota3;

    public double media() {
        double media = ((super.getNota1()
                +super.getNota2()+this.nota3)/3);
        return media;
    }

    public double getNota3() {
        return nota3;
    }

    public void setNota3(double nota3) {
        this.nota3 = nota3;
    }
}
```

```java
public class Matemática extends Disciplina {
    private double nota3;
    private double nota4;

    public double media() {
        return ((super.getNota1()+super.getNota2()+
                this.nota3 + this.nota4)/4);
    }

    public double getNota3() {
        return nota3;
    }
    public void setNota3(double nota3) {
        this.nota3 = nota3;
    }
    public double getNota4() {
        return nota4;
    }
    public void setNota4(double nota4) {
        this.nota4 = nota4;
    }
}
```

Lista de Exercícios

Próxima aula

Sobrecarga, Sobrescrita e Classes Abstratas:
https://youtu.be/30GVvg16w8Q

Interfaces:

https://youtu.be/GUuUVMl6vzU

Referências

CAELUM. Java e Orientação a Objetos. Acesso em 11 fev 14. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/>.

SIERRA, K. BATES, B. **Use a Cabeça! Java**. 2. ed. Rio de Janeiro: Alta Books, 2010. p. 127-146.

PROGRAMAÇÃO ORIENTADA A OBJETOS

AULA 11 – PILARES DA PROGRAMAÇÃO ORIENTADA A OBJETOS

WALTER TRAVASSOS SARINHO

@PROF.WALTERTRAVASSOS

WALTER.TRAVASSOS@SECTRAS.EDU.BR