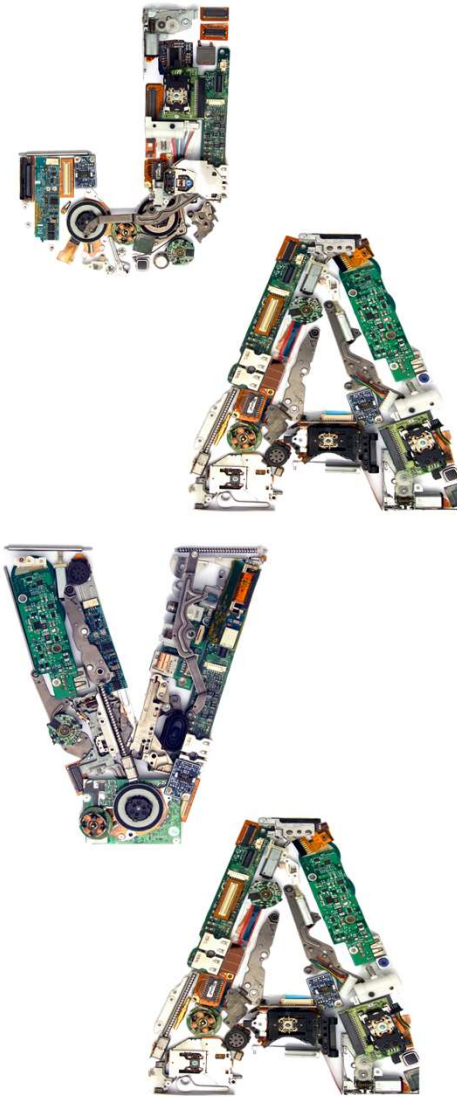


Programação Orientada a Objetos com Java e WEB

Implementação de Classes, Métodos e Objetos



Estrutura de uma classe no Java

```
//declaração de pacotes
public class NomeDaClasse { //public pode ser omitido (será tratado em outro momento)

    //declaração de variáveis - variáveis de instância (atributos)
    //declaração dos métodos

}
```

O conjunto de variáveis e métodos de uma classe é chamado de *membros da classe*.

Estrutura de uma classe no Java

Uma classe pode ter 3 tipos de **membros (o que está definido na classe)**:

- ❖ **Variáveis (atributos)**: também chamada de **variáveis de instância**.
- ❖ **Métodos**: definem o código executável da classe e o comportamento dos objetos.
- ❖ **Classes**: uma classe pode ter outras classes dentro dela (**não é muito comum**).

Estrutura de uma classe no Java

```
public class Pessoa { // é uma classe pública
```

```
String nome;  
int idade;
```

Atributos ou variáveis de instância
(variáveis declaradas na classe)

```
}
```

ou

```
class Pessoa { // não é uma classe pública
```

```
String nome;  
int idade;
```

```
}
```

Convenção de nomeação (padrão JavaBeans)

Classes

a primeira letra deve ser maiúscula e, se várias palavras forem escritas juntas, a primeira letra de cada palavra interna deve ser maiúscula (formato em inglês chamado de **camelCase**)

Para classes, os nomes devem normalmente ser substantivos

Exemplos: **Cachorro**, **Conta**, **Pessoa**, **Aluno**, **ContaCorrente**, etc...

Convenção de nomeação (padrão JavaBeans)

Métodos

a primeira letra deve ser minúscula, e depois a regra **camelCase** deve ser usada

Os nomes devem normalmente ser pares de verbo-substantivo

Exemplos: **obterSaldo()**,
imprimirNota(),
definirNomeDoCliente(),
etc...

Convenção de nomeação (padrão JavaBeans)

Variáveis

Segue a mesma regra definida
para os métodos

Exemplos: **larguraDoCampo**,
saldoDaConta, **nomeDoAluno**,
etc...

Convenção de nomeação (padrão JavaBeans)

Constantes

Sempre em letras maiúsculas

Exemplos: **ALTURA_MAXIMA**,
PI, **JUROS**, etc...

Convenção de nomeação de arquivo fonte

Só pode haver uma classe pública em cada arquivo-fonte

Se houver uma classe pública em um arquivo, o nome do arquivo deve ser o mesmo da classe pública

Um arquivo pode ter mais de uma classe não pública

Arquivos que não tenham classes públicas podem ter um nome que não seja o mesmo de nenhuma das classes do arquivo

Caso a classe utilizada no arquivo esteja em outro pacote, a declaração **import** deve vir no início do arquivo

Instanciação de classes

Instanciar uma classe significa **criar novos objetos** a partir da classe.

Todo objeto instanciado pertence a uma classe, ou seja, **o tipo de um objeto é sempre uma determinada classe.**

A sintaxe para a instanciação de um objeto é:

```
NomeDaClasse nomeDaVariável = new NomeDaClasse();
```

Instanciação de classes

Exemplo de instanciação: gerando um objeto de uma classe chamada **Pessoa** (ou instanciar a classe **Pessoa**).

```
Pessoa p = new Pessoa();
```

nome da classe
(tipo de dado)

variável
(variável de referência)

operador de
instanciação

nome da classe
(método construtor)

declaração de variável
(variável *p* do tipo *Pessoa*)

instanciação do objeto
(alocação de memória e construção do objeto)



Instanciação de classes

O processo de instanciação **pode ser** dividido em duas partes:

NomeDaClasse nomeDaVariável;

nomeDaVariável = new NomeDaClasse();



Exemplo: instanciação da classe **Pessoa**

Pessoa p; //apenas a declaração de uma variável (o objeto ainda não existe)

p = new Pessoa(); //neste momento o objeto passa a existir na memória

Atributos

Também conhecidos como *dados membros* ou *variáveis de instância*.

Os atributos são variáveis que devem ser declaradas dentro das classes (fora dos métodos). Exemplo:

```
public class Pessoa {  
    String nome;  
    int idade;  
}
```

atributos ou
variáveis de instância

Manipulação de objetos

Não se esqueça que uma classe somente define os atributos e métodos de um conjunto de objetos.

Para utilizar os atributos é necessário criar um ou mais objetos da classe.

Cada objeto contém uma “cópia” de todos os atributos da classe. Os atributos armazenam os dados (características) de cada um dos objetos.

variávelDeReferência.atributo;

operador ponto é utilizado para acessar
atributos e métodos

Manipulação de objetos

Exemplo de manipulação de objetos:

*declaração da
variável de referência*

Pessoa pessoa

Manipulação de objetos

Exemplo de manipulação de objetos:

*declaração da
variável de referência*

Pessoa pessoa

memória utilizada pelo programa

pessoa

Manipulação de objetos

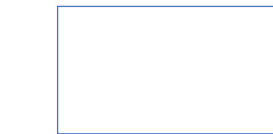
Exemplo de manipulação de objetos:

*declaração da
variável de referência* *operador de
instanciação*

Pessoa pessoa = new

memória utilizada pelo programa

pessoa



Manipulação de objetos

Exemplo de manipulação de objetos:

*declaração da
variável de referência* *operador de
instanciação*

Pessoa pessoa = new

memória utilizada pelo programa

pessoa

objeto alocado
na memória

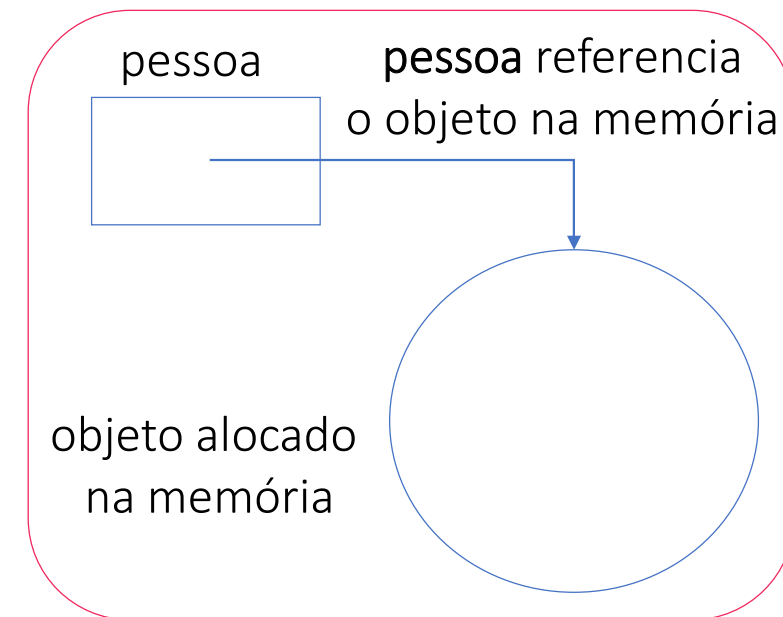
Manipulação de objetos

Exemplo de manipulação de objetos:

*declaração da
variável de referência* *operador de
instanciação*

Pessoa pessoa = new

memória utilizada pelo programa



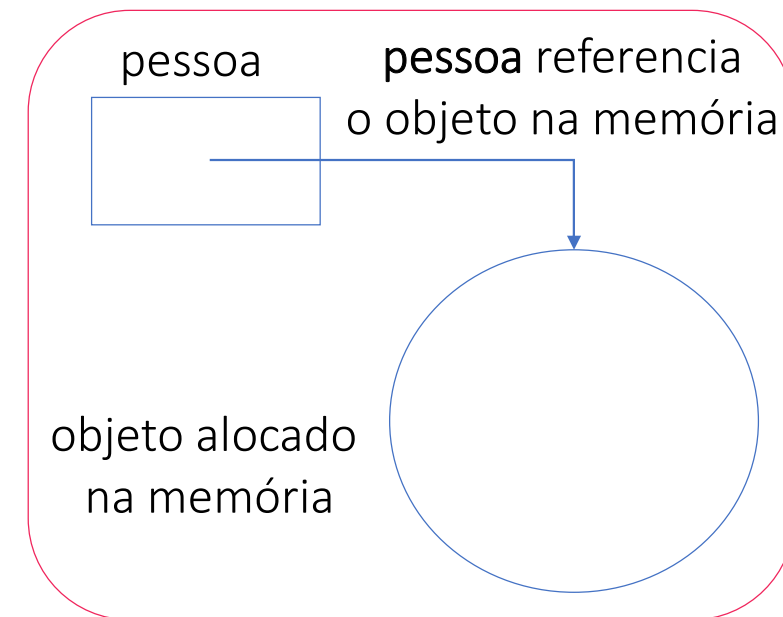
Manipulação de objetos

❑ Exemplo de manipulação de objetos:

*declaração da
variável de referência* *operador de
instanciação* *Método
construtor*

Pessoa pessoa = new Pessoa();

memória utilizada pelo programa



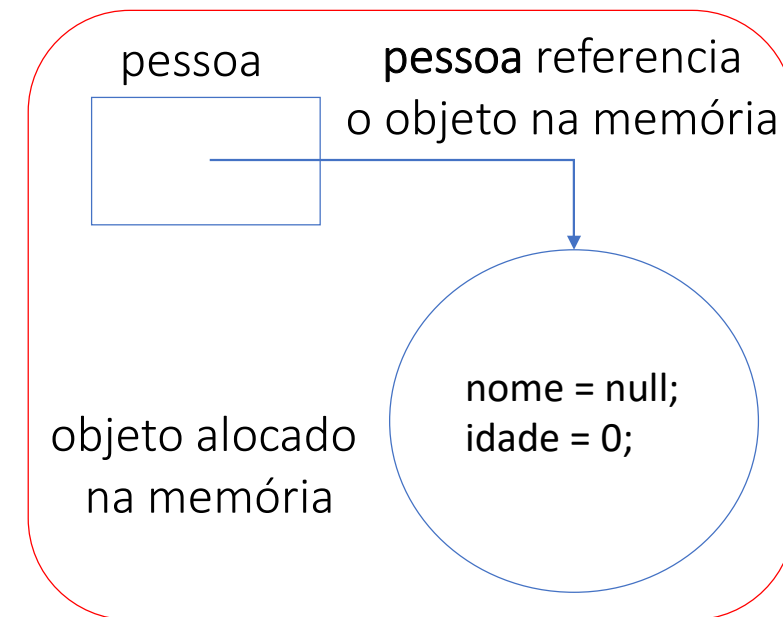
Manipulação de objetos

Exemplo de manipulação de objetos:

*declaração da
variável de referência* *operador de
instanciação* *Método
construtor*

Pessoa pessoa = new Pessoa();

memória utilizada pelo programa



Manipulação de objetos

Exemplo de manipulação de objetos:

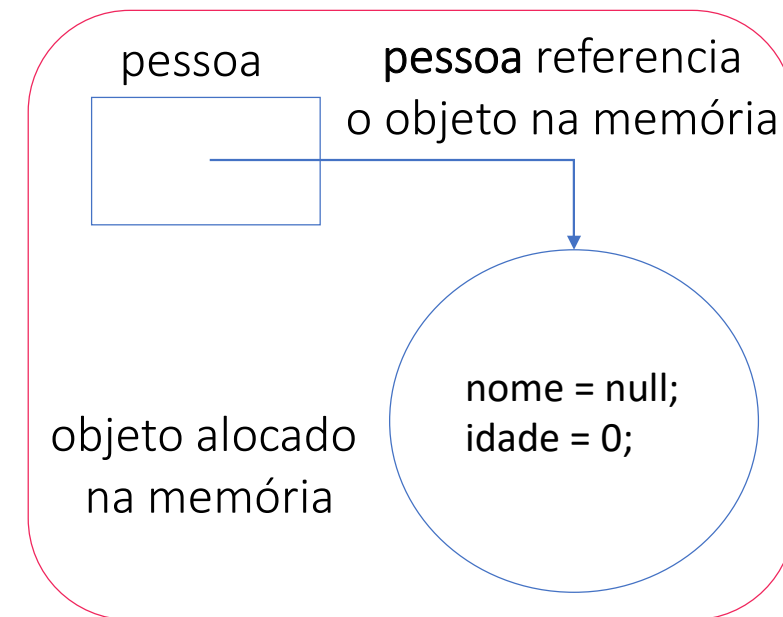
*declaração da
variável de referência* *operador de
instanciação* *Método
construtor*

Pessoa pessoa = new Pessoa();

//através da variável pessoa o objeto é acessado

pessoa.nome = "maria";

memória utilizada pelo programa



Manipulação de objetos

Exemplo de manipulação de objetos:

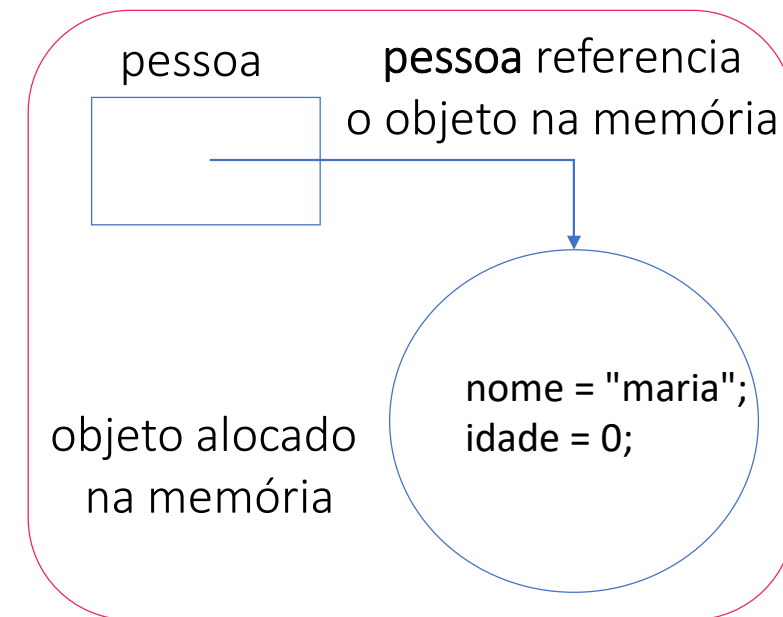
*declaração da
variável de referência* *operador de
instanciação* *Método
construtor*

```
Pessoa pessoa = new Pessoa();
```

//através da variável pessoa o objeto é acessado

```
pessoa.nome = "maria";
```

memória utilizada pelo programa



Manipulação de objetos

Exemplo de manipulação de objetos:

*declaração da
variável de referência* *operador de
instanciação* *Método
construtor*

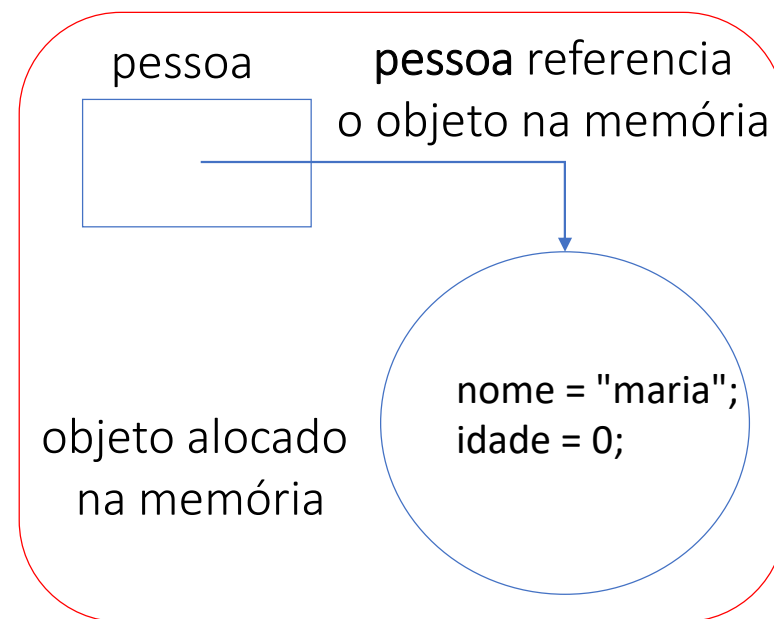
```
Pessoa pessoa = new Pessoa();
```

//através da variável pessoa o objeto é acessado

```
pessoa.nome = "maria";
```

```
pessoa.idade = 20;
```

memória utilizada pelo programa



Manipulação de objetos

Exemplo de manipulação de objetos:

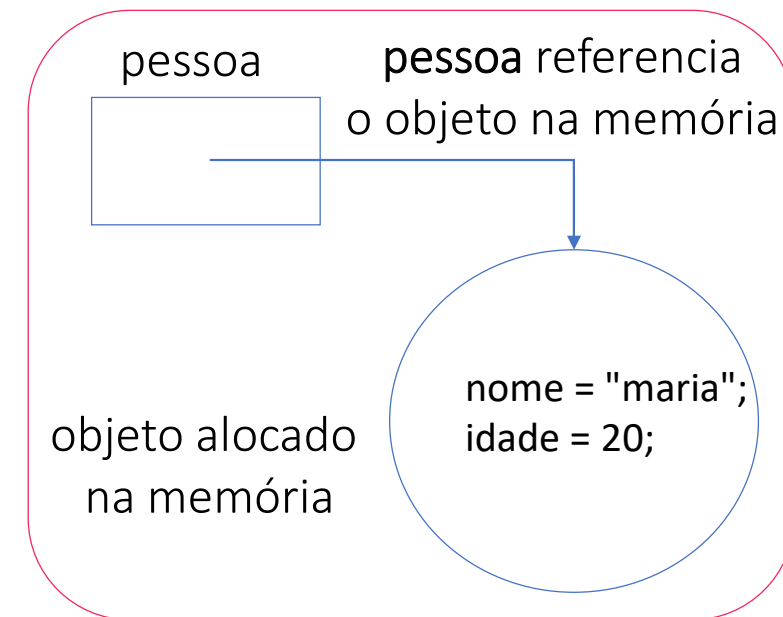
*declaração da
variável de referência* *operador de
instanciação* *Método
construtor*

```
Pessoa pessoa = new Pessoa();
```

//através da variável pessoa o objeto é acessado

```
pessoa.nome = "maria";  
pessoa.idade = 20;
```

memória utilizada pelo programa



Manipulação de objetos

Não se esqueça que os membros (atributos e métodos) de um objeto só podem ser acessados se o objeto for instanciado (criado).

`Pessoa pessoa;` declaração da variável de referência. Não há erros nessa linha.

`pessoa.nome = "Antonio";`
`pessoa.idade = 35;` Como o objeto não foi instanciado, a variável de referência não está referenciando nenhum objeto na memória → `NullPointerException`

`Pessoa pessoa = new Pessoa();` objeto instanciado e atribuído para a variável de referência

`pessoa.nome = "Antonio";`
`pessoa.idade = 35;` Sem erros porque a variável de referência está referenciado o objeto na memória

Pilha e heap

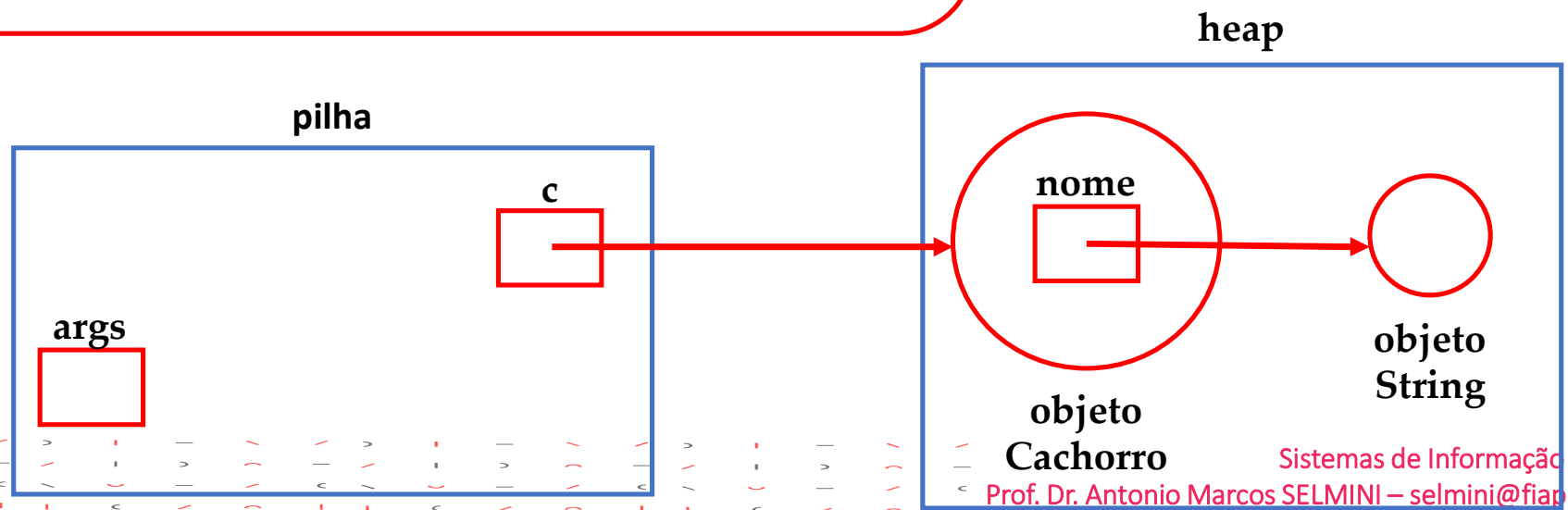
As várias partes (métodos, variáveis e objetos) dos programas Java residem em um dos dois seguintes lugares da memória: a *pilha* ou o *heap*.

As variáveis de instância e os objetos residem no *heap*. As variáveis locais e os métodos residem na *pilha*. Exemplo:

```
public class Dog {  
    String nome; //variável de instância  
    public static void main(String args[]) {  
        Dog d; //variável local (variável de referência)  
        d = new Dog();  
    }  
}
```

Pilha e heap

```
public class Cachorro {  
    String nome;  
  
    public static void main(String[] args) {  
        Cachorro c;  
        c = new Cachorro();  
    }  
}
```



Tipo primitivo x referência

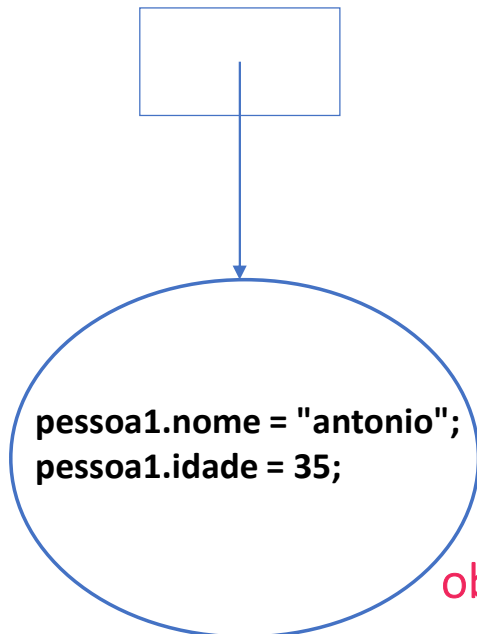
A linguagem de programação Java apresenta basicamente dois tipos de dados:
primitivo e **referência**.

- ❑ **Tipo primitivo**: tipos básicos da linguagem.
 - ❑ int, float, double, char, long, boolean, byte, short.
 - ❑ **São armazenados em memória, na pilha.**
- ❑ **Tipo de referência**: não armazenam tipos primitivos, mas sim referência (endereço) de um objeto na memória (heap).
 - ❑ String, int[], Pessoa, Aluno, Professor, etc...
 - ❑ **Objetos são armazenados no heap.**
 - ❑ A variável de **referência** é armazenada na **pilha**, mas os **objetos** são armazenados no **heap**.

Tipo primitivo x referência

```
Pessoa pessoa1 = new Pessoa();  
pessoa1.nome = "antonio";  
pessoa1.idade = 35;
```

pessoa1 (referência)

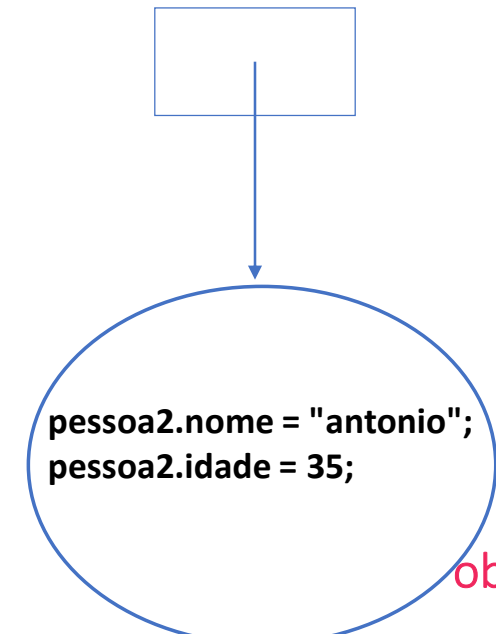


if(pessoa1 == pessoa2) {...}
true ou false?

É false!!
As variáveis *pessoa1* e *pessoa2* referenciam posições de memória diferentes!!!

```
Pessoa pessoa2 = new Pessoa();  
pessoa2.nome = "antonio";  
pessoa2.idade = 35;
```

pessoa2 (referência)



Tipo primitivo x referência

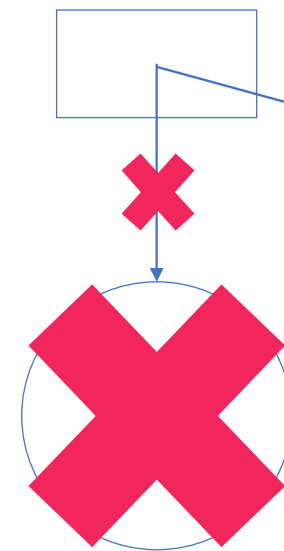
o que acontece nessa instrução?

pessoa1 = pessoa2;

a variável *pessoa1* armazena o mesmo conteúdo da variável *pessoa2*, ou seja, a variável *pessoa1* passa a referenciar o objeto referenciado por *pessoa2*

Pessoa pessoa1 = new Pessoa();

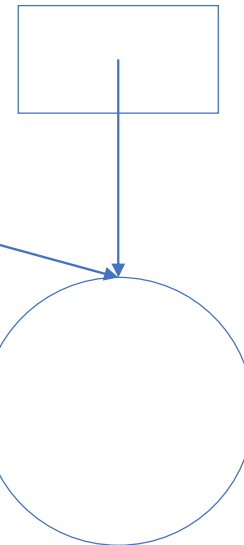
pessoa1 (referência)



objeto pessoa

Pessoa pessoa2 = new Pessoa();

pessoa2 (referência)



objeto pessoa

Métodos

Contêm o código que “entende” e **manipula o estado (conjunto de valores)** de um objeto.

Uma **declaração de método** consiste de duas partes: **o cabeçalho do método e o corpo do método.**

assinatura do método

```
visibilidade tipo nomeDoMétodo(parâmetros) {  
corpo do método  
}
```

visibilidade: normalmente (**private**, **public**, **protected** ou **default**) .

tipo: tipo retornado pelo método (por exemplo, **int**, **double**, *etc.*). Pode ser **void** para indicar que o método não tem retorno.

parâmetros: dados passados para o método.

Métodos

A **assinatura do método** representa o cabeçalho do método (**visibilidade + tipo + nome do método + parâmetros**).

O **corpo do método** contém o código necessário para manipular os estados do objeto. O **corpo do método representa a lógica da aplicação**.

Métodos são invocados como operações sobre objetos através de referência usando o operador ponto (.). Sintaxe:

variávelDeReferência.nomeDoMétodo(argumentos);

Métodos

Você pode enviar valores para um método.

Um método recebe parâmetros. Um chamador passa argumentos.

Quando um método é invocado, uma **lista de argumentos do tipo adequado** deve ser fornecido.

Métodos também têm um **tipo de retorno**, seja um **tipo primitivo** ou um **tipo de referência**. Se não tiver retorno deve ser declarado com *void*.

Métodos

assinatura do método

visibilidade do método

saída do método

nome do método

valores de entrada do método (parâmetros)

```
public int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

corpo do método

Métodos

Diferença entre *argumento* e *parâmetro*.

```
Teste teste = new Teste();  
teste.meuMetodo(25);
```

↑
argumento

parâmetro



```
public class Teste {  
    public void meuMetodo(int valor) {  
        int aux = 1;  
        while(aux <= valor) {  
            System.out.println(aux);  
            aux++;  
        }  
    }  
}
```

*argumento é o valor enviado e,
parâmetro, é o valor recebido.*

Métodos

o primeiro *argumento* será recebido pelo primeiro *parâmetro*, o segundo *argumento* será recebido pelo segundo *parâmetro* e assim por diante...

você pode enviar mais de um valor para o método.

```
Teste teste = new Teste();  
teste.meuMetodo(15, 45);
```

```
public class Teste {  
    public void meuMetodo(int valor1, int valor2) {  
        while(valor1 <= valor2) {  
            System.out.println(valor1);  
            valor1++;  
        }  
    }  
}
```

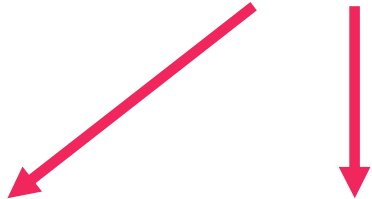
Métodos

variáveis podem ser passadas para um método, desde que o tipo da variável seja igual ao tipo do parâmetro

o valor do argumento *x* será recebido pelo parâmetro *valor1*, o valor do argumento *y* será recebido pelo parâmetro *valor2*.

```
int x = 5, y = 95;  
Teste teste = new Teste();  
teste.meuMetodo(x, y);
```

```
public class Teste {  
    public void meuMetodo(int valor1, int valor2) {  
        while(valor1 <= valor2) {  
            System.out.println(valor1);  
            valor1++;  
        }  
    }  
}
```



Métodos

A linguagem Java passa os *argumentos por valor*, ou seja, uma *cópia* do valor do argumento é passado para o parâmetro.

Ao passar a variável x como argumento, os bits que representam o número 5 serão copiados e armazenados no parâmetro *valor1*. Alterações na variável *valor1* não alteram a variável x .

```
int x = 5;  
Teste teste = new Teste();  
teste.meuMetodo(x);
```

```
public class Teste {  
    public void meuMetodo(int valor1) {  
        valor1 = valor1 * 2;  
    }  
}
```

Métodos

memória para o método **main()**

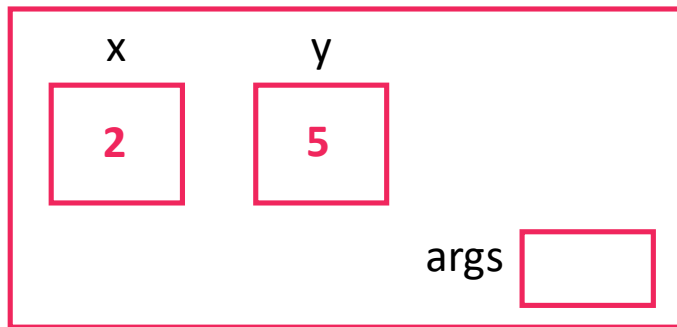


```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```


Métodos

memória para o método **main()**

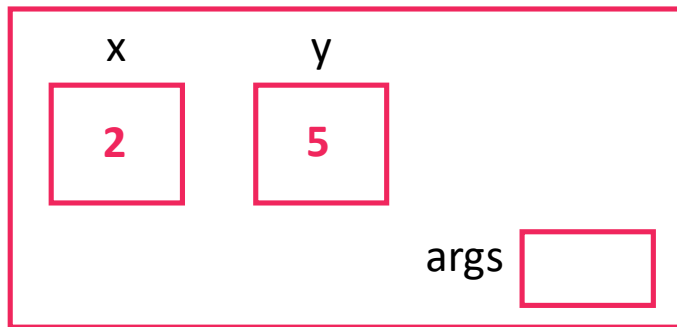


```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

memória para o método **main()**

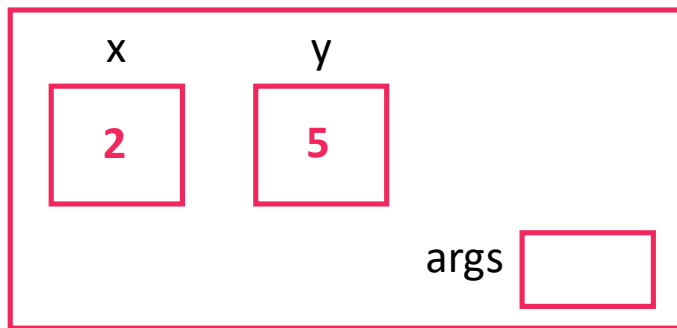


```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

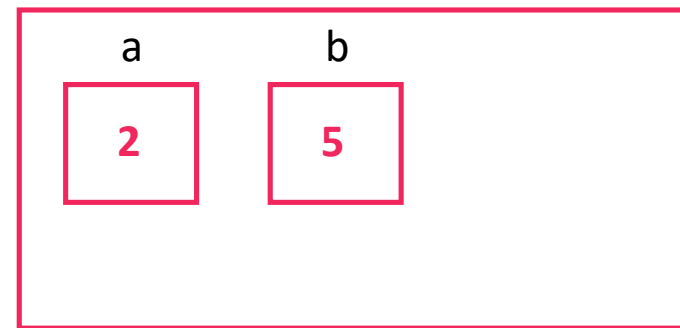
```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

memória para o método **main()**



memória para o método **soma()**

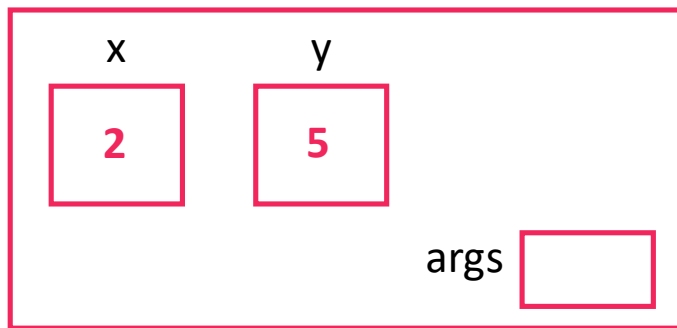


```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

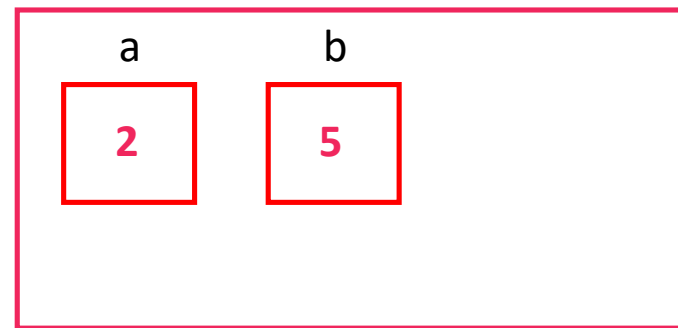
```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

memória para o método **main()**



memória para o método **soma()**

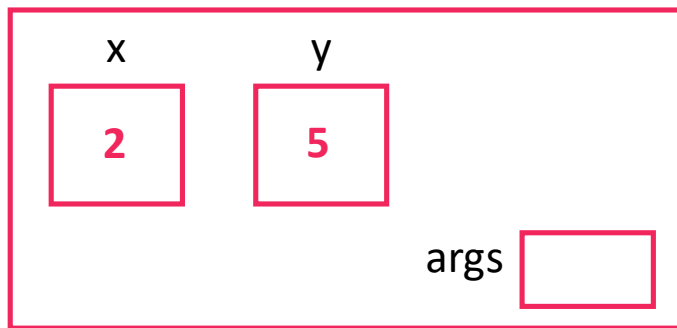


```
public static int soma(int a, int b) {  
    int r = a + b; 7  
    return r;  
}
```

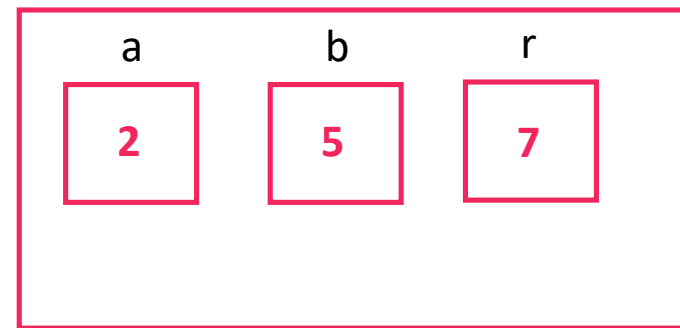
```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

memória para o método **main()**



memória para o método **soma()**

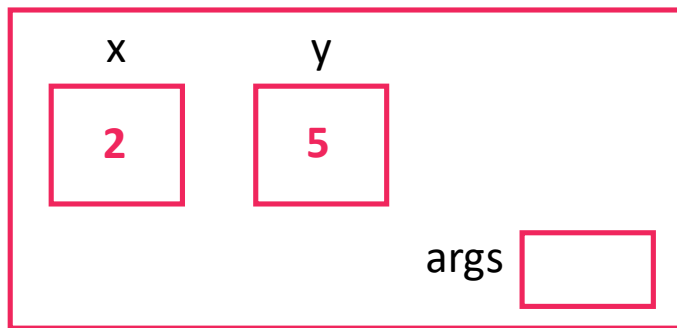


```
public static int soma(int a, int b) {  
    int r = a + b; 7  
    return r;  
}
```

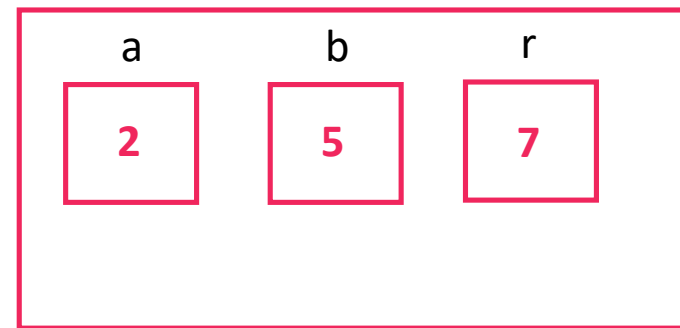
```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

memória para o método **main()**



memória para o método **soma()**

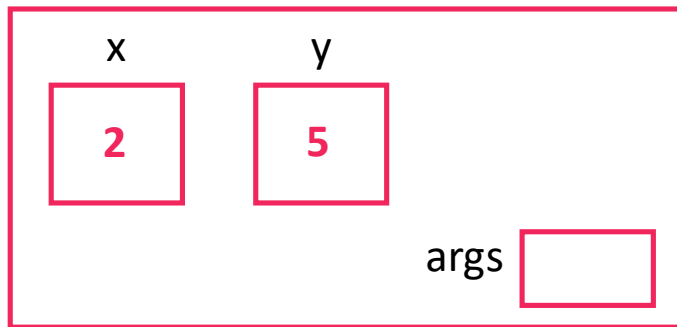


```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r; 7  
}
```

```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

memória para o método **main()**



memória para o método **soma()**

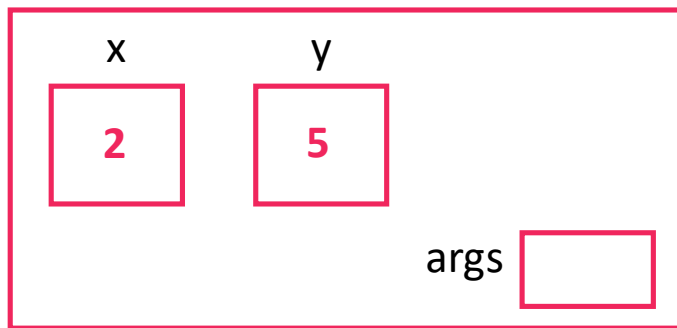


```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

memória para o método **main()**

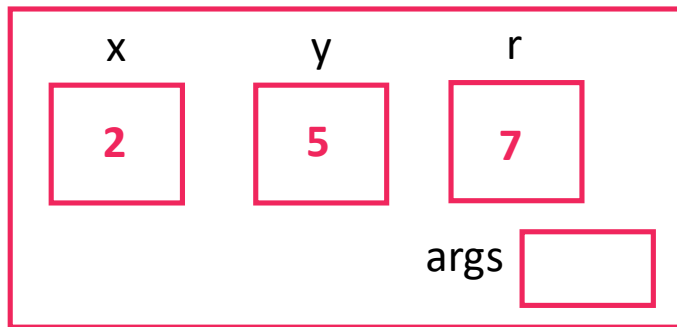


```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```


Métodos

memória para o método **main()**



```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);
```

Métodos



```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

```
public static int soma(int a, int b) {  
    int r = a + b;  
    return r;  
}
```

```
public static void main(String[] args) {  
    int x = 2, y = 5;  
    int r = soma(x, y);  
}
```

Métodos

métodos também podem receber e retornar referências de objetos!! Lembre-se: objetos são representados pelas variáveis de referência.

```
public class Aluno {  
    String nome;  
    int rm;  
}
```

```
Aluno aluno = new Aluno();  
meuMetodo(aluno);
```

uma cópia do argumento *aluno* é passada para o parâmetro *a*.

```
public void meuMetodo(Aluno a) {  
    System.out.println(a.nome);  
    System.out.println(a.rm);  
}
```

Métodos

métodos também podem receber e retornar referências de objetos!! Lembre-se: objetos são representados pelas variáveis de referência.

```
public class Aluno {  
    String nome;  
    int rm;  
}
```

```
Aluno aluno = new Aluno();  
Aluno aux = meuMetodo();
```

o valor de retorno armazenado na variável local *a* será armazenado na variável *aux*.

```
public Aluno meuMetodo() {  
    Aluno a = new Aluno();  
    return a;  
}
```

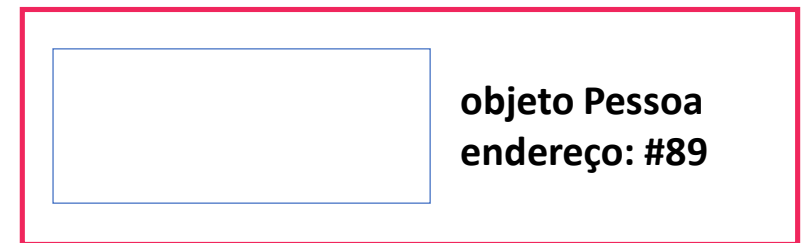
Métodos

```
public static void main(String[] args) {  
    Pessoa pessoa = new Pessoa(); #89  
    pessoa.nome = "Antonio";  
    pessoa.idade = 35;  
  
    meuMetodo(pessoa);  
}  
  
public static void meuMetodo(Pessoa p) {  
    //corpo do método  
}
```

memória para o método **main()**



heap



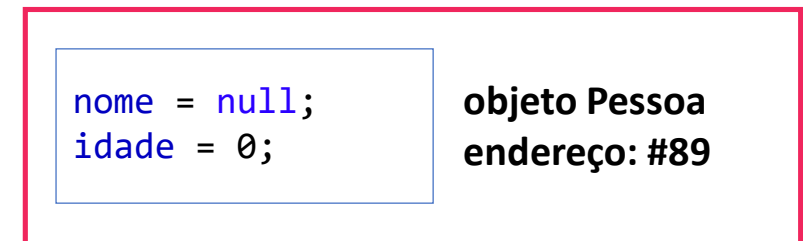
Métodos

```
public static void main(String[] args) {  
    Pessoa pessoa = new Pessoa(); #89  
    pessoa.nome = "Antonio";  
    pessoa.idade = 35;  
  
    meuMetodo(pessoa);  
}  
  
public static void meuMetodo(Pessoa p) {  
    //corpo do método  
}
```

memória para o método **main()**



heap



Métodos

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

```
    Pessoa pessoa = new Pessoa(); #89
```

```
    pessoa.nome = "Antonio";
```

```
    pessoa.idade = 35;
```

```
    meuMetodo(pessoa);
```

```
public static void meuMetodo(Pessoa p) {
```

```
    //corpo do método
```

memória para o método **main()**

pessoa



args



heap

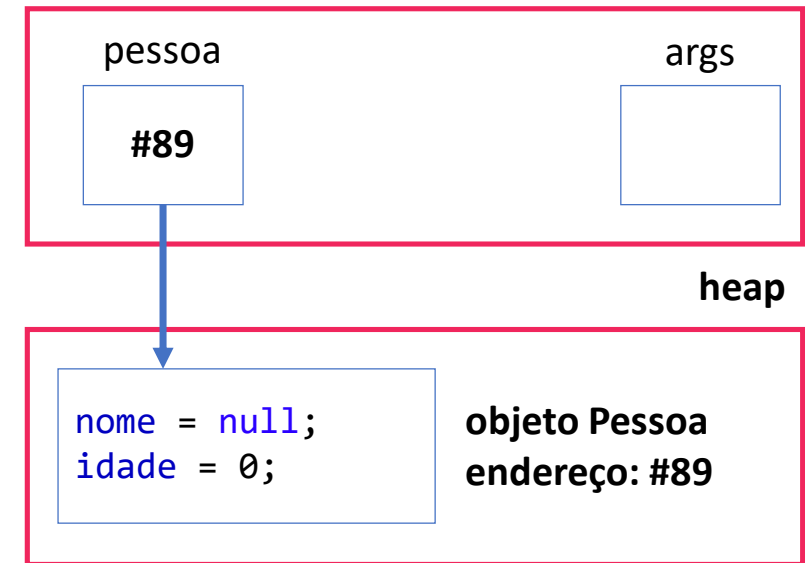
```
nome = null;  
idade = 0;
```

objeto Pessoa
endereço: #89

Métodos

```
public static void main(String[] args) {  
    Pessoa pessoa = new Pessoa(); #89  
    pessoa.nome = "Antonio";  
    pessoa.idade = 35;  
  
    meuMetodo(pessoa);  
}  
  
public static void meuMetodo(Pessoa p) {  
    //corpo do método  
}
```

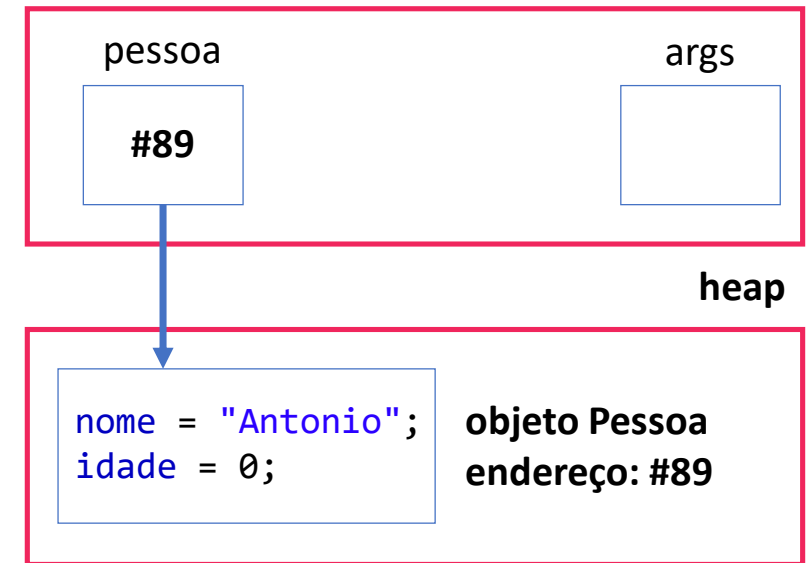
memória para o método **main()**



Métodos

```
public static void main(String[] args) {  
    Pessoa pessoa = new Pessoa(); #89  
    pessoa.nome = "Antonio";  
    pessoa.idade = 35;  
  
    meuMetodo(pessoa);  
}  
  
public static void meuMetodo(Pessoa p) {  
    //corpo do método  
}
```

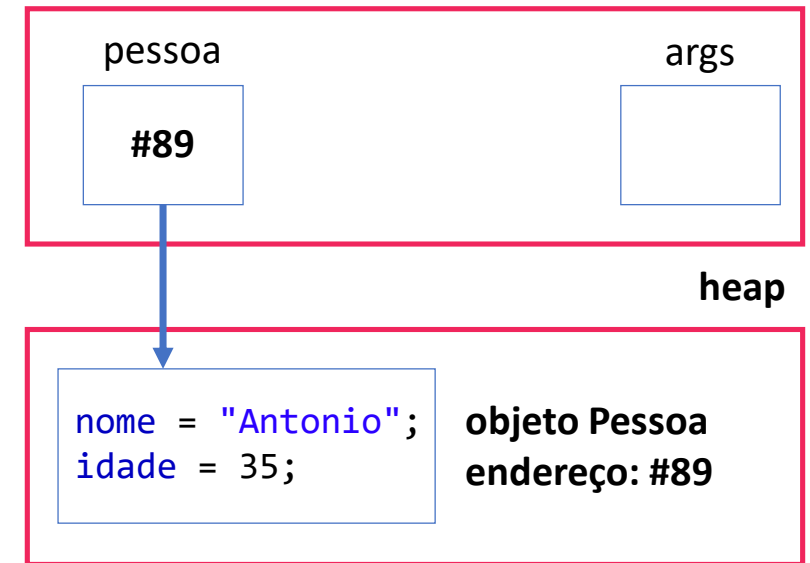
memória para o método **main()**



Métodos

```
public static void main(String[] args) {  
    Pessoa pessoa = new Pessoa(); #89  
    pessoa.nome = "Antonio";  
    pessoa.idade = 35;  
    meuMetodo(pessoa);  
}  
  
public static void meuMetodo(Pessoa p) {  
    //corpo do método  
}
```

memória para o método **main()**



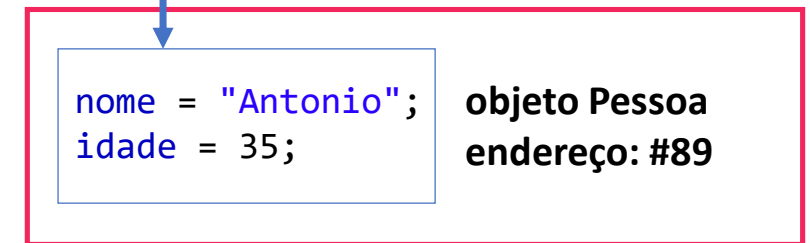
Métodos

```
public static void main(String[] args) {  
    Pessoa pessoa = new Pessoa(); #89  
    pessoa.nome = "Antonio";  
    pessoa.idade = 35;  
  
    meuMetodo(pessoa);  
}  
  
public static void meuMetodo(Pessoa p) {  
    //corpo do método  
}
```

memória para o método **main()**



heap



memória para o método **meuMetodo()**

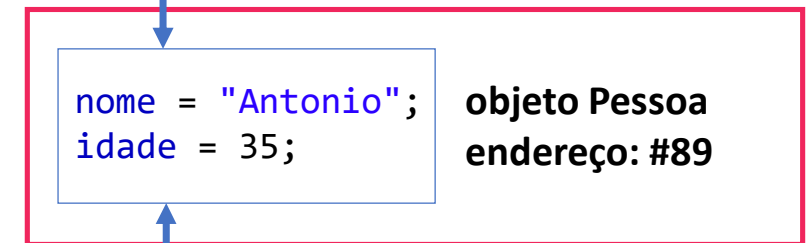
Métodos

```
public static void main(String[] args) {  
    Pessoa pessoa = new Pessoa(); #89  
    pessoa.nome = "Antonio";  
    pessoa.idade = 35;  
  
    meuMetodo(pessoa);  
}  
  
public static void meuMetodo(Pessoa p) {  
    //corpo do método
```

memória para o método **main()**



heap



memória para o método **meuMetodo()**

Métodos

```
public double meuMetodo(int x, double y) {  
    double z = 0.0;
```

```
    public void validar() {  
        int t = x;  
    }
```

```
    return z;
```

```
}
```

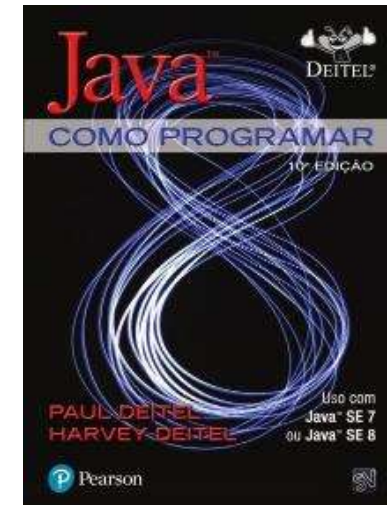
Observe atentamente os métodos ao lado. Tem alguma coisa de errada?

Sim!!

Um método não pode ser codificado dentro de outro método!!

Bibliografia

- ❑ DEITEL, H. M., DEITEL, P. J. JAVA como programar. 10ª edição. São Paulo: Prentice-Hall, 2010.
- ❑ SCHILDT, H. Java para Iniciantes – Crie, Compile e Execute Programas Java Rapidamente. 6ª Edição, Editora Bookman, Porto Alegre, RS, 2015.



Bibliografia

- ❑ KNUDSEN, J., NIEMEYER, P. Aprendendo Java. Rio de Janeiro: Editora Elsevier Campus, 2000.
- ❑ FLANAGAN, D. Java – o guia essencial. Porto Alegre: Editora Bookman, 2006.



Bibliografia

- ❑ ARNOLD, K., GOSLING, J., HOLMES, D., Java programming language. 4th Edition, Editora Addison-Wesley, 2005.
- ❑ JANDL JUNIOR, P. Introdução ao Java. São Paulo: Editora Berkeley, 2002.

