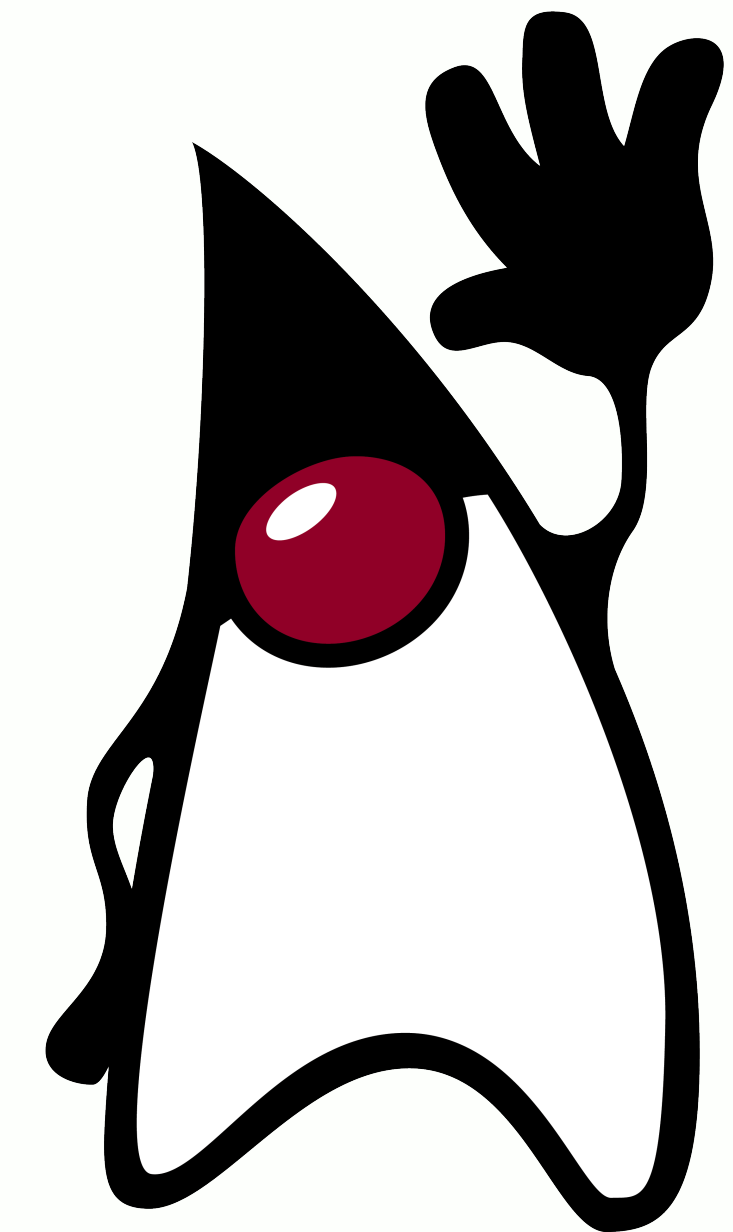




UNIVERSIDADE  
FEDERAL DO CEARÁ  
CAMPUS QUIXADÁ

# Classes, Objetos e Encapsulamento

QXD0007 - Programação Orientada a Objetos

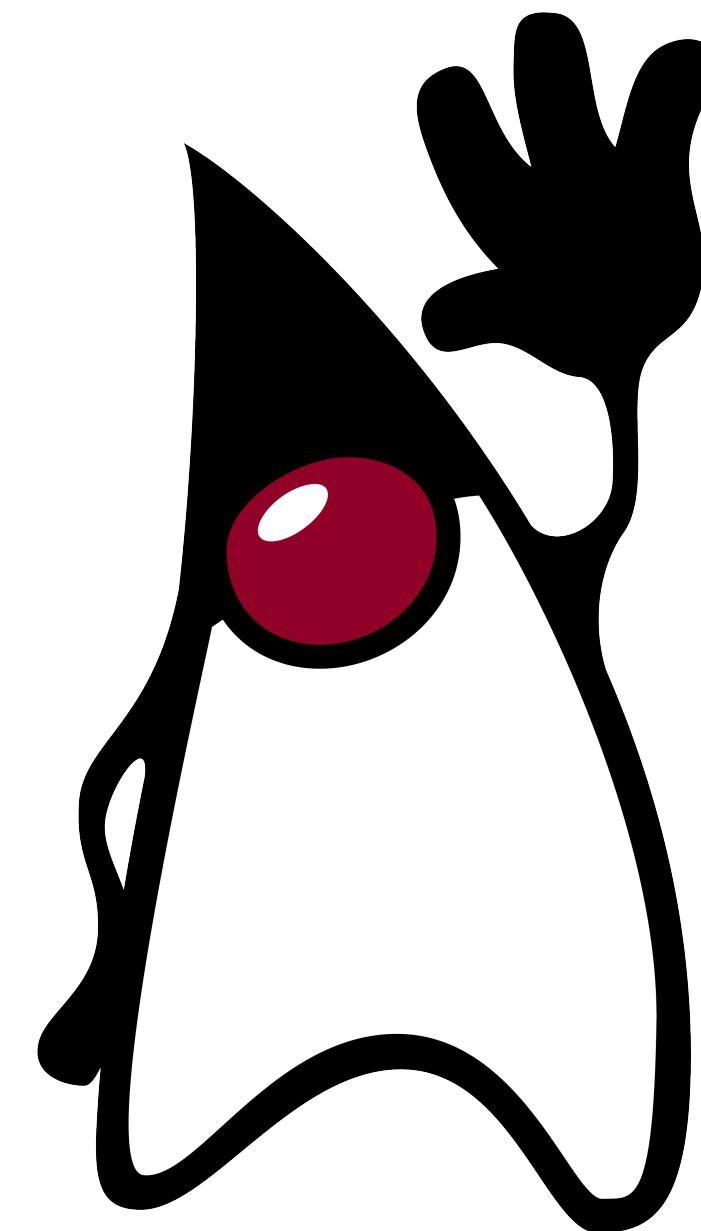


Prof. Bruno Góis Mateus ([brunomateus@ufc.br](mailto:brunomateus@ufc.br))

# Conteúdo

- Classes e Objetos
- Atributos
- Métodos
- Encapsulamento e Modificadores de acesso

# Classes e Objetos

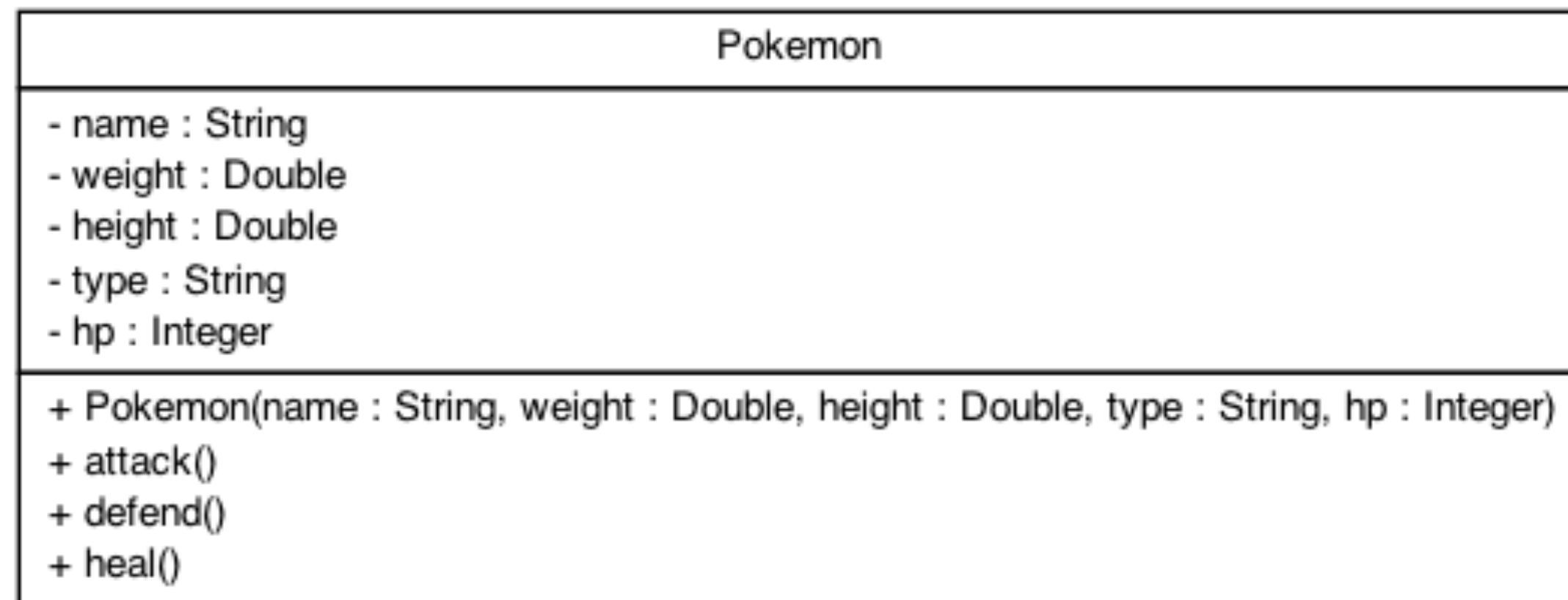


# Classes e objetos

- Representa um conjunto de objetos com características afins
- Define
  - Quais estados eles podem alcançar (através seus atributos)
  - O comportamento dos seus objetos (através de seus métodos)

# Classes e objetos

## Representação UML



Nome da classe

Propriedades / Atributos

Operações / Métodos

# Classes e objetos

## Assinatura

Pokemon
<ul style="list-style-type: none"><li>- name : String</li><li>- weight : Double</li><li>- height : Double</li><li>- type : String</li><li>- hp : Integer</li></ul>
<ul style="list-style-type: none"><li>+ Pokemon(name : String, weight : Double, height : Double, type : String, hp : Integer)</li><li>+ attack()</li><li>+ defend()</li><li>+ heal()</li></ul>

```
public class Pokemon {  
    // Atributos  
    private String name;  
    private Double weight;  
    private Double height;  
    private String type;  
    private Integer hp;  
    // Métodos  
    public void attack ()  
    public void defend ()  
    public void heal ()  
}
```

# Classes e objetos

## Objeto

- É uma **instância** de uma classe
- É capaz de armazenar **estados** através de seus **atributos** e reagir à **mensagens** enviadas a ele, assim como se relacionar e **enviar mensagens** para outros objetos

# Classes e objetos

## Um objeto é uma entidade independente

- **Estado interno:** uma memória interna em que valores podem ser armazenados e modificados ao longo da vida do objeto (**conjunto de atributos ou variáveis de instância**)
- **Comportamento:** conjunto de ações pré-definidas (métodos) através das quais o objeto responderá a demanda de processamento por parte de outros objetos





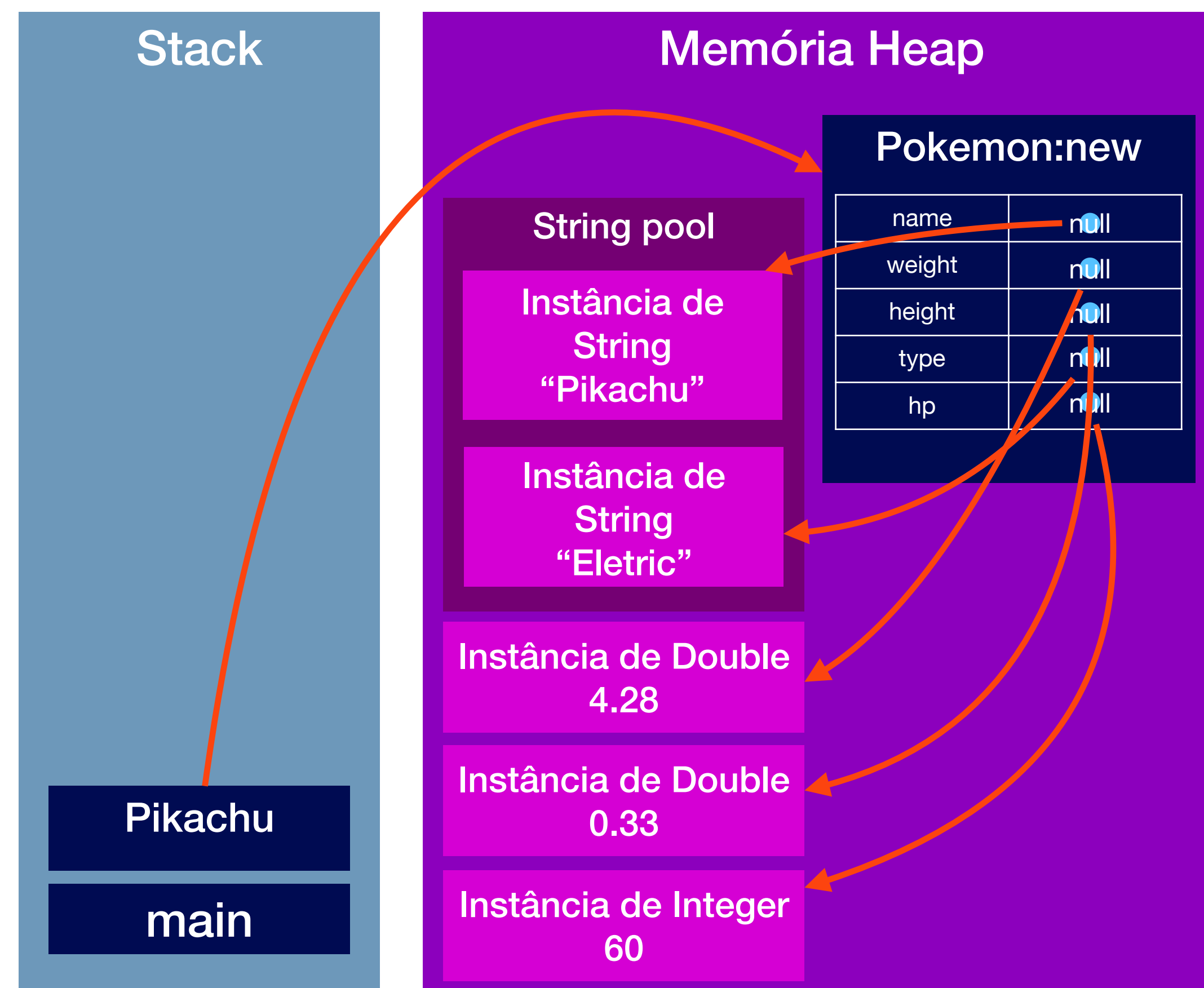
# Classes e objetos

## Instanciação

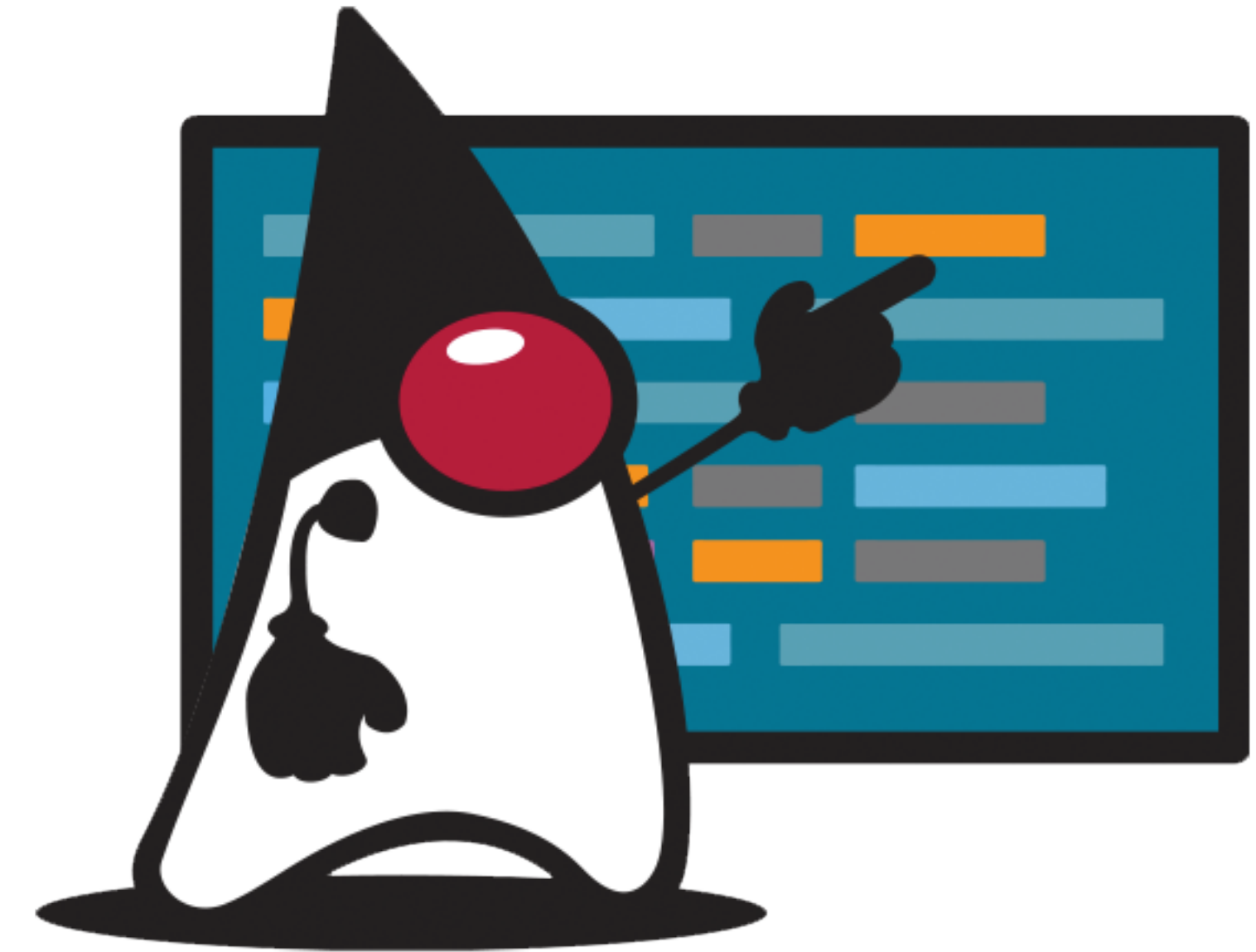
```
public class Pokemon {  
    private String name;  
    private Double weight;  
    private Double height;  
    private String type;  
    private Integer hp;  
  
    public Pokemon(String name, Double weight,  
        Double height, String type, Integer hp) {  
        this.name = name;  
        this.weight = weight;  
        this.height = height;  
        this.type = type;  
        this.hp = hp;  
    }  
}
```

→ 

```
public static void main(String[] args) {  
    Pokemon pikachu = new Pokemon("Pikachu",  
        4.28, 0.33, "Eletric", 60);  
}
```



# Atributos



# Atributos

- São propriedades de um objeto
- Representam os dados que um objeto vai armazenar
- Possuem valores que podem ser alterado durante o ciclo de vida de um objeto



# Atributos

## Atributo de classe

- Em algumas situações **compartilhar atributos entre objetos** é uma boa ideia
- Usando a palavra reservada **static** é possível definir atributos de classe
  - Estes atributos são compartilhados com todos as instâncias dessa classe

```
public class Pokemon {  
    private static int contador = 0;  
}  
}
```

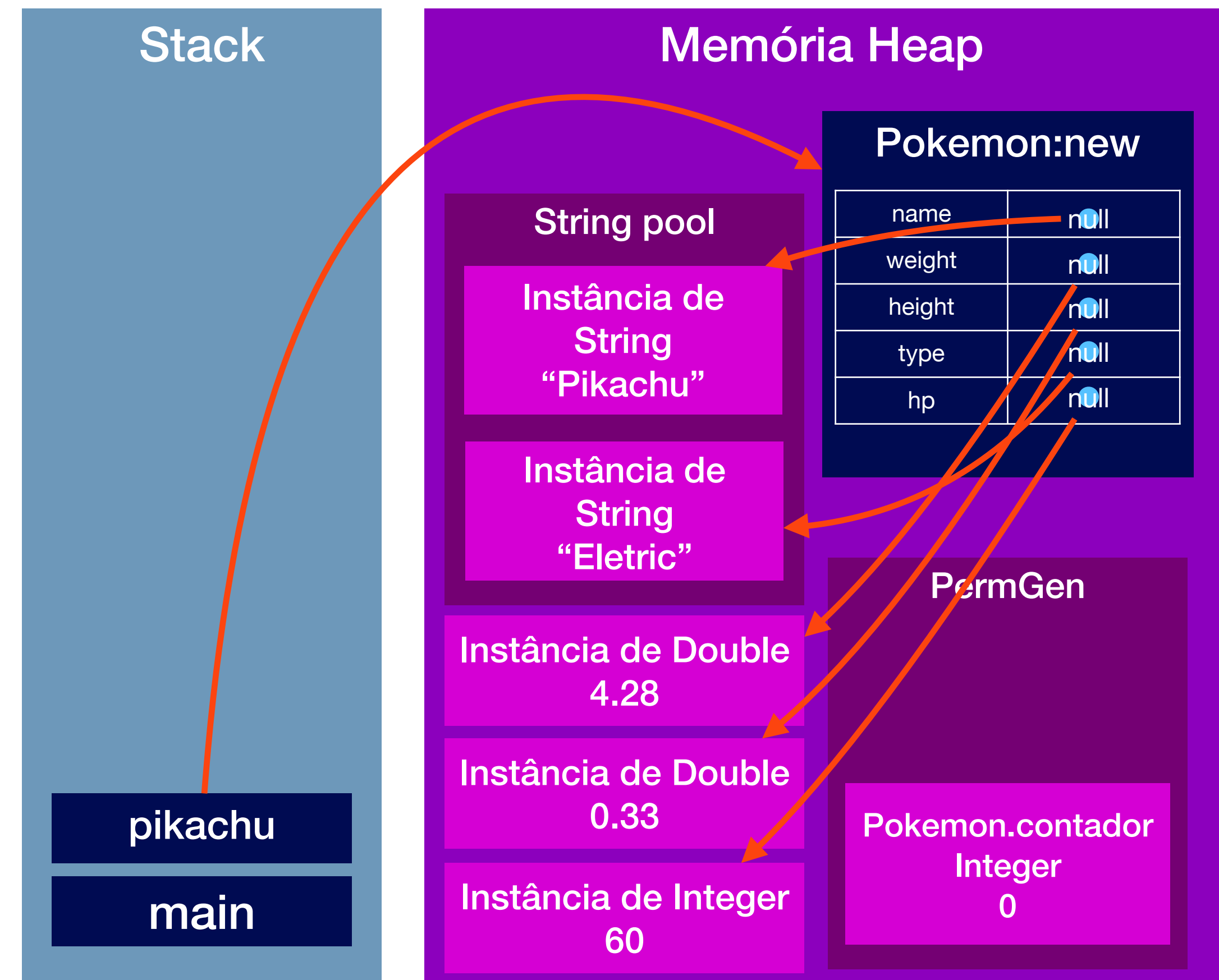
# Atributos

## Atributo de classe

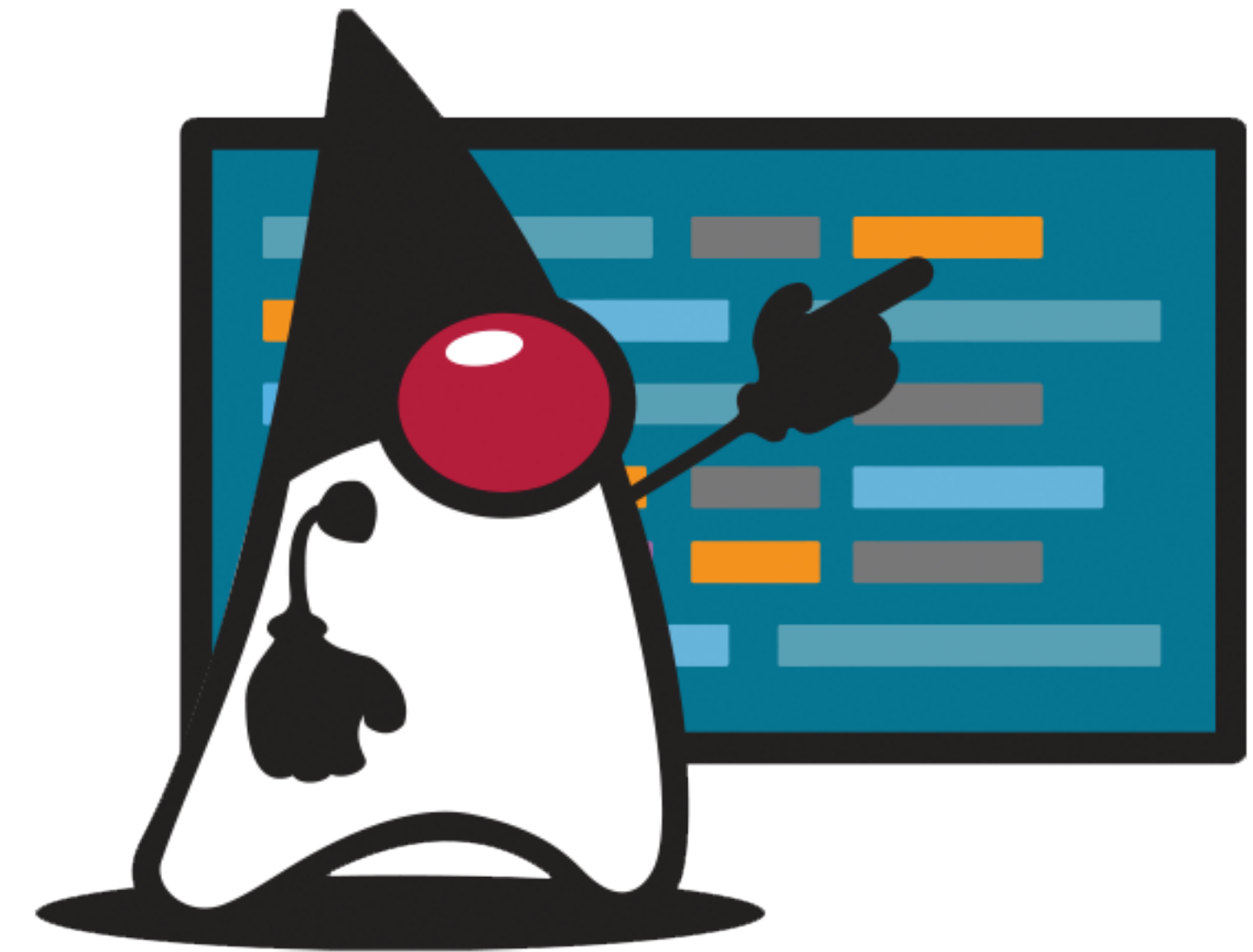
```
public class Pokemon {
    private String name;
    private Double weight;
    private Double height;
    private String type;
    private Integer hp;
    private static Integer contador;

    public Pokemon(String name, Double weight, Double height, String type, Integer hp) {
        this.name = name;
        this.weight = weight;
        this.height = height;
        this.type = type;
        this.hp = hp;
    }

    public static void main(String[] args) {
        Pokemon pikachu = new Pokemon("Pikachu",
            4.28, 0.33, "Eletric", 60);
    }
}
```



# Métodos



# Métodos

- Definem as “habilidades” dos objetos
- A ação definida por métodos **ocorre somente quando um método é invocado**
- Assim como funções em C, métodos podem ou não retornar valores



# Métodos

## Passagem de parâmetros

- O número de parâmetros passados deve ser igual ao da definição
  - Cada parâmetro individualmente, deve ter tipo compatível com o da definição
  - Para tipos primitivos, a passagem é por valor
  - Para vetores (arrays) e objetos, uma cópia da referência é passada
- Uma expressão pode ser usada como argumento



# Métodos

## Número variável de parâmetros

```
public String formatWithVarArgs (String... values) {  
    // ...  
}
```

# Métodos

## O método construtor

- Conceito introduzido por linguagens orientadas a objetos
- Determina as ações que serão executadas durante a criação (**instanciação**) de um objeto
- São **métodos especiais**
  - Possuem o **mesmo nome da classe e não possuem retorno**
  - **Unicamente invocado** no momento da criação por meio do comando **new**
- Fornece uma oportunidade de **inicializar de maneira apropriada o objeto**

# Métodos

## O construtor padrão

- Toda classe em Java possui um construtor, mesmo quando não declaramos nenhum
- Outra ação do construtor padrão é chamar o **construtor** da **superclasse**

```
public class Pokemon {  
  
    public Pokemon() {  
        super();  
    }  
}
```

# Métodos

## A necessidade de um método construtor

- Possibilita ou obriga o usuário de uma classe a passar argumentos durante o processo de criação
  - Ex: Garante que um Pokemon sem nome não seja criado
- Pode fornecer diversas maneiras para se criar um objeto ao definirmos múltiplos construtores
- É uma boa prática definir pelo menos um construtor

# Métodos

## A necessidade de um método construtor

```
public class Pokemon {  
    public Pokemon(String name, Double weight, Double height,  
String type, Integer hp) {  
        this.name = name;  
        this.weight = weight;  
        this.height = height;  
        this.type = type;  
        this.hp = hp;  
    }  
  
    public Pokemon(String name, String type) {  
        this.name = name;  
        this.type = type;  
    }  
}
```

# Métodos

## O método de classe

- Só acessam atributos **estáticos**
- **Não é possível acessar atributos** dos objetos da mesma classe **diretamente**
  - Não temos acesso ao **this**, já que o método é **chamado via classe**

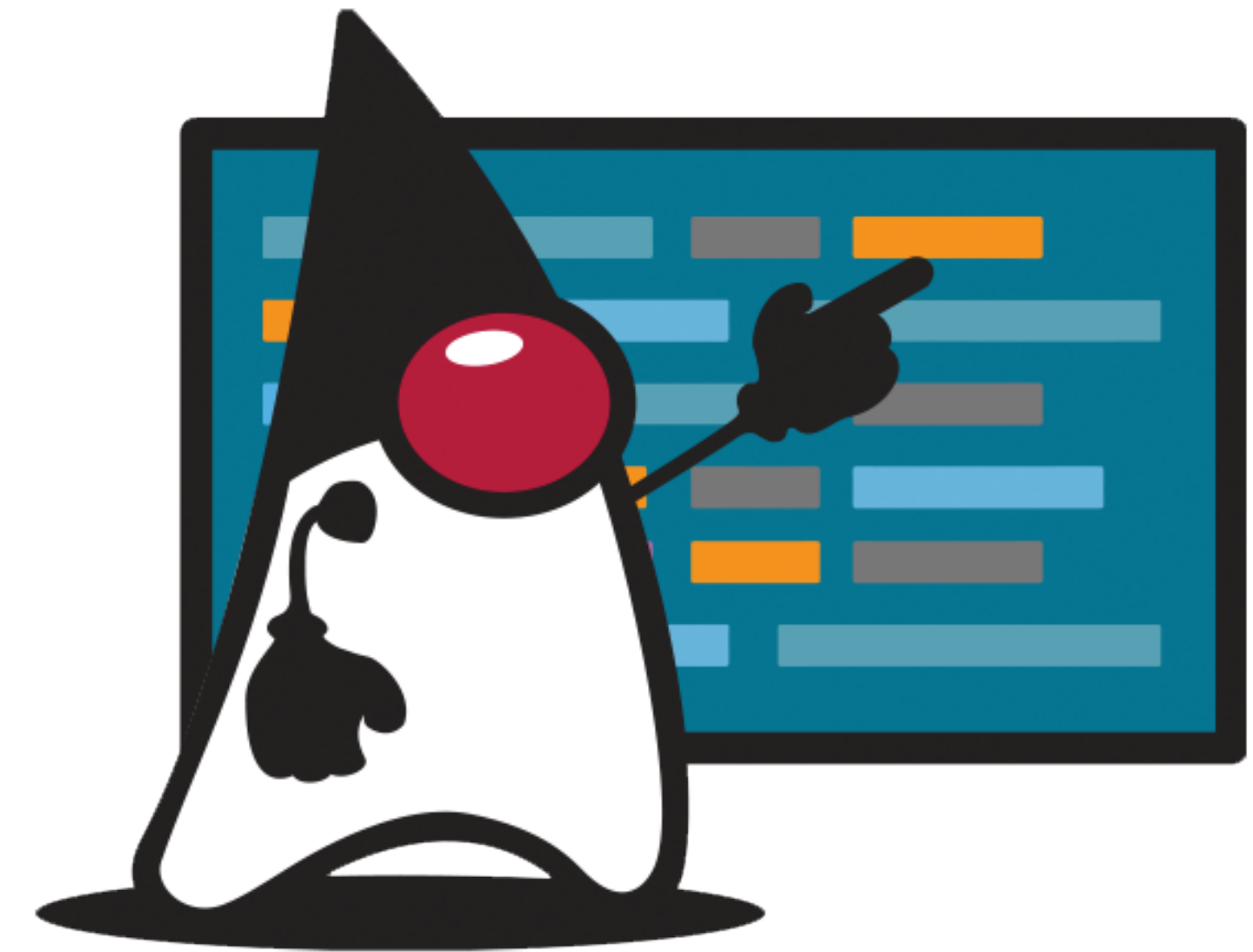
# Métodos

## O método de classe

```
public class Conversor {  
  
    // Método de classe para converter de Celsius para Fahrenheit  
    public static double celsiusParaFahrenheit(double celsius) {  
        return (celsius * 9/5) + 32;  
    }  
    // Método de classe para converter de Fahrenheit para Celsius  
    public static double fahrenheitParaCelsius(double fahrenheit) {  
        return (fahrenheit - 32) * 5/9;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        double fahrenheit = Conversor.celsiusParaFahrenheit(25);  
        System.out.println("25°C em Fahrenheit é: " + fahrenheit);  
    }  
}
```

# Encapsulamento e Modificadores de acesso





# Encapsulamento e Modificadores de acesso

## Encapsulamento

- É uma forma de restringir o acesso ao estado interno do objeto e permitir a colaboração apenas via mensagens
- Boa parte das linguagens orientadas a objetos da suporte a esse conceito
  - Escondendo o membros (atributos e métodos) de uma classe
  - Escondendo o funcionamento das rotinas(métodos) do sistema
- É fundamental quando para desenvolver sistemas suscetíveis a mudanças

# Encapsulamento e Modificadores de acesso

- O conjunto de **métodos públicos** de uma classe são também chamados de **interface da classe**
  - É a única maneira a qual você se comunica com objetos dessa classe
- Cada classe especifica sua **interface de inicialização e operação**
- Na maioria das linguagens OO, os métodos partes da interface são **public**

# Encapsulamento e Modificadores de acesso

## Programando para a interface

- É sempre bom programar pensando na interface
  - Como os usuário irão utilizar
  - A implementação em si não tem tanta importância
- O usuário precisa saber usar a a classe
  - Não precisa se preocupar como é feito
  - A implementação pode mudar com o tempo

# Encapsulamento e Modificadores de acesso

## Programando para a interface

- No momento de projetar a classe, sempre pense na perspectiva do usuário
  - A classe em contempla todas as necessidades do usuário
- Forneça apenas o necessário para o usuário, **uma interface mínima**
  - É melhor adicionar interfaces do que fornecer mais do que o necessário
  - **Tenha certeza que à medida que o sistema evoluir, a sua classe pode evoluir também**

# Encapsulamento e Modificadores de acesso

## Default

- Para esse modificador **não há uma palavra chave**
  - O modificador aqui **é a omissão dos outros modificadores**
- O método ou atributo fica num **estado de visibilidade intermediário entre private e o public**
- Restringe o acesso ao item (atributo ou método) sob o modificador **deixando-o visível somente para as classes que se encontram no mesmo pacote da classe que possui o item**

# Encapsulamento e Modificadores de acesso

## Public

- O modificador de acesso `public` permite a todos acessarem um método ou atributo
- Em geral não usamos esse modificador em atributos

# Encapsulamento e Modificadores de acesso

## Private

- Marcando o atributo como **privado** evitamos o acesso ao mesmo
  - Apenas a própria classe pode acessá-lo diretamente
- Na orientação a objetos, é prática **quase que obrigatória** proteger seus atributos com **private**
- É muito comum, e faz todo sentido, que seus atributos sejam **private** e quase todos seus métodos sejam **public** (**não é uma regra!**)
  - Toda conversa de um objeto com outro é feita por troca de mensagens



# Encapsulamento e Modificadores de acesso

## Getters e Setters

- São usados para proteger dados de classes
  - Para cada atributo, um método **getter** retorna seu valor, enquanto um método **setter** o define ou atualiza.
- Convenção:
  - **getters** começam com a palavra "**get**" e **setters** com a palavra "**set**", seguidos de um nome de variável
  - Em ambos os casos, a primeira letra do nome da variável será maiúscula



# Encapsulamento e Modificadores de acesso

## Getters e Setters

```
public class Pokemon {  
  
    private String nome;  
    private int nivel;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public int getNivel() {  
        return nivel;  
    }  
  
    public void setNivel(int nivel) {  
        this.nivel = nivel;  
    }  
  
}
```

# Encapsulamento e Modificadores de acesso

## Private

- Cada classe é responsável por seus atributos
  - Ela deve decidir se um novo valor é válido ou não
- Centralizar essa responsabilidade facilita mudanças futuras
- É comum que esconder a existência de um atributo por completo
  - Nesses casos tal atributo diz respeito ao funcionamento interno do objeto

# Encapsulamento e Modificadores de acesso

## Private

- O modificador `private` também pode ser aplicado a um método
  - Quando existe um método que serve apenas para auxiliar a própria classe
    - Ex: Quando há código repetido dentro de dois métodos da classe
- Sempre devemos expôr o mínimo possível de funcionalidades, para criar um baixo acoplamento entre as nossas classes

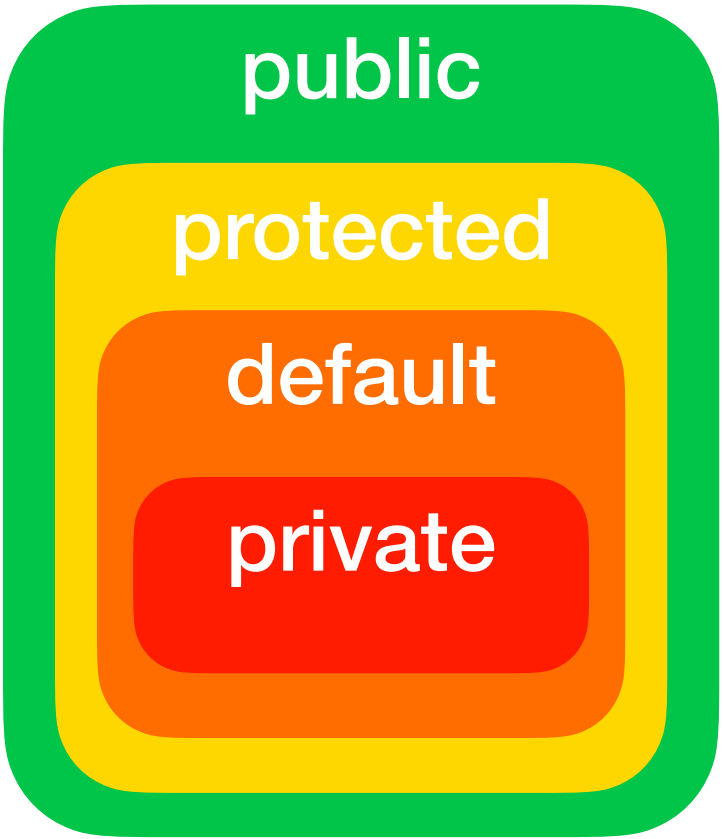
# Encapsulamento e Modificadores de acesso

## Protected

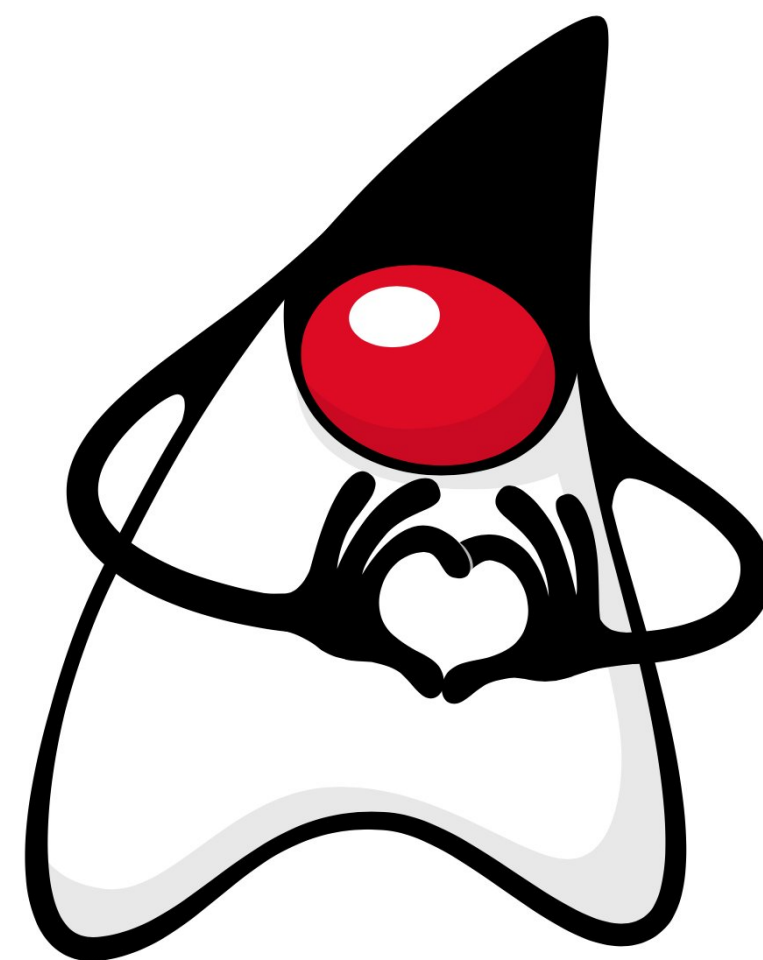
- Restringe o acesso ao item sob o **modificador deixando-o visível somente para as classes filhas** da classe que possui o item e **para classes que se encontram sob o mesmo pacote** da classe que possui o item

# Encapsulamento e Modificadores de acesso

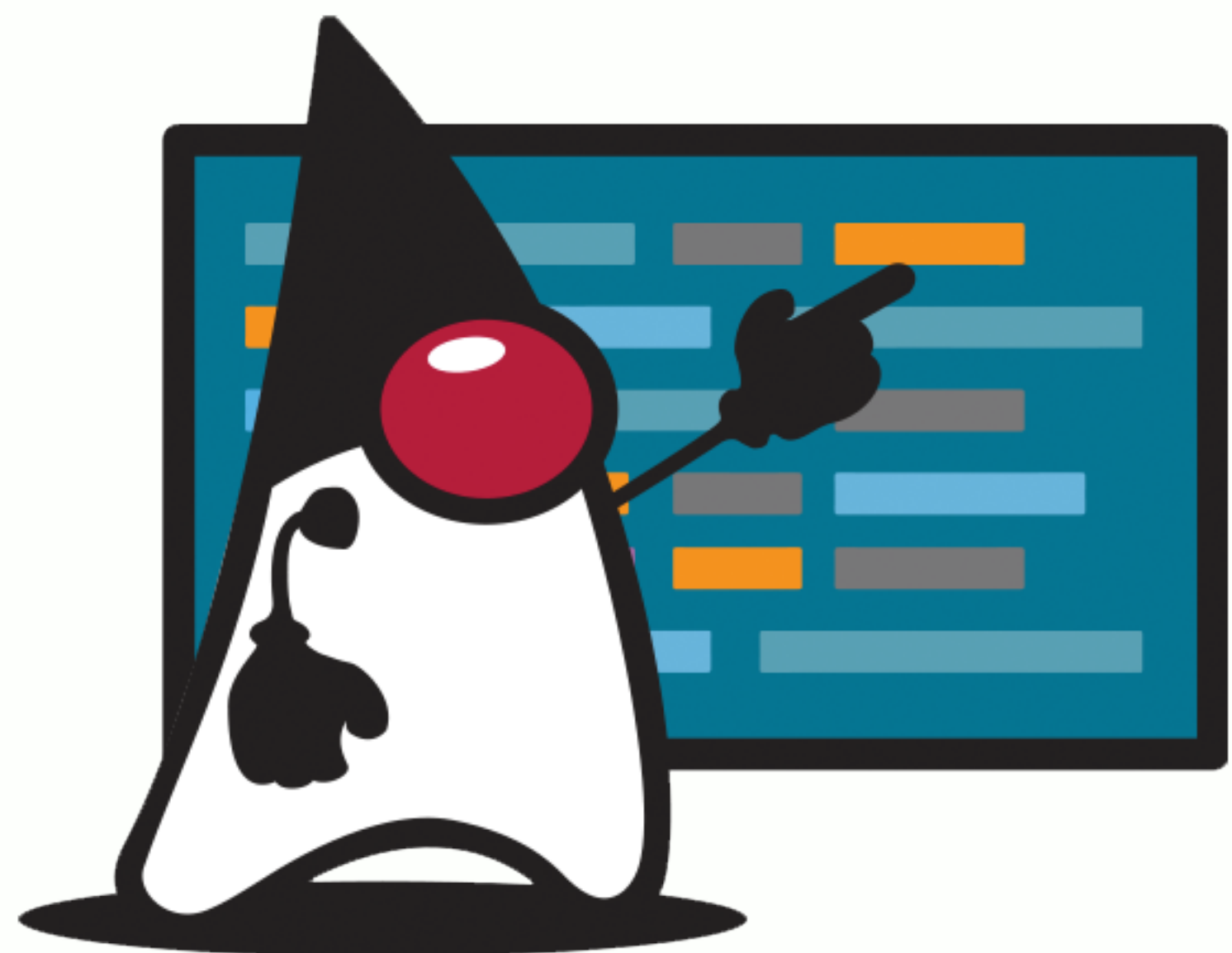
Visibilidade	+ public	# protected	~ default	- private
A partir da mesma classe	Visível	Visível	Visível	Visível
Qualquer classe no mesmo pacote	Visível	Visível	Visível	X
Qualquer classe filha no mesmo pacote	Visível	Visível	X	X
Qualquer classe filha em pacote diferente	Visível	Visível	X	X
Qualquer classe em pacote diferente	Visível	X	X	X



Por hoje é só



# Mão na massa



NOT VISIBLE