PROGRAMAÇÃO ORIENTADA A OBJETOS

AULA 9 – O PARADIGMA ORIENTADO A OBJETOS

WALTER TRAVASSOS SARINHO

@PROF.WALTERTRAVASSOS

WALTER.TRAVASSOS@SECTRAS.EDU.BR

Vamos começar falando sobre ArrayLists de objetos das classes que nós criamos

- Em nossos exemplos, utilizamos até agora uma lista de objetos da classe **String**.
- Você pode compor o ArrayList com objetos das classes que você criou no seu projeto, como por exemplo, a classe Produto:

Por exemplo:

- ArrayList<Produto> produtos = new ArrayList<Produto>();

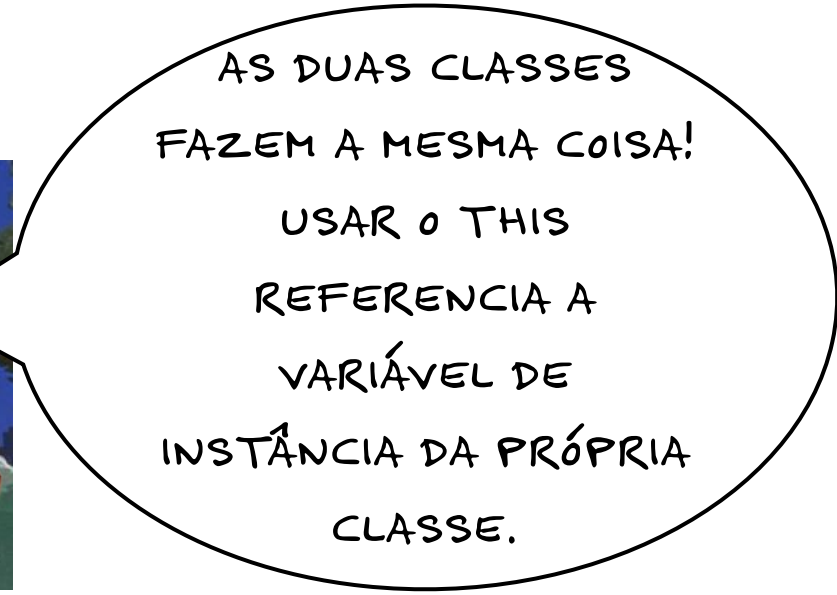
*Vamos criar a classe Produto **(variáveis: nome e preço).**

O que é a auto referência this?

A que (ou quem) essa referência se refere?

A auto referência this

A palavra reservada **this** permite que o método acesse as variáveis de instância (e métodos) da própria classe.




```
public class Produto {
    public String nome = "";
    public double preço = 0;

    public Produto (String a, double b) {
        nome = a;
        preço = b;
    }
}
```

```
public class Produto {
    public String nome = "";
    public double preço = 0;

    public Produto (String nome, double preço) {
        this.nome = nome;
        this.preço = preço;
    }
}
```

Método toString()

Método toString()

- Atualmente se você exibir um objeto (que não seja da classe String). Será mostrado um código estranho (referência ao endereço na memória).
- Por exemplo:

```
Produto p1 = new Produto("Vassoura", 20);
System.out.println(p1);
```

- Será exibido no console algo como:

```
Produto@15db9742
```

Método toString()

- Em geral, para exibir o valor de um objeto, acessamos suas variáveis de instância por meio dos métodos **get**.

```
Produto p1 = new Produto("Vassoura", 20 );
System.out.println( p1.getNome() );
System.out.println( p1.getPreço() );
```

- No console será exibido:

Vassoura

20

Método toString()

- Não seria mais fácil se o comando **System.out.println(p1);** retornasse os valores das variáveis de instância?
- Isso é possível fazendo a **sobrescrita** do método **toString();**
- O método já existe, o que iremos fazer é uma sobrescrita!

```
@Override
public String toString() {
        return "Produto[nome=" + nome + ", preço=" + preço + "]";
}
```

Atalho para sobrescrever o método toString():
alt + shift + s + s

Em seguida pressione ok!

Lembre-se que esse método deve ser escrito dentro da classe Produto (no caso de nosso exemplo)

Classe Object e Sobrescrita (Override)

- Você deve ter visto a anotação **@Override** quando criou o método **toString()** por atalho.
- Mas o que significa essa *annotation*?

Classe Object e Sobrescrita (Override)

- Toda classe em Java herda métodos de uma classe base chamada **Object**. Ela é a classe pai de todas as classes.
- Se você criar uma classe, ela herda automaticamente os métodos de Object.
- Esses métodos são marcadores e fazem algo diferente do que você deseja em seu projeto.
- A importância desses métodos existe quando você os sobrepõe (faz um *Override*).
- Quando você sobrepõe um método herdado, a implementação anterior do método é substituída pela atual.
- Vamos falar mais sobre isso na aula 11!

Exercícios

Exercício

Classe Pessoa

1. Crie a classe Pessoa. Adicione os atributos privados String nome e int idade.
2. Crie o construtor recebendo os dois parâmetros (**alt + shift + s + o**).
3. Crie os gets e sets (**alt + shift + s + r**)
4. Sobrescreva o método toString() (**alt + shift + s + s**);

Classe Principal

1. Crie a classe Principal. Adicione o método main. Dentro do método main:
2. Crie um ArrayList da classe Pessoa chamado pessoas.
3. Crie 3 objetos da classe Pessoa.
4. Adicione ao ArrayList os 3 objetos criados.
5. Exiba o conteúdo do ArrayList.

Resposta

*Principal.java | Pessoa.java

```java
import java.util.ArrayList;

public class Principal {

    public static void main(String[] args) {

        ArrayList<Pessoa> pessoas = new ArrayList<Pessoa>();
        Pessoa p1 = new Pessoa("Yuri", 25);
        pessoas.add(p1);

        pessoas.add(new Pessoa("Ricardo", 37));
        pessoas.add(new Pessoa("Walter", 35));

        for (Pessoa p: pessoas) {
            System.out.println(p);
        }
    }

}
```

Console | Problems | @ Javado

```
<terminated> Principal (5) [Java Application
Pessoa [nome=Yuri, idade=25]
Pessoa [nome=Ricardo, idade=37]
Pessoa [nome=Walter, idade=35]
```

Principal.java | *Pessoa.java

```java

public class Pessoa {
    private String nome;
    private int idade;
    public Pessoa(String nome, int idade) {
        super();
        this.nome = nome;
        this.idade = idade;
    }

    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    @Override
    public String toString() {
        return "Pessoa [nome=" + nome + ", idade=" + idade + "]";
    }
}
```

Dúvidas sobre a aula passada

Aula de Hoje

- Apresentar o paradigma de programação Orientado a Objetos (OO).
- Conhecer o significado das expressões: **classes**, **objetos**, **métodos** e **modificadores de acesso**.
- Aprender a modelar e implementar um diagrama de classes (UML 2.0).
- Aprender a utilizar os métodos get e set.

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

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

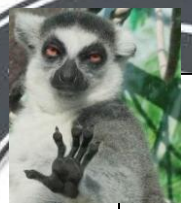
AULA 09
O paradigma Orientado a Objetos

AULA 10
Construindo métodos e Métodos construtores

Avaliação 2

Java

Mapa de Estudos



AULA 12
Classes Abstratas e Interfaces

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

Avaliação 3

Java

Paradigma de Programação Orientada a Objetos

Uma viagem até objetópolis

Leitura Complementar - Capítulo 2 – Classes e Objetos p. 21-36

SIERRA, K.; BATES, B. Use a Cabeça – Java. Rio de Janeiro: Alta Books. 2 ed. 2007.

Precisamos conhecer algumas poucas regras antes de ir até objetópolis...

Margaret Hamilton
Diretora de engenharia de software

1. As coisas que um objeto conhece sobre si mesmo se chamam variáveis de instância.

Elas representam o estado de um objeto (os dados) e podem ter valores exclusivos para cada objeto desse tipo.

Use a Cabeça! Java – p. 27

2. Considere instância como outra maneira de dizer objeto.

Use a Cabeça! Java – p. 27

3. As coisas que um objeto pode fazer se chamam métodos.

Use a Cabeça! Java – p. 27

E mais: métodos são funções evocadas por uma classe. No fim das contas, procedimentos, funções e métodos são blocos nomeados de código.

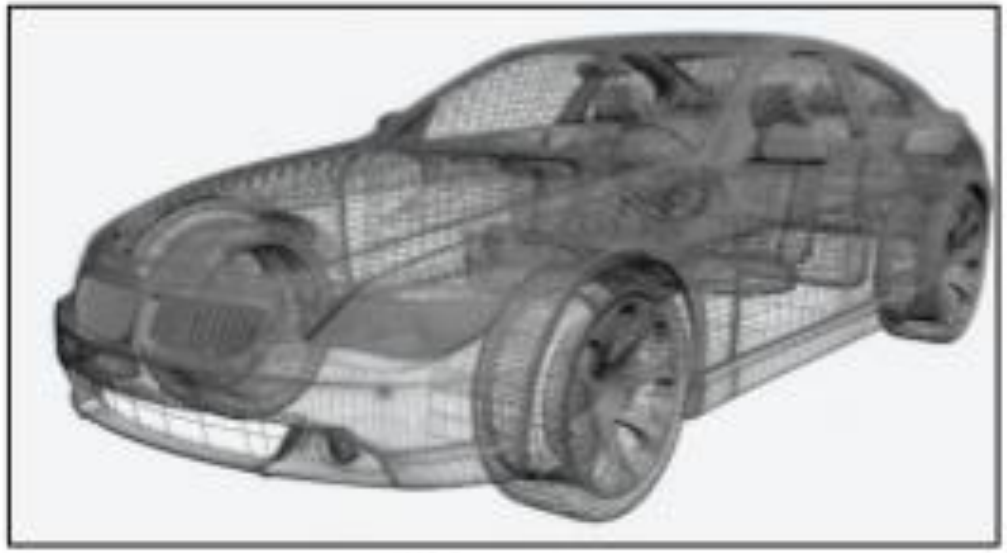



4. Uma classe não é um objeto! (mas é usada para construí-los).

A classe é o projeto de um objeto. Ela informa à maquina virtual como criar um objeto desse tipo específico. Cada objeto criado a partir dessa classe terá seus próprios valores para as variáveis de instância da classe. Por exemplo, você pode usar a classe Button para criar vários botões diferentes, e cada botão poderá ter sua própria cor, tamanho, forma, rótulo e assim por diante.

Use a Cabeça! Java – p. 27

A Relação entre Classe e Objeto



Classe

Objeto

Fonte: http://blog.stancke.com.br/introducao-a-orientacao-a-objetos/

A relação entre Classe e Objeto

EXEMPLO DE OBJETO

Um carro é um objeto que possui características (cor, altura, largura, comprimento, potência, etc...) e executa ações (mover, abrir porta, acelerar, parar, buzinar, etc...).

As características e as ações que ele executa podem ser modeladas em um software que trata o carro como um objeto. Todo carro pode ser modelado a partir do grupo de objetos similares, ou seja, a classe carro.

Objeto Carro

Fonte: http://www.usandoaccess.com.br/tutoriais/classe-no-access-orientacao-a-objetos.asp

Primeiro vamos conhecer como representar graficamente uma classe!

Utilizando o Diagrama de Classes da UML 2.0

O Diagrama de Classes

- É um dos diagramas mais utilizados na UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada).
- Define as estruturas das classes utilizadas pelo sistema.
- Apresenta atributos (**variáveis de instância**) e **métodos** de cada classe.
- Também mostra **como as classes se relacionam** e trocam informações entre si.

Uma observação

- O paradigma é orientado a objeto. Portanto **o diagrama de classe é uma meta representação de um dado domínio** ou problema a ser resolvido.
- O diagrama de classes **pode ser implementado em qualquer linguagem** que ofereça suporte ao paradigma orientado a objetos.
- Nesta disciplina, utilizamos o java como linguagem para implementar o paradigma orientado a objetos, no entanto, outras linguagens também oferecem suporte a este paradigma!

Exemplo:
Diagrama da Classe Pessoa

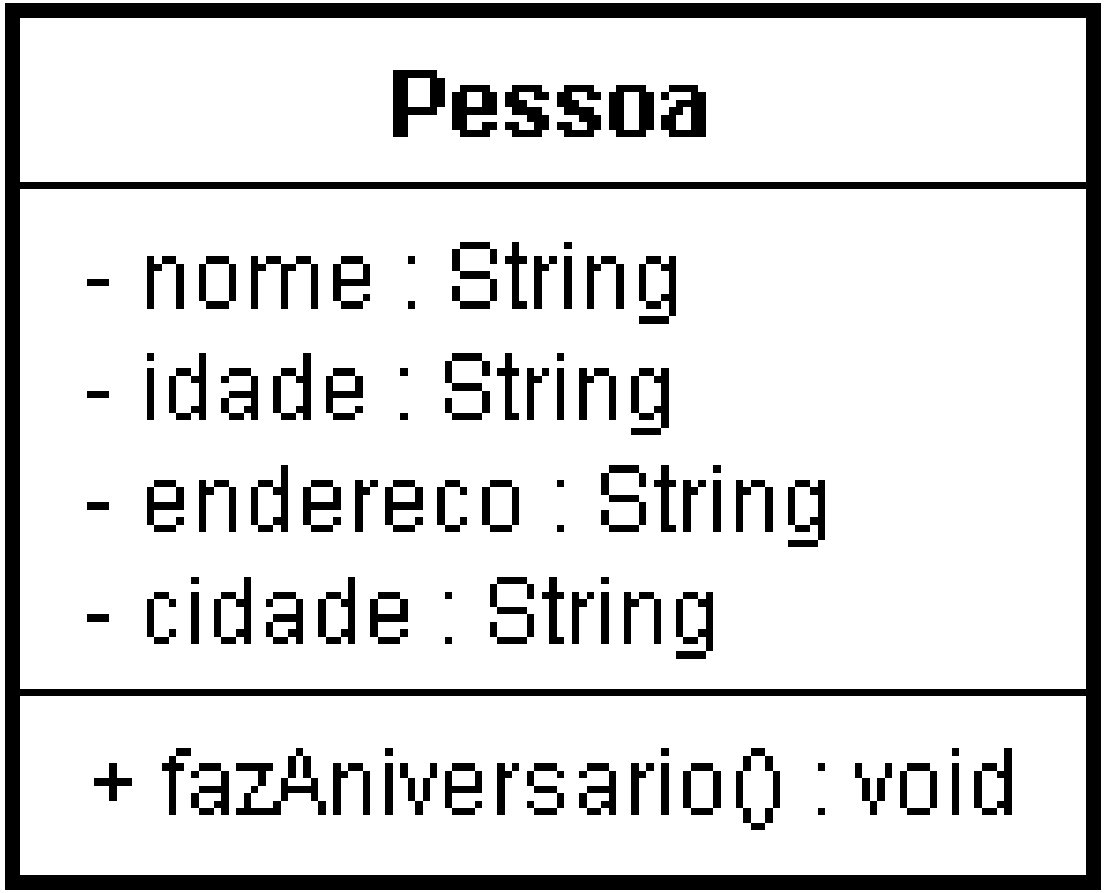


Diagrama da Classe Pessoa

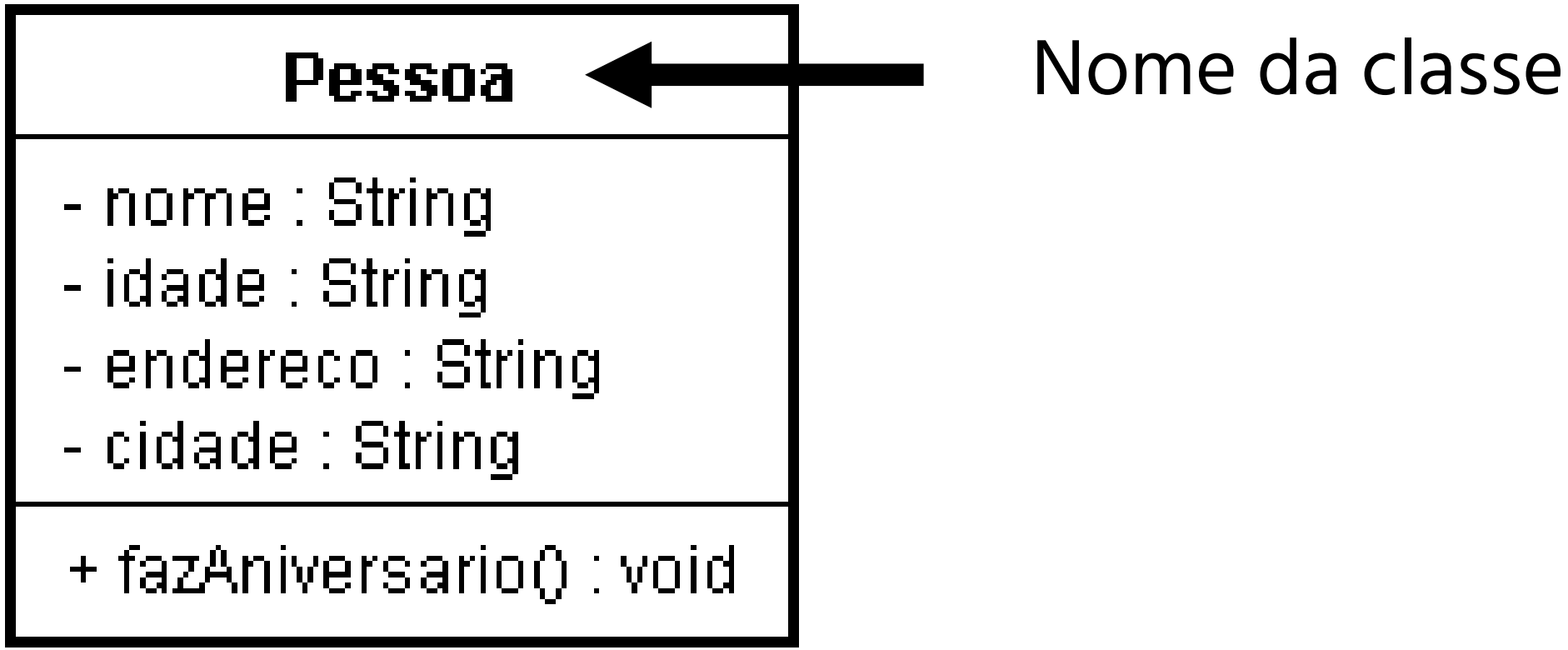


Diagrama da Classe Pessoa

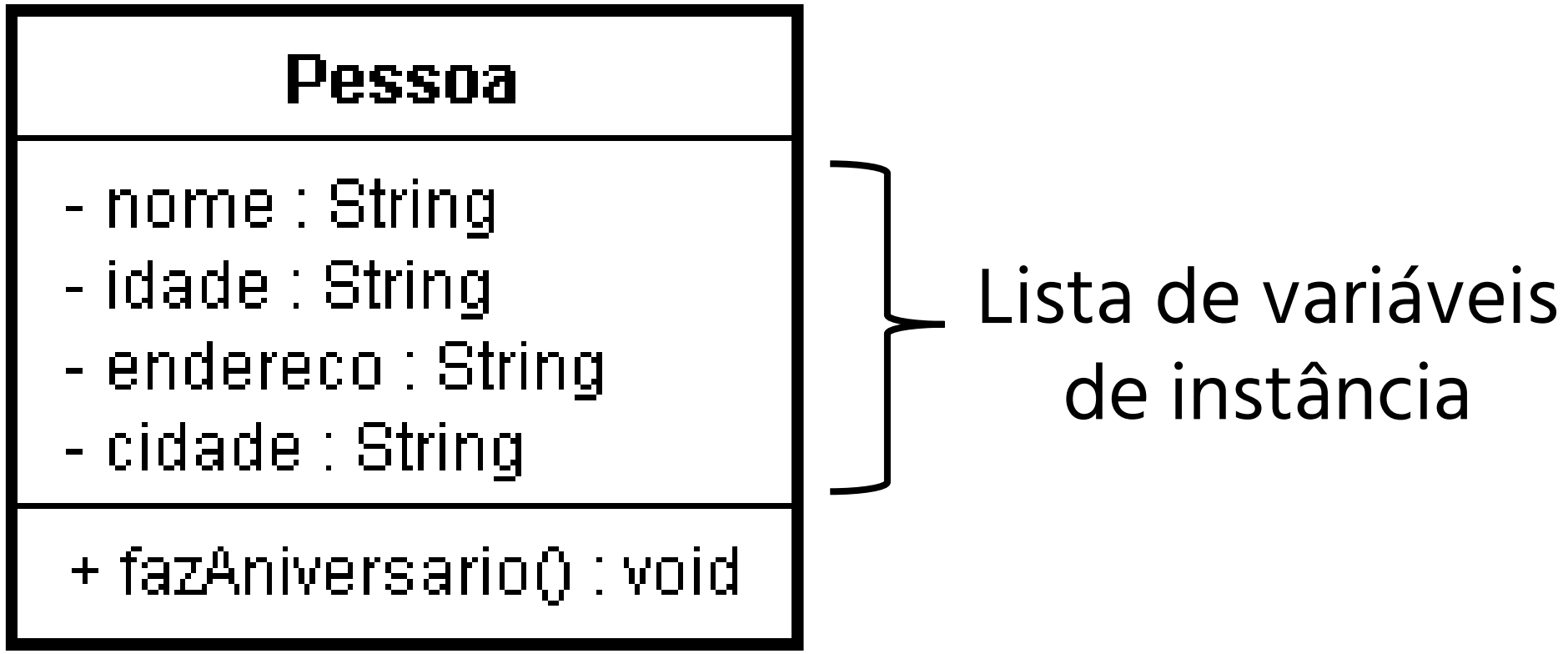
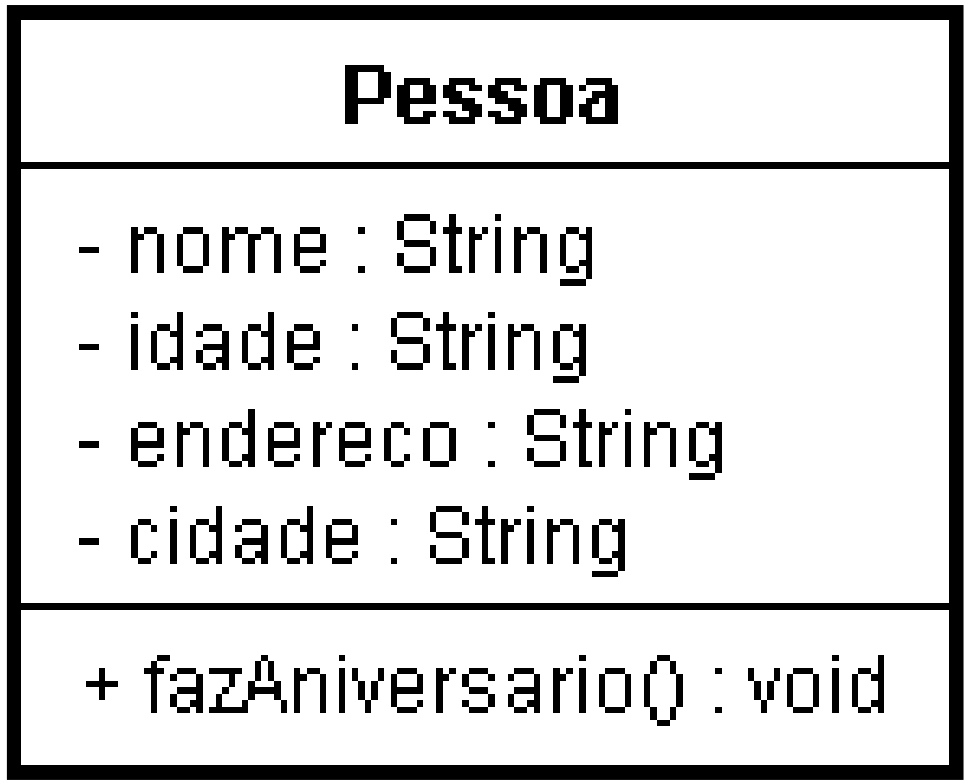
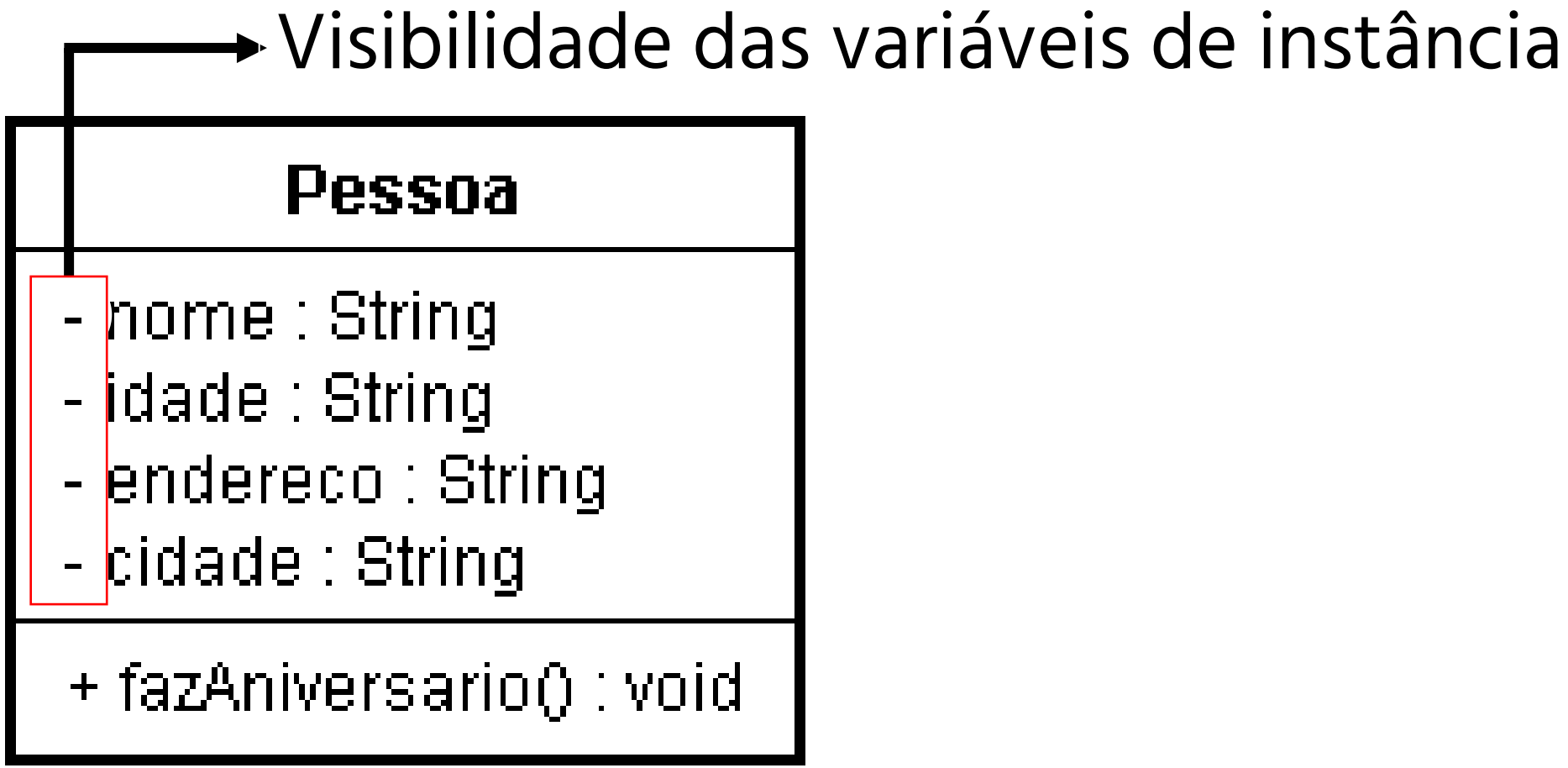


Diagrama da Classe Pessoa



Lista de Métodos

Diagrama da Classe Pessoa



Visibilidade (Modificadores de Acesso)

- private
+ public
# protected

Modificadores de Acesso

- **Podemos utilizar** modificadores de acesso nas **classes, métodos** e **objetos.**
- Exemplos de modificadores de acesso: **public**, **private** e **protected**
  - Públicos (public) – Podem ser utilizados por **membros** de qualquer classe.
  - Protegidos (protected) – Podem ser usados por **membros de uma classe** e em qualquer subclasse.
  - Privados (private) – Podem ser usados por **membros** de UMA classe (apenas a classe que os declarou).

MEMBROS DE UMA CLASSE SÃO OS CAMPOS (VARIÁVEIS DE INSTÂNCIA), MÉTODOS, EVENTOS OU PROPRIEDADES QUE PODEM SER CHAMADOS EM UMA CLASSE.

Modificadores de Acesso

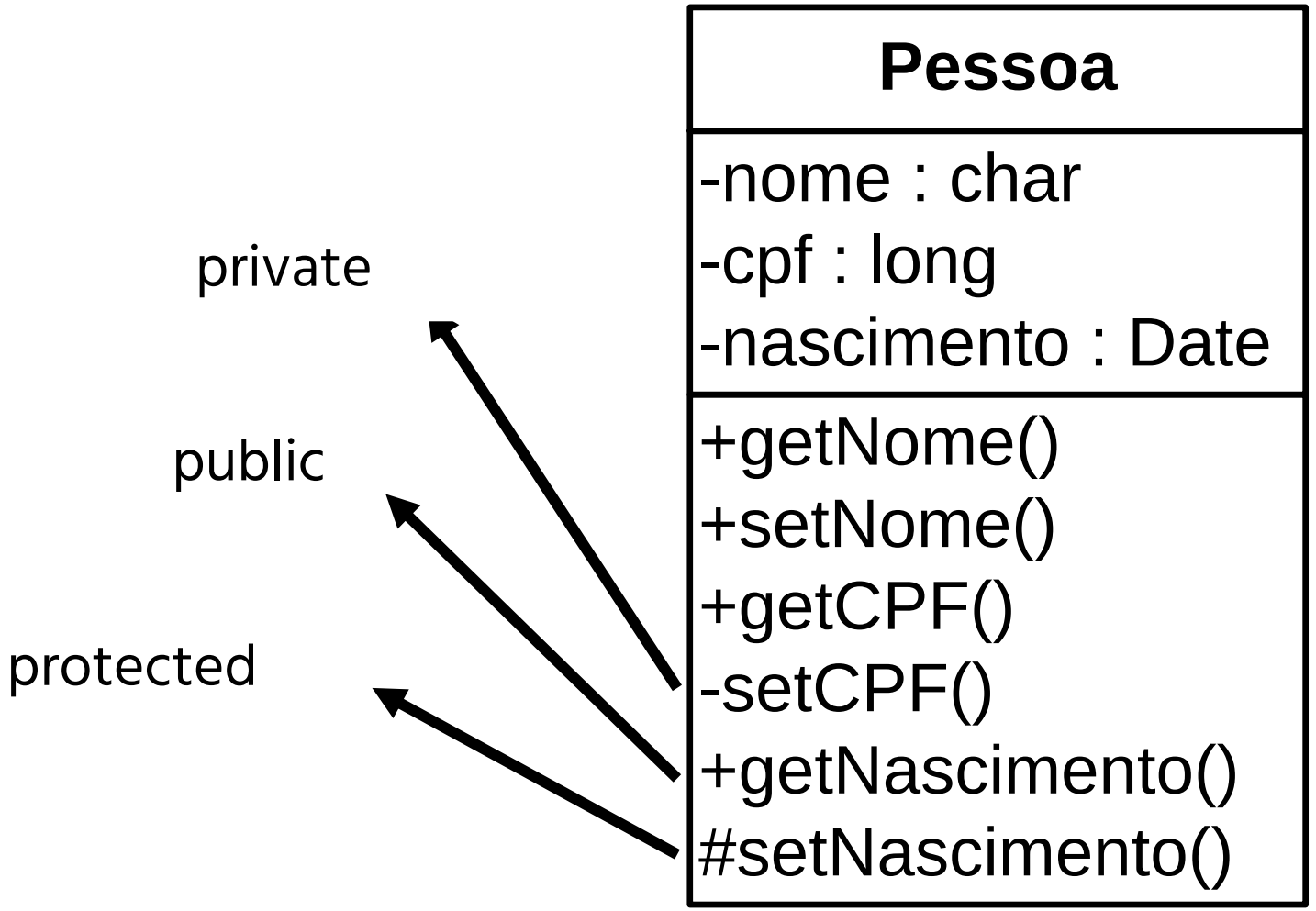


Diagrama da Classe Pessoa

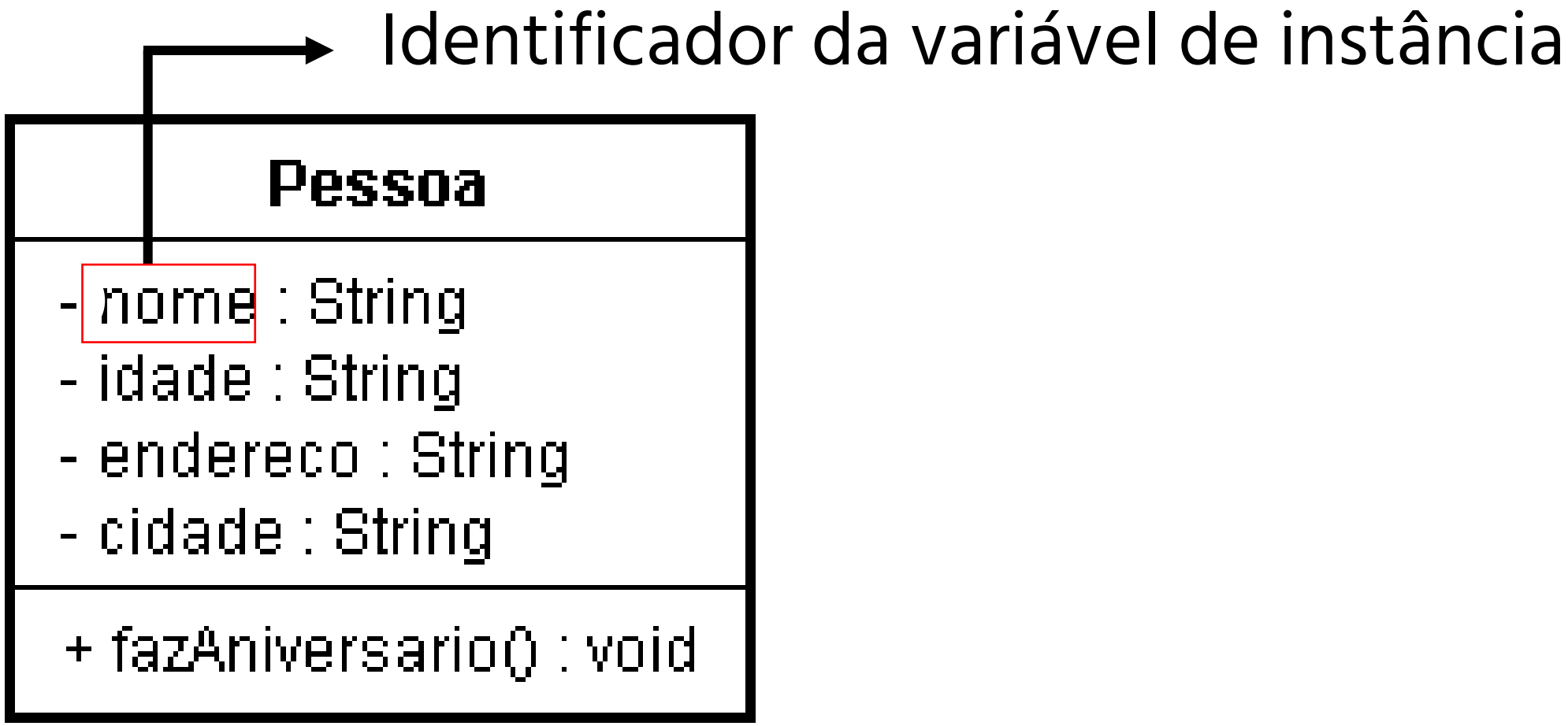
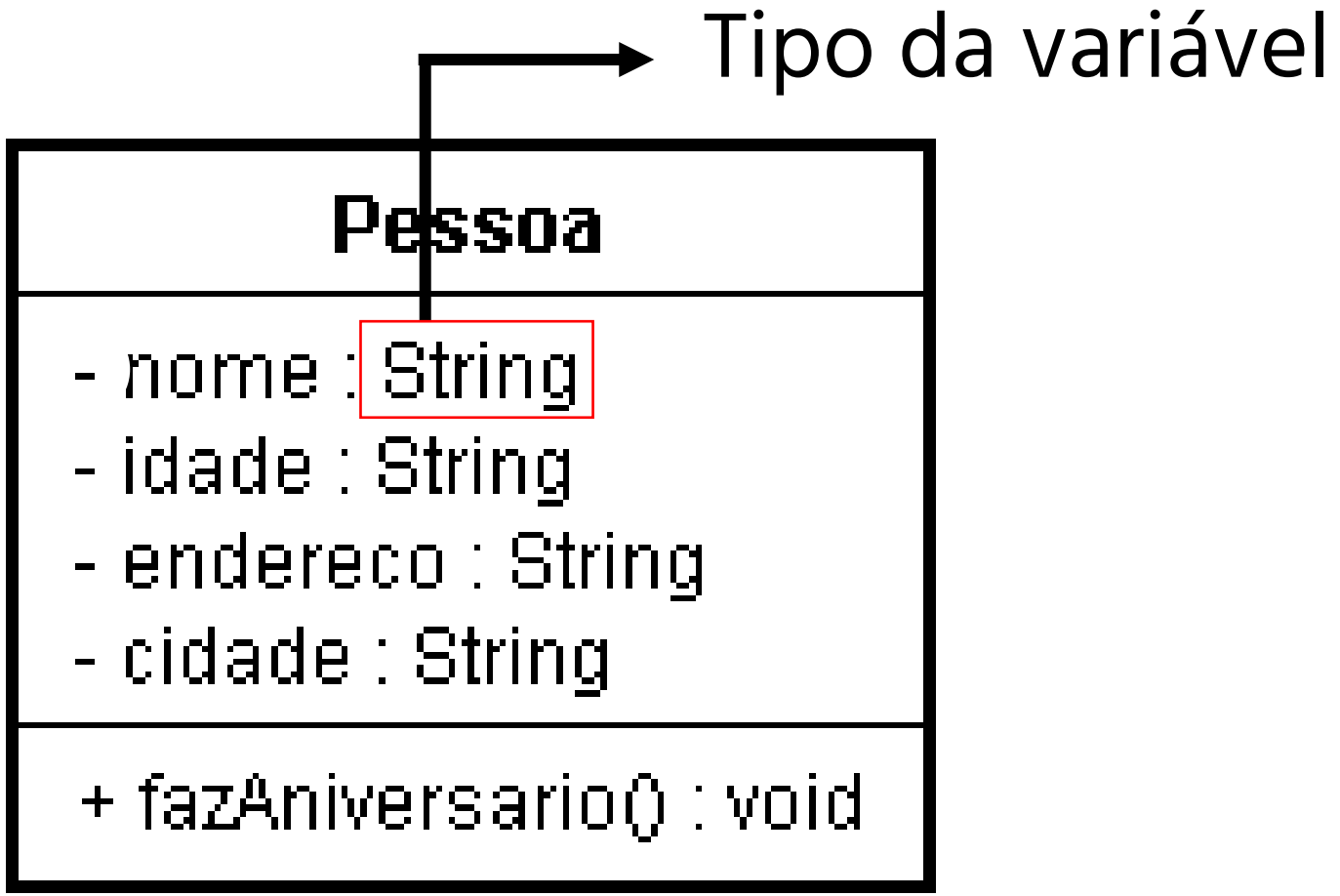
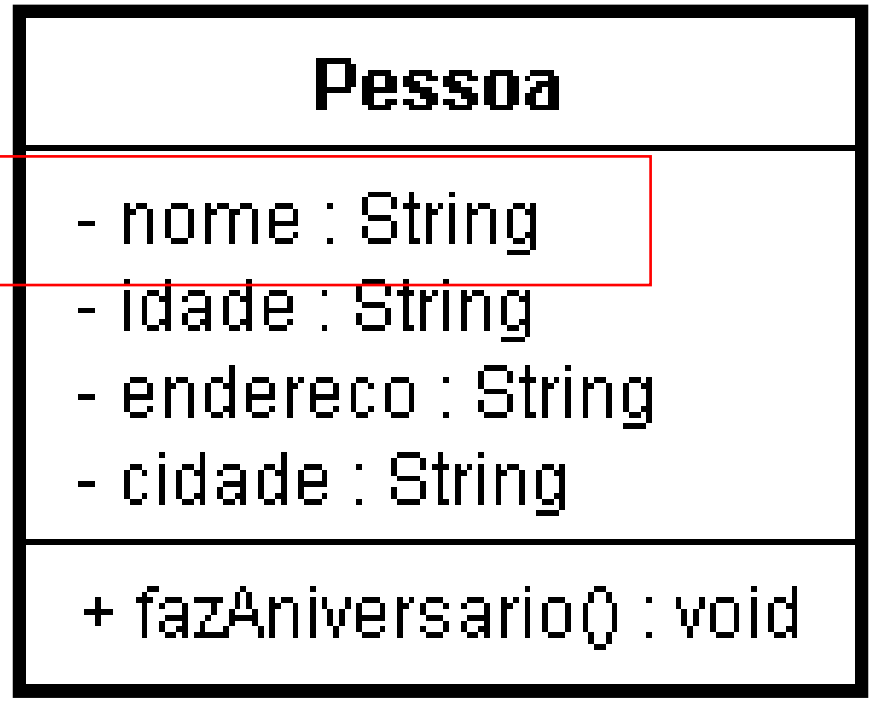


Diagrama da Classe Pessoa



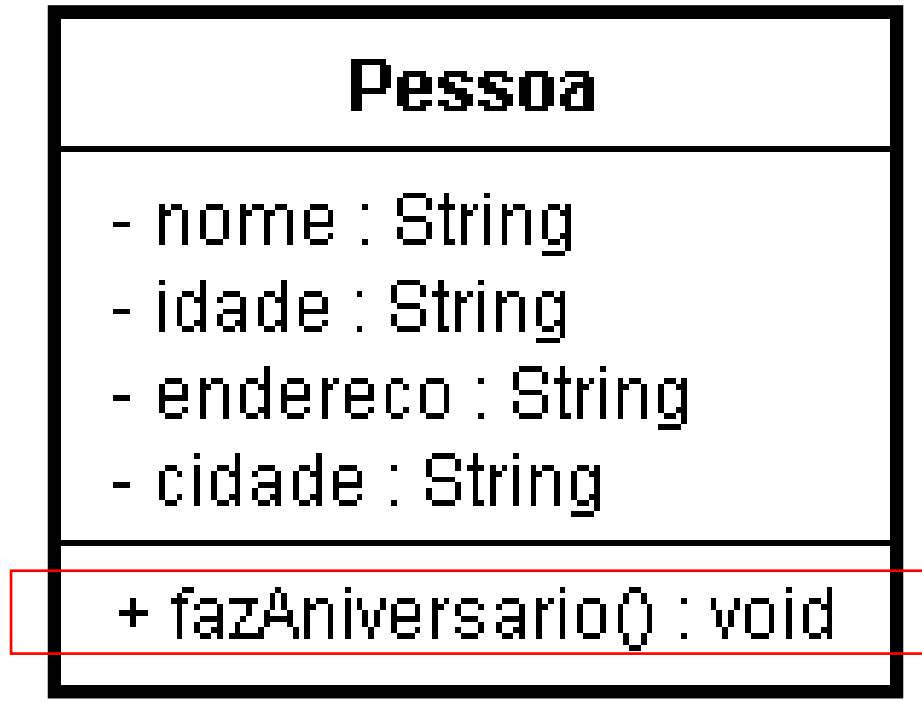
Para construir as variáveis de instância...

- Insira o modificador de acesso: +, - ou #.
- Adicione o identificador da variável.
- Escolha o tipo de variável.
- Separe o identificador e o tipo de dado da variável por dois pontos.



Para construir métodos...

- Insira o modificador de acesso: +, - ou #.
- Insira o nome do método (**com semântica**).
- Caso tenha parâmetros, coloque os tipos deles entre parênteses, separando-os por vírgula, por exemplo: **+ soma(int, int): int**
- Insira o tipo do retorno. Caso não tenha retorno, identifique como void.
- Separe o nome do método e o tipo de retorno por dois pontos.




POR ENQUANTO, IREMOS PROGRAMAR AS **CLASSES** E **MÉTODOS** COM O ACESSO **PÚBLICO** E AS **VARIÁVEIS DE INSTÂNCIA** COM ACESSO **PRIVADO**.

Boas Práticas

- Como boas práticas (*best practices*) do Java, na maioria das declarações de variáveis de instância são definidos os seus atributos com a palavra reservada **private**, para garantir a **segurança de alterações acidentais**, sendo somente acessíveis através dos métodos.
- Essa ação tem como efeito ajudar no **encapsulamento** dos dados, **preservando ainda mais a segurança** e a aplicação de programação orientada a objetos do Java.
- Nas próximas aulas abordaremos o encapsulamento.

Uma classe não vive sozinha num diagrama de classes.
O diagrama é composto por várias classes.

Vamos a outro exemplo...

Classe Androide e seus possíveis objetos

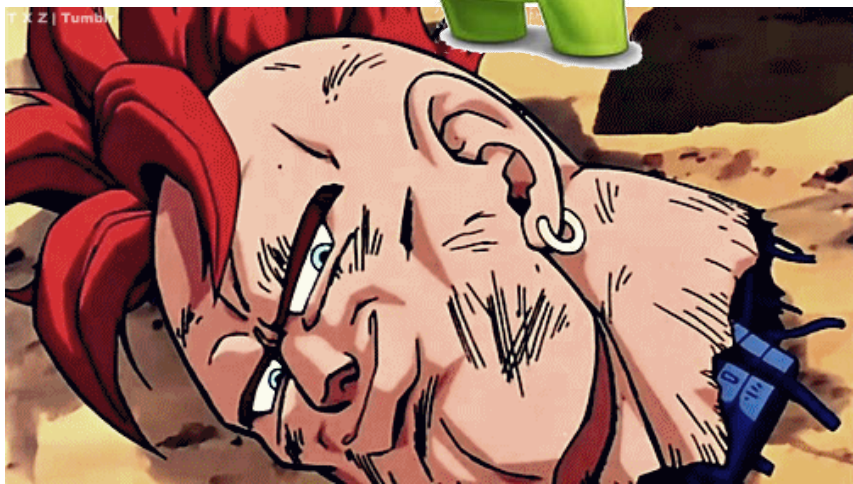
Androide
-nome: String -altura: float -peso: float -poder_de_luta: int
+liberarPoder(int): int +aplicarGolpe(String): int

-nome: 17
-altura: 1,75
-peso: 70
-poder de luta: 45.000.000

-nome: 18
-altura: 1,70
-peso: 60
-poder de luta: 40.000.000

-nome: Android
-altura: 0,15
-peso: 0,5
-poder de luta: 20

~~a16.destroy();~~



~~destroy()?~~

- No Java, **não existe** um método nativo chamado destroy() para destruir objetos ou liberar recursos de forma explícita. **Por quê?**
- O Java utiliza um **coletor de lixo automático** que gerencia a memória alocada para objetos e os libera automaticamente quando não são mais necessários.

- Embora o método destroy() não exista, existem outras maneiras de gerenciar a memória e finalizar objetos em Java, como por exemplo, o método **finalize()** que pode ser implementados para executar ações quando um objeto está prestes a ser coletado pelo coletor de lixo, no entanto, seu uso não é recomendado.

O mais interessante, é que os objetos podem fazer parte da construção de outros objetos...

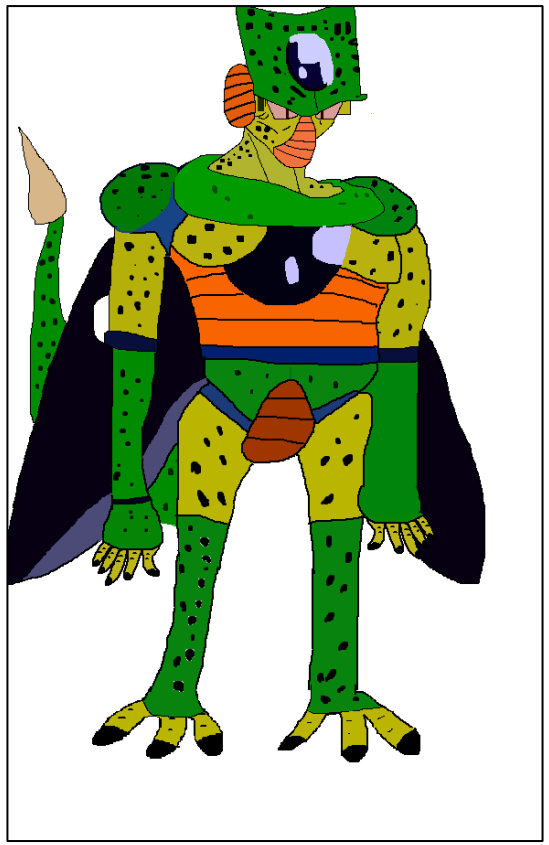
A história contada a seguir utiliza personagens do universo de Dragonball. Dragonball é uma série criada por Akira Toriyama. A utilização desses personagens para esta explicação não tem fins lucrativos.

Era uma vez...
um Androide chamado Cell.

Cell vivia sozinho e estava muito triste por ter um baixo poder de luta...

O que Cell fez?

???



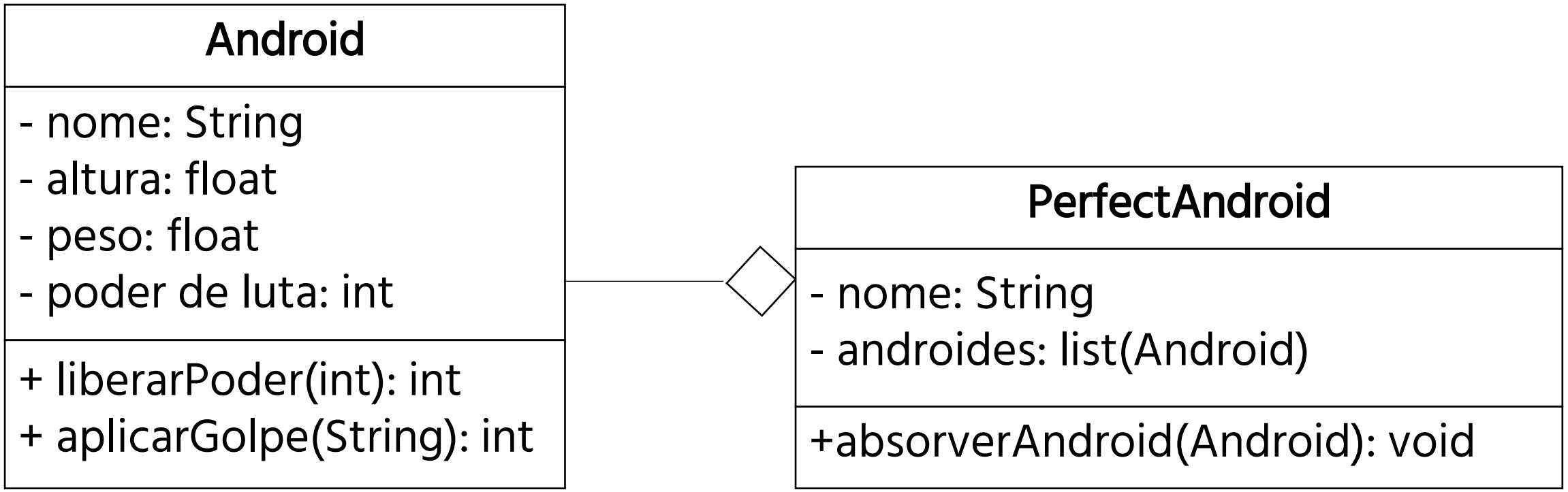
Para aumentar suas habilidades, Cell absorveu os Androides 17 e 18!

E assim, se transformou em algo muito mais poderoso!

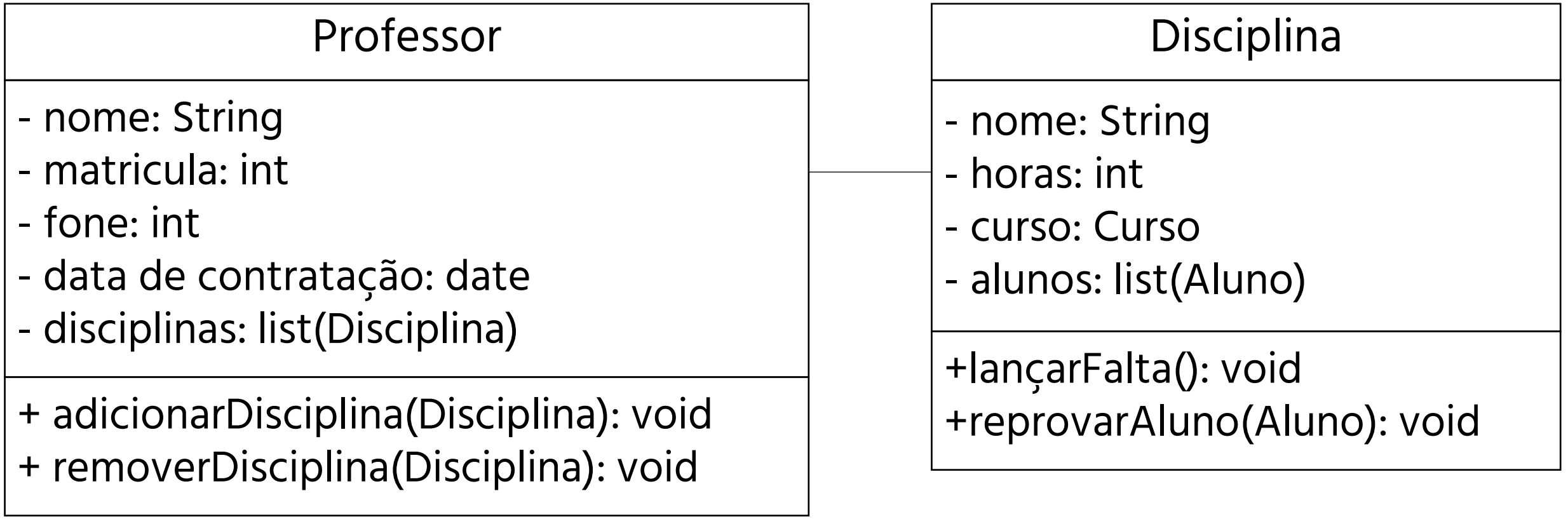
Cell sozinho,
nunca poderia ser tão forte!

Moral da História

- Tudo isso é pra dizer que as **variáveis de instância de uma classe podem ser formadas por objetos de outras classes**!
- Declaramos um objeto como sendo variável de instância de outro!



Exemplo de um diagrama de classes



Associação, Agregação e Composição

Associação: simples associação entre objetos.

- Exemplo: Pessoa possui Carro.

Agregação: Associa um objeto ao outro, mas se forem separados, não resulta na destruição de uma das partes.

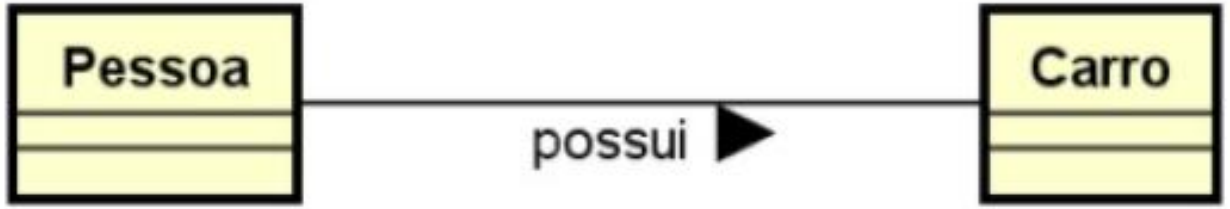
- Exemplo: Carro possui Motor.

Composição: associa objetos de uma forma que ao serem separados, um deixa de existir.

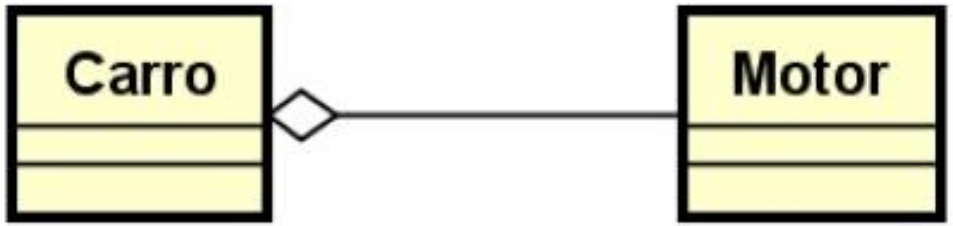
- Exemplo: Funcionário possui Dependente.

Diagrama de Classes

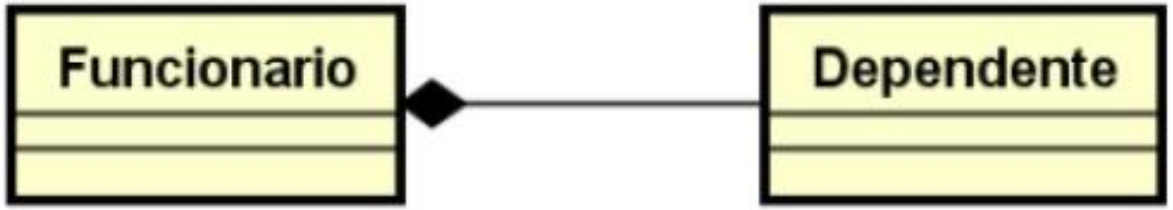
- Associação



- Agregação



- Composição



Intervalo

15 minutos


ATÉ AGORA, JÁ SABEMOS O QUE É UMA CLASSE, UM OBJETO, OS MÉTODOS E ALGUNS TERMOS UTILIZADOS NA POO.
Puff puff.

AGORA VAMOS VER COMO **ESSES ELEMENTOS CONVERSAM ENTRE SI** PARA RESOLVER OS PROBLEMAS DO MUNDO REAL!

Os métodos gets e sets!

Os Métodos gets e sets

- Um método é uma função evocada (ou invocada?) por um objeto.
- Os métodos gets e sets são os "**métodos padrões**" para **acessar** (get) e **modificar** (set), respectivamente, as **variáveis** de instância de uma classe.
- Os métodos **sets não retornam valor**. São void. Eles mudam apenas o conteúdo da variável de instância de um objeto.
- Já **os métodos gets retornam** alguma coisa. O que vai ser retornado, vai depender do tipo da variável de instância (por exemplo, se a variável de instância for int, o método vai retornar um int).

Abram o Eclipse...

Considere a classe Retângulo

*Principal.java *Retângulo.java

```java

public class Retângulo {
    private int largura;
    private int altura;

    public int getLargura() {
        return largura;
    }

    public void setLargura(int largura) {
        this.largura = largura;
    }

    public int getAltura() {
        return altura;
    }

    public void setAltura(int altura) {
        this.altura = altura;
    }

    public int calcularPerimetro() {
        return ( (2*getAltura()) + (2*getLargura()) );
    }

    public int calcularArea() {
        return (this.altura * this.largura);
    }

    @Override
    public String toString() {
        return "Retângulo [largura=" + largura + ", altura=" + altura + "]";
    }

}
```

Os Métodos gets e sets

*Principal.java | Retângulo.java

```
1
2  public class Principal {
3
4      public static void main(String[] args) {
5          Retângulo r1 = new Retângulo();
6          r1.setLargura(10);
7          r1.setAltura(20);
8
9          System.out.println("Altura: "+r1.getAltura());
10         System.out.println("Largura: "+r1.getLargura());
11         System.out.println("Perímetro: "+r1.calcularPerimetro());
12         System.out.println("Área: "+r1.calcularArea());
13         System.out.println(r1);
14     }
15 }
```

Problems | Javadoc | Declaration | Console

<terminated> Principal (38) [Java Application] C:\Program Files\Java\jdk-16.0.2\bin\javaw.exe (18 de mai. de 2022 09:55

```
Altura: 20
Largura: 10
Perímetro: 60
Área: 200
Retângulo [largura=10, altura=20]
```

São os métodos que modificam as variáveis de instância de uma classe.

O método **setLargura()** é responsável por modificar o valor da largura do objeto **r1**. Já o **getLargura()** recupera o valor da largura do objeto **r1**.

E QUAL O ATALHO NO ECLIPSE PARA CONSTRUIRMOS OS MÉTODOS GETS E SETS DE UMA CLASSE?

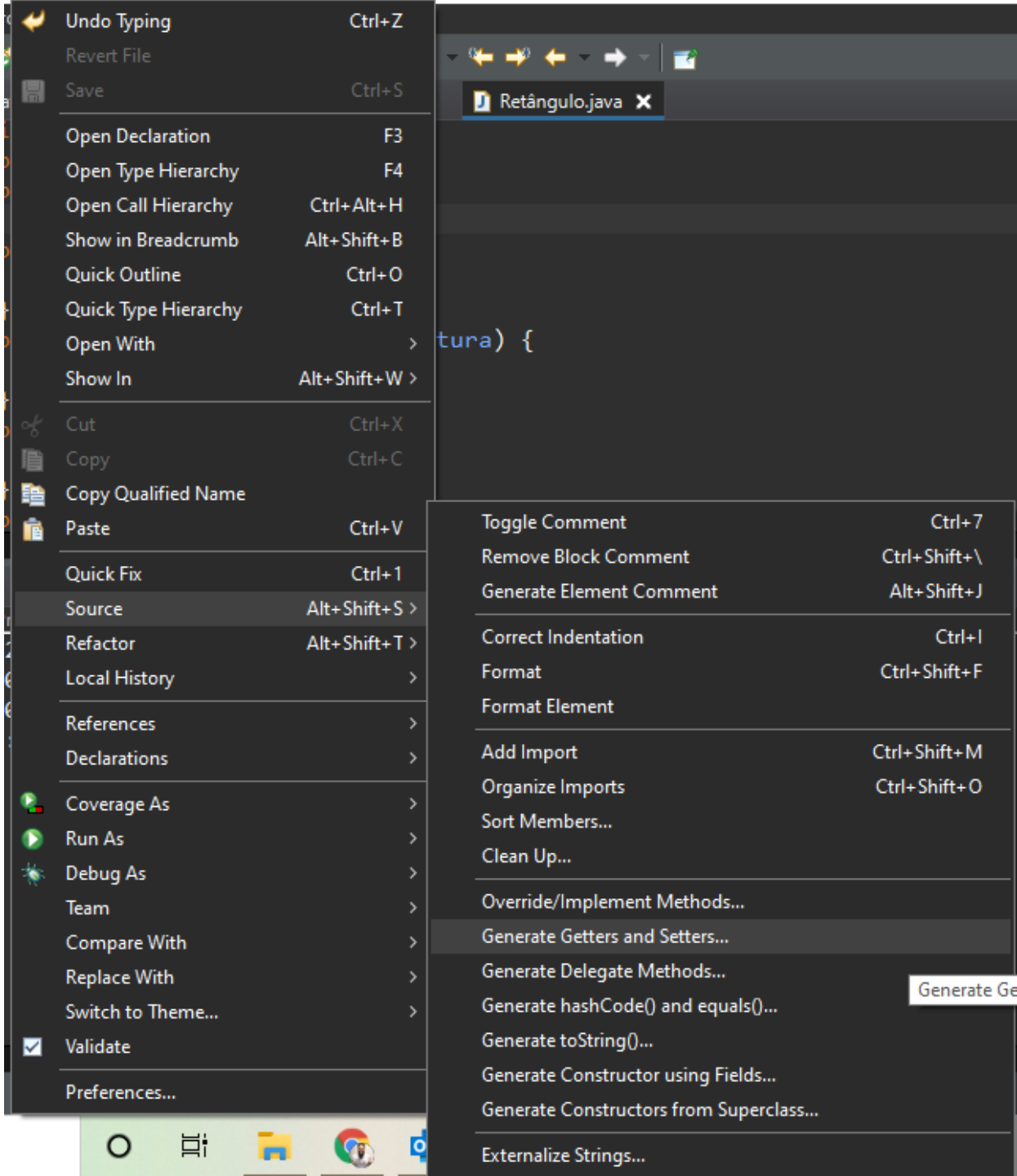
E QUAL O ATALHO NO ECLIPSE PARA CONSTRUIRMOS OS MÉTODOS GETS E SETS DE UMA CLASSE?

ALT +

SHIFT+

S + R

OU ENTÃO CLICA COM O BOTÃO DIREITO DO MOUSE E ESCOLHE A OPÇÃO **SOURCE**, EM SEGUIDA **GENERATE GETTERS AND SETTERS**.

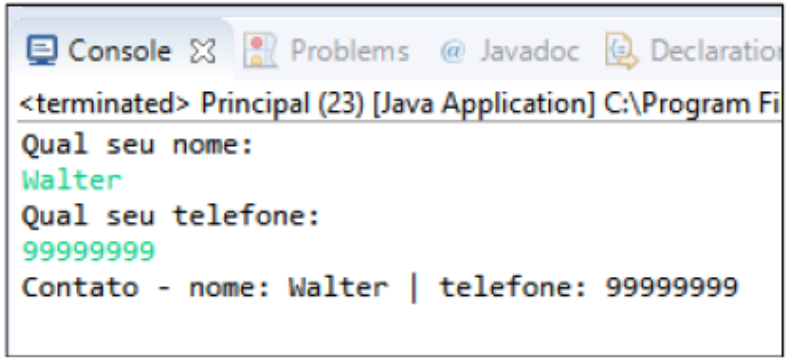


Exercícios

Exercício – Parte 1

- Crie uma classe Java chamada Contato...
  - Variáveis de instância: nome, telefone.
    - Todos são Strings.
    - O acesso aos atributos é privado.
  - Métodos:
    - setNome (público, sem retorno, com um parâmetro String)
    - setTelefone (público, sem retorno, com um parâmetro String)
    - getNome (público, com retorno String, sem parâmetro)
    - getTelefone (público, com retorno String, sem parâmetro)
    - exibeContato (público, sem retorno, sem parâmetro)

Exercício – Parte 2



- Crie uma classe Java chamada Principal, que...
  - Terá o método main implementado.
  - Instancie um objeto da classe Scanner (para receber as entradas) e um da classe Contato (para manipular nossa agenda).
    - OBS: Instanciar = Criar os objetos de cada classe
  - Receberá o nome e o telefone digitados pelo usuário.
  - Irá enviar os dados para a classe Contato.
  - Fará com que o objeto da classe Contato exiba os dados digitados pelo usuário.

Resposta

*Principal.java *Contato.java

```java

public class Contato {
    private String nome;
    private String telefone;

    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getTelefone() {
        return telefone;
    }
    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }

    public void exibeContato() {
        System.out.println("Contato - nome: "+this.nome+ " | telefone: "+this.telefone);

    }
}

```

*Principal.java *Contato.java

```java
import java.util.Scanner;
public class Principal {

    public static void main(String[] args) {

        Scanner entrada = new Scanner (System.in);
        Contato c1 = new Contato();

        System.out.println("Qual seu nome: ");
        String nome = entrada.nextLine();
        c1.setNome(nome);

        System.out.println("Qual seu telefone: ");
        c1.setTelefone(entrada.nextLine());

        c1.exibeContato();

        entrada.close();
    }
}

```

Console Problems @ Javadoc Declaratio

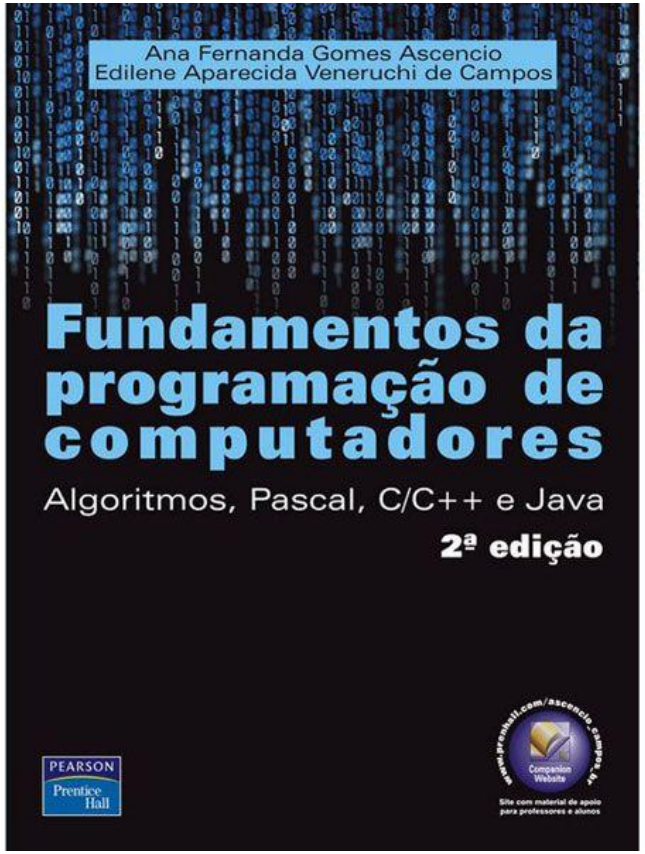
```
<terminated> Principal (23) [Java Application] C:\Program Fi
Qual seu nome:
Walter
Qual seu telefone:
99999999
Contato - nome: Walter | telefone: 99999999
```

Lista de Exercícios 7

Explore o assunto!

Referências

ASCENCIO, A. F. G., CAMPOS, E. A. V. **Fundamentos da Programação de Computadores**: Algoritmos, Pascal, C/C++ e Java - 2. ed. / 2008 - São Paulo (SP): Pearson Prentice Hall, 2008.



Referências

MANSOOUR, I. H. Paradigmas de Linguagens I. [Internet]. [citado em 2014 Feb 04]. Disponível em: https://www.inf.pucrs.br/~gustavo/disciplinas/pli/material/paradigmas-aula12.pdf.

PROGRAMAÇÃO ORIENTADA A OBJETOS

AULA 9 – O PARADIGMA ORIENTADO A OBJETOS

WALTER TRAVASSOS SARINHO

@PROF.WALTERTRAVASSOS

WALTER.TRAVASSOS@SECTRAS.EDU.BR