

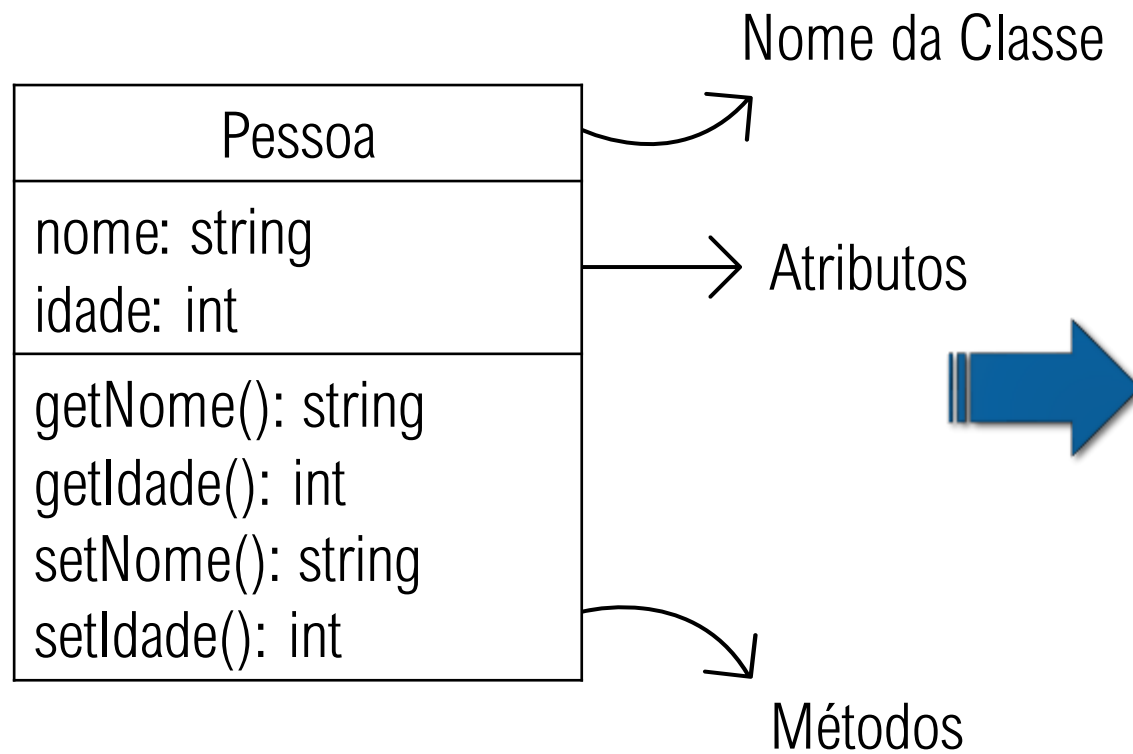
Programação Orientada a Objetos

- ✓ MODELOS DE MEMÓRIA
- ✓ MÉTODOS

amlucena@cruzeirosul.edu.br

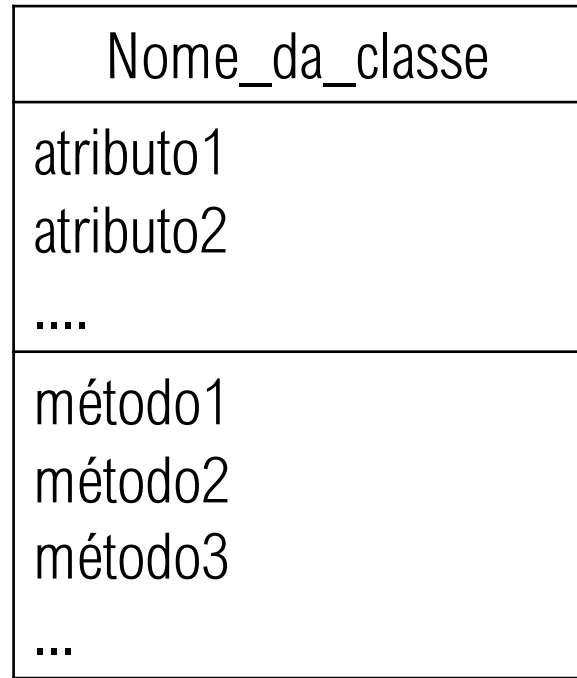
Aula anterior: Classes

Exemplo: A classe Pessoa deverá ter atributos e métodos comuns



Representação em Diagrama de Classe UML (*Unified Modelling Language*)

Aula anterior: Construindo classes em Java



Modelo UML de classe



```
public class <Nome_da_classe> {  
    <lista de atributos>  
    <lista de métodos>  
}
```



Um arquivo em Java precisa ser uma classe pública com o **mesmo nome do arquivo**

Resumindo a aula anterior

Classe ou Objeto?

```
Class Cachorro(nome, idade) {  
  this.nome = nome;  
  this.idade = idade;  
}  
  
objeto = Cachorro("caramelo", 7);
```

Classe é uma "receita", um template para criar vários objetos.

Um **objeto** é uma entidade da classe que tem seus próprios valores e estados.



Instância

Estrutura de uma classe Java executável

class é a palavra reservada que marca o início da declaração de uma classe.

Nome da Classe

Dever ter o mesmo nome do arquivo

```
public class [nome]
{
    public static void main (String args[ ])
    {
        ...
    }
}
```

Método main: onde inicia a execução

Instruções

Essa estrutura estará em todos os programa desenvolvidos em java

Exercício da aula passada...

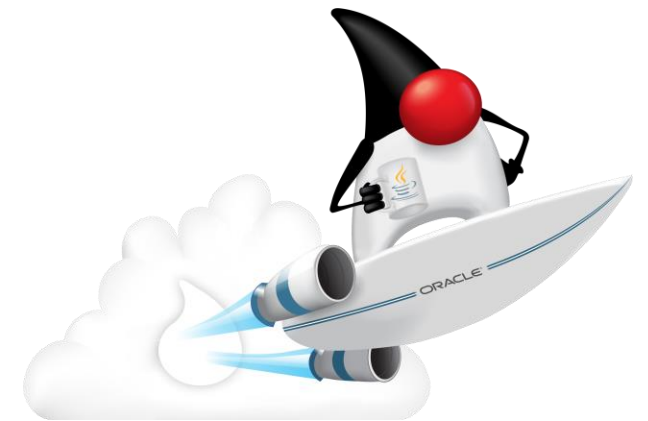
Considere os diagramas UML do seguinte item, observe que ele possui 2 construtores distintos. Construa o código da respectiva classe e crie 2 objetos usando os construtores criados e os preencha com dados fornecidos pelo usuário. Exiba os dados dos objetos.

Produto
marca: string fabricante: string cod_barras: string preco: float
Produto() Produto (m: string, f:string, c:string, p:float)

Modelo de memória

- ✓ O runtime do Java (JRE – ambiente em tempo de execução) separa a memória em dois espaços chamados de Stack e Heap.
- ✓ As variáveis de tipos primitivos são armazenadas na Stack
- ✓ Os objetos são armazenados na Heap e suas referências na Stack

<http://www.ibm.com/developerworks/br/library/j-codetoheap/>



Modelo de memória (tipos primitivos)

```
int var1; } ①  
var1 = 20;  
int var2; ②  
var2 = var1; ③  
var1 = 80; ④  
System.out.println("var1 is "+ var1+", var2  
is "+var2);
```

Pergunta: O que será impresso na tela?

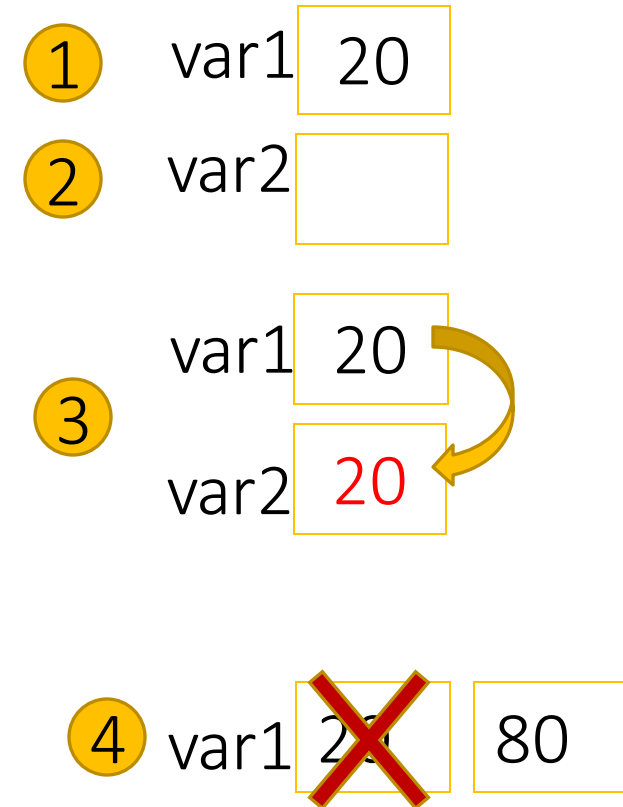
Opções:

1- var1 is 20, var2 is 80

2- var1 is 80, var2 is 20 

3- var1 is 80, var2 is 80

Memoria Stack



Modelo de memória (tipos primitivos)

```
int var1=150;  
int var2 = var1+1;  
var1 = var2+1;  
System.out.println("var1:"+var1+",var2:"+var2)
```

O que será impresso?

var1: 152 , var2 : 151



Tipos primitivos VS Objeto

(boolean, int,...) versus Arrays e Classes

```
int var1 = 20;
```

```
Pessoa p1;
```

```
p1= new Pessoa("Fulano1", 33);
```

```
Pessoa p2 = new Pessoa("Fulano",35);
```

```
p2.nome = "Fulano2";
```

```
Pessoa p3;
```

```
p3 = p2;
```

Quando é objeto não faz
cópia!!!
p2 e p3 são o mesmo
objeto!!!

Stack

var1 20

p1 @34

p2 @38

p2 @38

p3 @38

Endereço de memória

@34

@38

@38

Memória Heap

nome Fulano1
idade 33

nome Fulano
idade 35

nome Fulano2
idade 35

Escopo (variáveis e objetos)

- ✓ Escopo é a visibilidade de uma variável ou objeto em nosso código.
- ✓ Existem três escopos: da classe, do método e de bloco.

```
public class Pessoa {  
    //lista de atributos  
    String nome;  
    int idade;  
    double renda;  
  
    //Construtores  
    Pessoa() {}  
  
    Pessoa(String n, int i, double r){  
        nome=n;  
        idade=i;  
        renda=r;  
    }  
  
    //lista de métodos  
}
```

A variável é
válida
dentro de
seu escopo

Global
x
Local



Escopo da classe



Escopo do
método

```
public class UsaPessoa {  
    public static void main(String[] args) {  
        // TODO code application logic here  
        Pessoa p1 = new Pessoa();  
  
        p1.nome="Beltrano";  
        p1.idade=30;  
        p1.renda=1000.00;  
  
        if (p1.renda <= 1000)  
        {  
            double aumento = p1.renda * 0.5;  
            p1.renda = p1.renda+aumento;  
        }  
  
        System.out.println("\nNome: " + p1.nome);  
        System.out.println("Idade: " + p1.idade);  
        System.out.println("Renda: " + p1.renda);  
    }  
}
```



Escopo de bloco

Escopo (variáveis e objetos)

```
public class UsaPessoa {  
    public static void main(String[] args) {  
        // TODO code application logic here  
        int idade = 40;  
        Pessoa p = new Pessoa("Fulano", idade, 10000.00);  
        Pessoa p1 = new Pessoa();  
  
        p1.nome="Beltrano";  
        p1.idade=30;  
        p1.renda=1250.00;  
    }  
}
```

Stack

idade	40
p	
p1	

Escopo do main

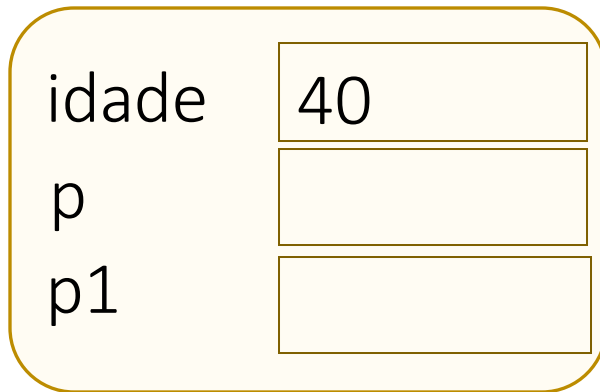
Heap

nome	
idade	
renda	

nome	
idade	
renda	

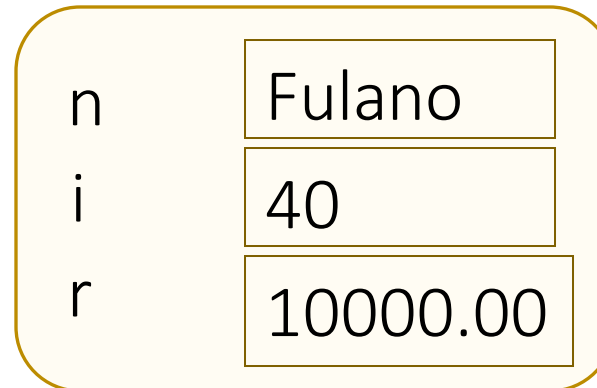
Escopo (variáveis e objetos)

```
public class Pessoa {  
    //lista de atributos  
    String nome;  
    int idade;  
    double renda;  
  
    //Construtores  
    Pessoa(){}  
    Pessoa(String n, int i, double r){  
        nome=n;  
        idade=i;  
        renda=r;  
    }  
}
```



Escopo do main

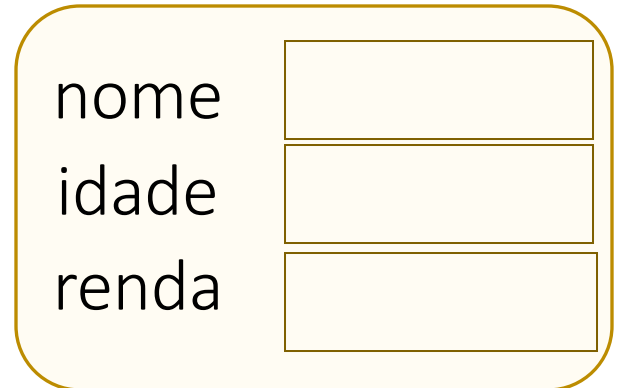
```
public class UsaPessoa {  
    public static void main(String[] args) {  
        // TODO code application logic here  
        int idade = 40;  
        Pessoa p = new Pessoa("Fulano", idade, 10000.00);  
        Pessoa p1 = new Pessoa();  
  
        p1.nome="Beltrano";  
        p1.idade=30;  
        p1.renda=1250.00;  
    }  
}
```



Escopo do construtor

Heap

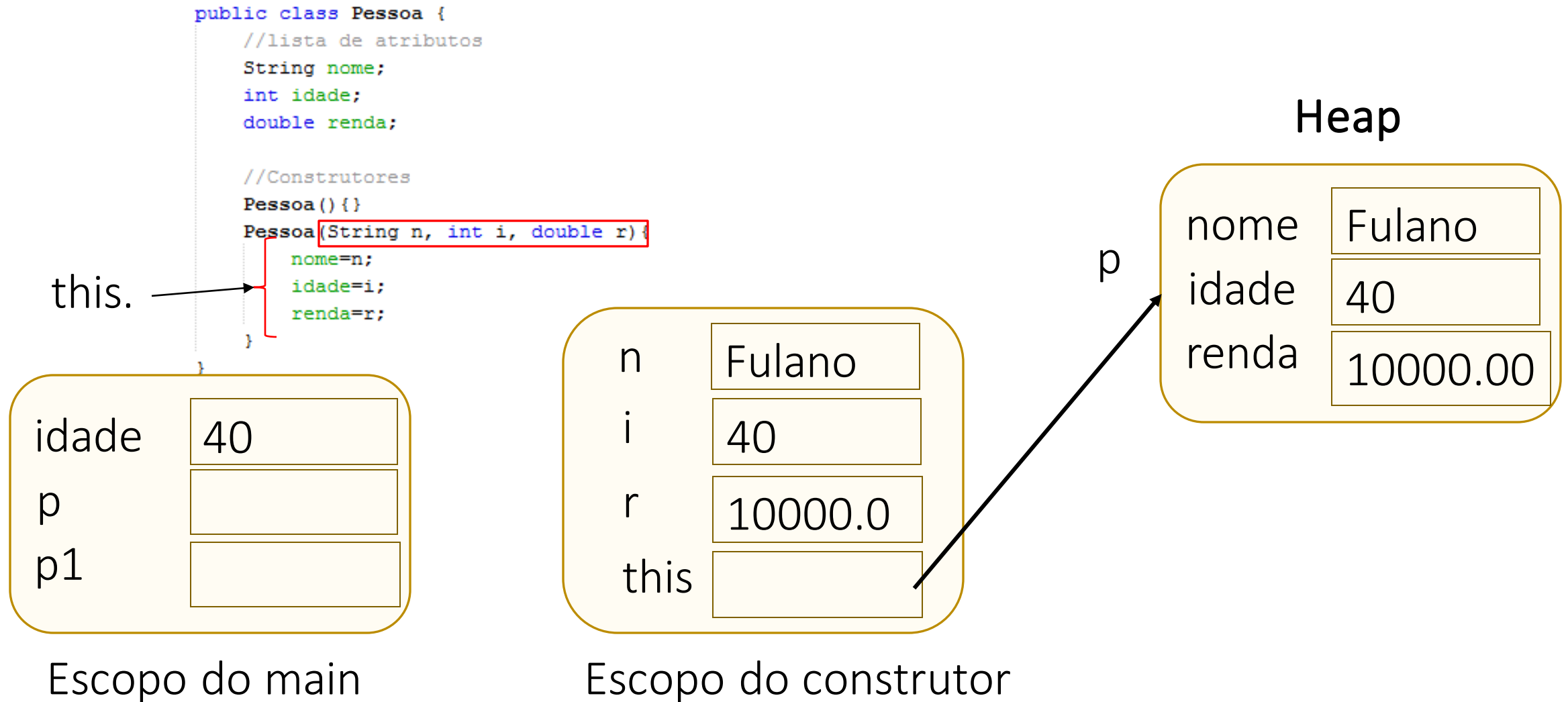
p



p1



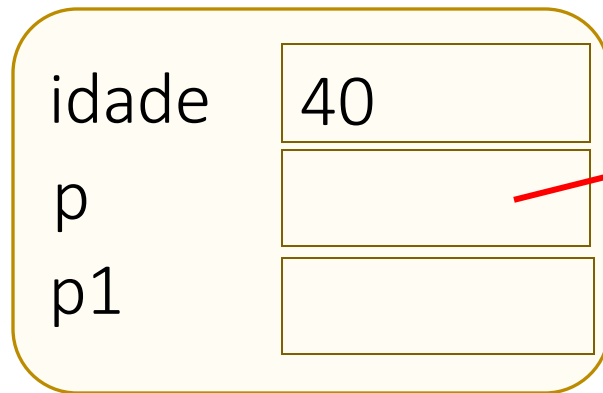
Escopo (variáveis e objetos)



Escopo (variáveis e objetos)

Após finalizar o uso do construtor, seu escopo é apagado e a referência é feita no objeto criado.

```
public class UsaPessoa {  
    public static void main(String[] args) {  
        // TODO code application logic here  
        int idade = 40;  
        Pessoa p = new Pessoa("Fulano", idade, 10000.00);  
        Pessoa p1 = new Pessoa();  
  
        p1.nome="Beltrano";  
        p1.idade=30;  
        p1.renda=1250.00;  
    }  
}
```

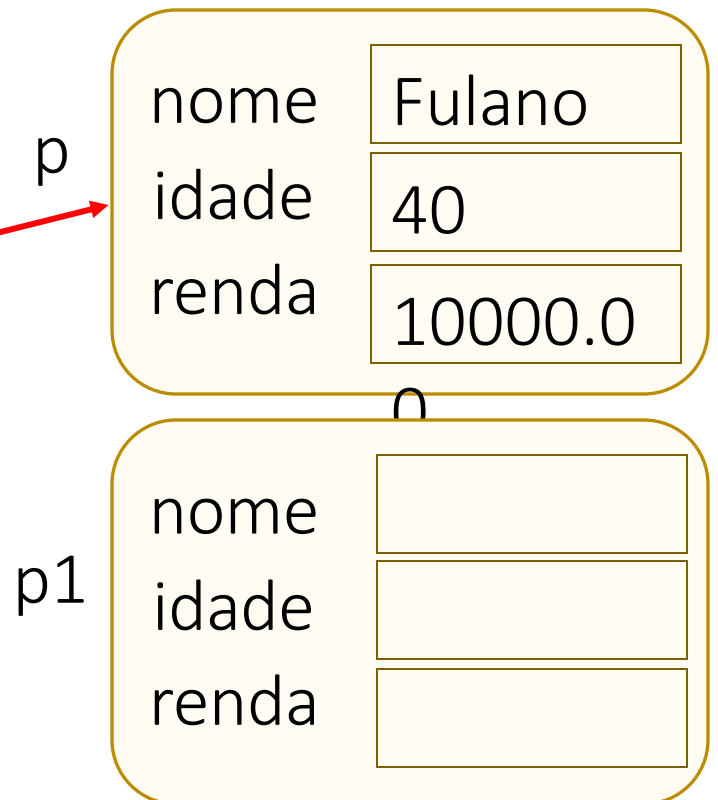


Escopo do main



Escopo do construtor

Heap



Escopo (variáveis e objetos)



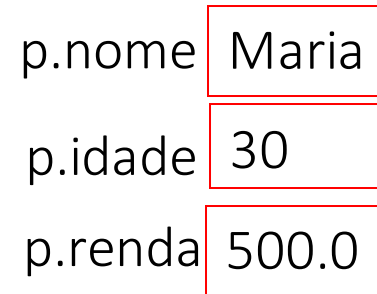
Desafio 1

```
public class Pessoa {  
    String nome;  
    int idade;  
    double renda;  
  
    Pessoa(String n, int i, double r){  
        nome=n;  
        idade=i;  
        renda=r;  
    }  
}
```

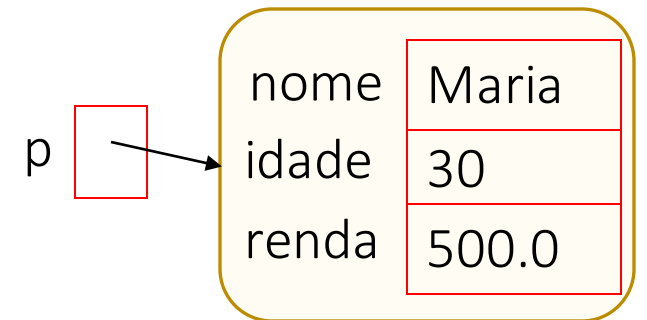
```
public class UsaPessoa {  
    public static void main(String[] args) {  
        Pessoa p = new Pessoa("Maria", 30, 500.0);  
    }  
}
```

Selecione a opção com o modelo de memória correto após a execução do programa principal.

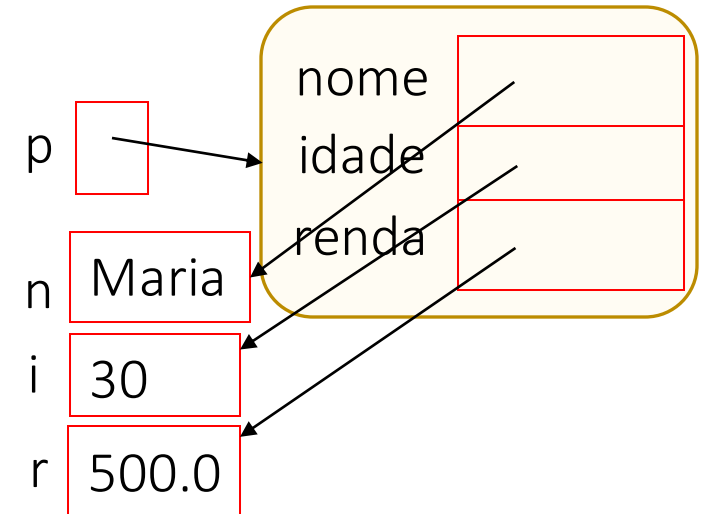
(A)



(B)

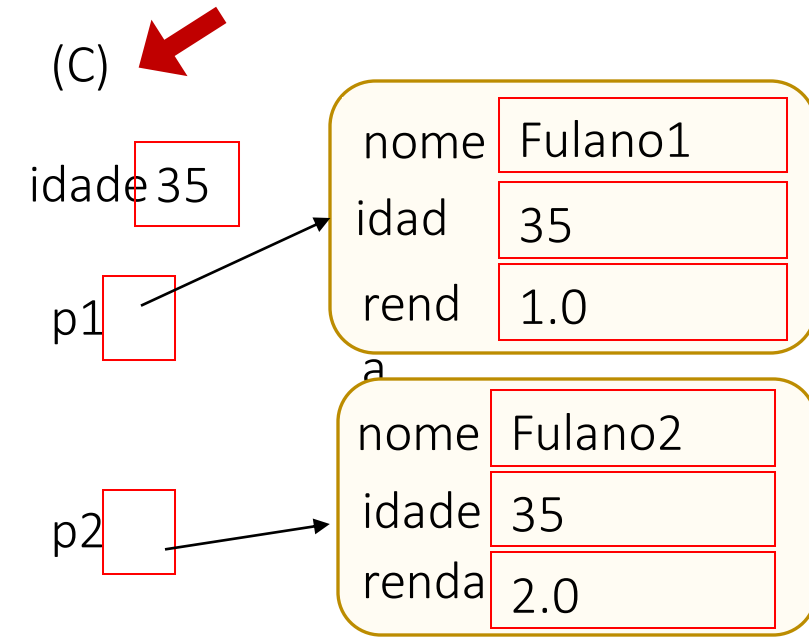
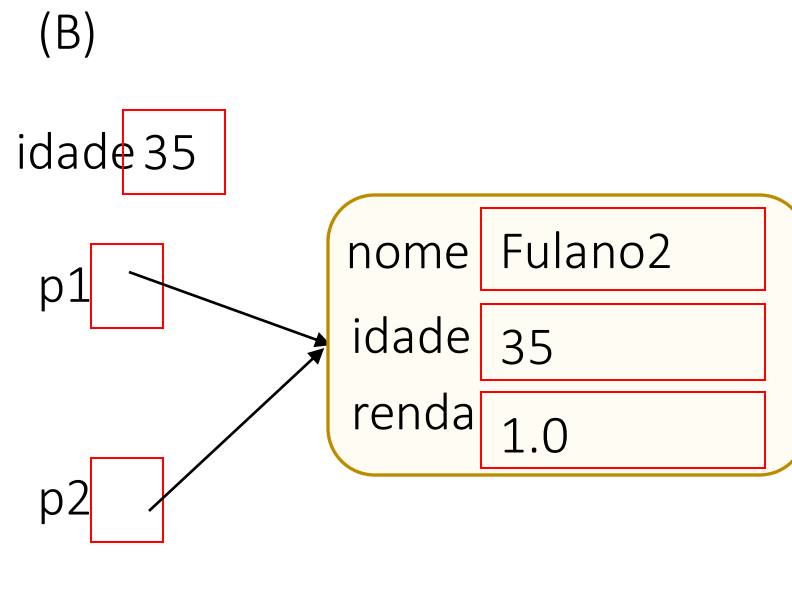
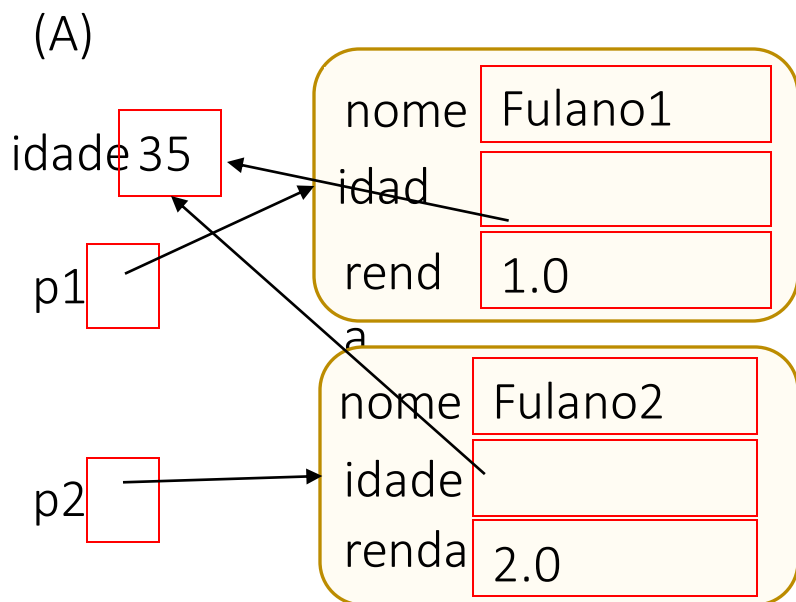


(C)



Desafio 2


```
public class UsaPessoa {  
    public static void main(String[] args) {  
        int idade = 35;  
        Pessoa p1 = new Pessoa("Fulano1", idade, 1.0);  
        Pessoa p2 = new Pessoa("Fulano2", p1.idade, 2.0);  
    }  
}
```



Desafio 3

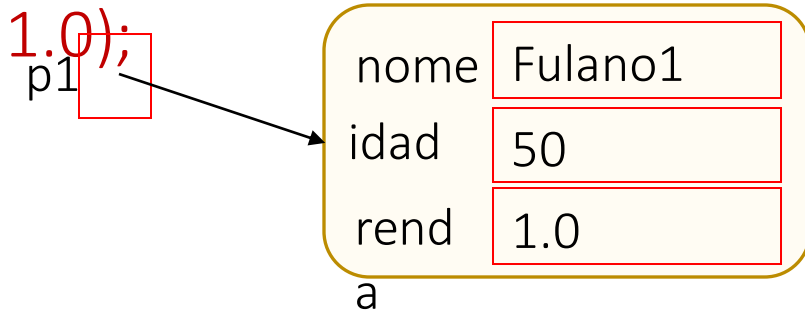
```
public class UsaPessoa {  
    public static void main(String[] args) {  
        Pessoa p1 = new Pessoa("Fulano1", 50, 1.0);  
        Pessoa p2 = new Pessoa("Fulano2", 10, 2.0);  
        p1 = p2;  
        p1.idade = 20;  
        System.out.print("Nome: "+p2.nome+", idade: "+p2.idade);  
    }  
}
```

O que será impresso no final da execução do código acima?

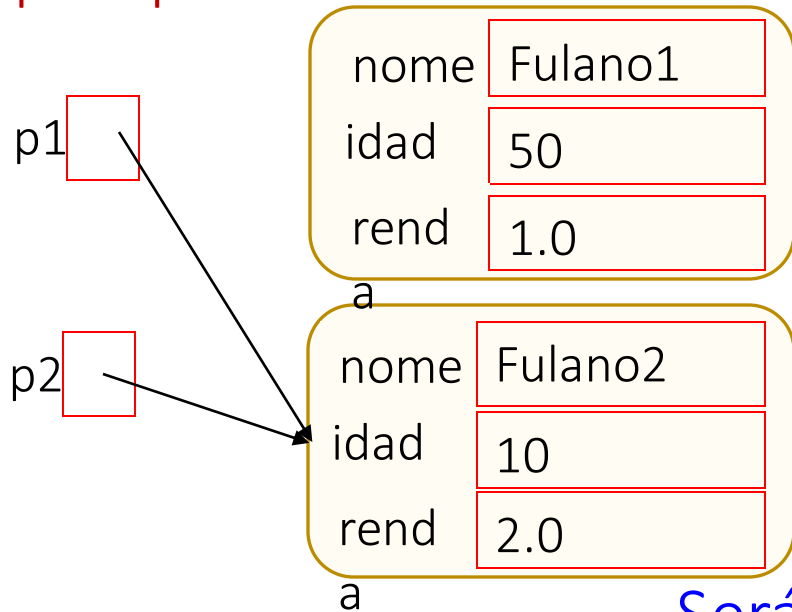
- (A) Nome: Fulano2, idade: 10
-  (B) Nome: Fulano2, idade: 20
- (C) Nome: Fulano1, idade: 20
- (D) Nome: Fulano1, idade: 50

Desafio 3 (resolução)

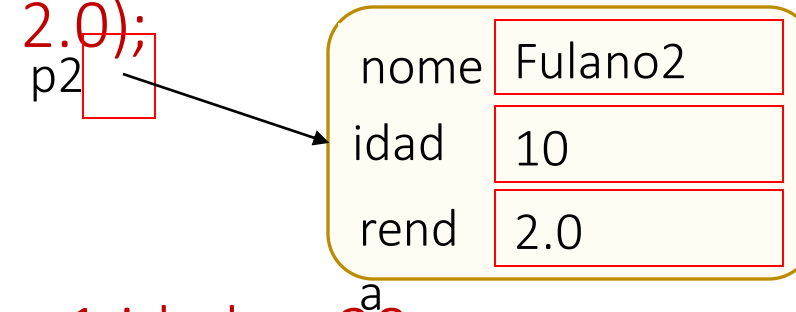
Pessoa p1 = new Pessoa("Fulano1", 50, 1.0);



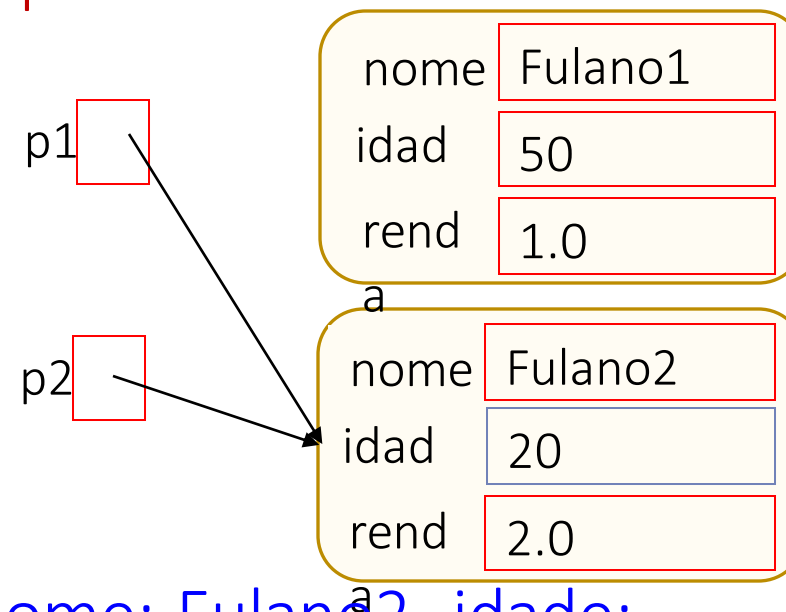
p1 = p2



Pessoa p2 = new Pessoa("Fulano2", 10, 2.0);



p1.idade = 20



Será impresso "Nome: Fulano2, idade: 20"

Desafio 4

```
public class UsaPessoa {  
    public static void main(String[] args) {  
        int idade = 10;  
        Pessoa p1 = new Pessoa("Fulano1", idade, 1.0);  
        idade = 30;  
        System.out.print("Idade: "+p1.idade);  
    }  
}
```

Qual o valor que será impresso para idade?



- (A) 10
- (B) 30
- (C) Erro de compilação

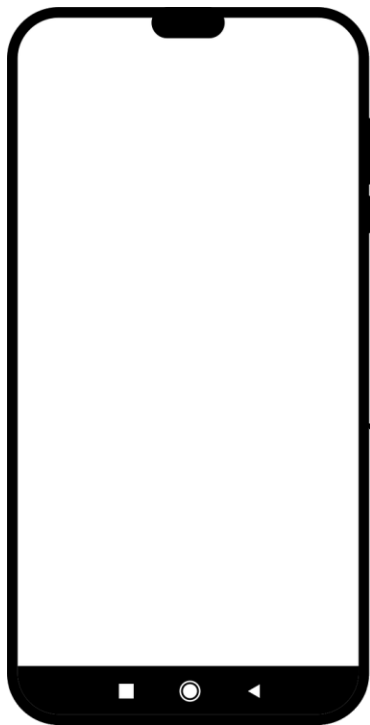
Métodos



Métodos

Classe =

Atributos (Propriedades)
+
Métodos (Comportamento)



→ Propriedades:

- marca
- modelo
- Sistema operacional

→ Comportamentos:

- liga
- desliga
- envia mensagem

Celular
marca: string modelo: string so: string
ligar(): void desligar(): void enviarMensagem(msg: string): void

Métodos

São conjuntos de instruções que cumprem uma tarefa específica e devolvem ou não um valor ao programa que o chamou.

```
tipo de retorno nome(parâmetros)
{
    instruções;
    <return resposta>;
}
```



Métodos

- ✓ Os métodos também possuem padrão de nomenclatura. Eles sempre começam com a primeira inicial minúscula e as outras iniciais maiúsculas.
 - `pegarInformacao()`
 - `executarComandoInicial()`
- ✓ Os métodos geralmente são ações que podem ser efetuadas entre os atributos do objeto.
- ✓ Métodos que não retornam valores, apenas executam ações, são do tipo *void*.



Métodos

- ✓ Os métodos podem ou não assumir tipos de dados, caso não assumam, são chamados de procedimentos, pois executam um conjunto de instruções sem devolverem valor algum a quem os chamou.
- ✓ Um método sem tipo recebe em sua definição a palavra-chave void no lugar do tipo.

```
//Procedimento sem parâmetro  
void imprime() {  
    System.out.println("Estamos imprimindo!");  
}
```



Métodos

- ✓ Quando os métodos assumem algum tipo, eles são chamados de funções e precisam do comando return para devolver o valor resultante da execução de suas instruções internas.

```
//Função sem parâmetro  
int numero() {  
    int num = Math.random();  
    return num;  
}
```



Métodos

- ✔ Os métodos podem receber dados para serem utilizados internamente, os quais são chamados de parâmetros ou de argumentos.
- ✔ Quando os parâmetros são passados para os métodos, é criada uma cópia dos valores.

```
//Procedimento com parâmetro  
void dobro (int n) {  
    int resp;  
    resp = n * 2;  
    System.out.println(resp);  
}
```

```
//Função com parâmetro  
int soma(int n, int m) {  
    int s;  
    s = n + m;  
    return s;  
}
```

Métodos

- ✓ Podemos passar vários parâmetros para os métodos, inclusive de tipos diferentes.

```
//Procedimento com parâmetro  
void linha (int n, char ch) {  
    System.out.println(ch);  
    System.out.println(n);  
}
```



Métodos

```
//definição das funções
float quad (float z) {
    return (z * z);
}
float soma (float m, float n) {
    return (m + n);
}
float somaquad (float m, float n)
{
    return soma(quad(m), quad(n));
}
```

Uso do método *quad*
como parâmetro do
soma



Métodos – Exemplo 1

ContaPoupanca
agencia: string numero: string saldo: float taxa: float
ContaPoupanca() ContaPoupanca(a: string, n: string, s: float, t: float) imprimeDados(): void depositar(valor: float): void calculaRendimento(): float



Métodos – Exemplo 1

```
public class ContaPoupanca {  
    //atributos  
    String agencia;  
    String numero;  
    float saldo;  
    float taxa;  
  
    public ContaPoupanca() {}  
    public ContaPoupanca(String agencia, String numero, float saldo, float taxa) {  
        this.agencia = agencia;  
        this.numero = numero;  
        this.saldo = saldo;  
        this.taxa = taxa;  
    }  
}
```



Métodos – Exemplo 1

```
//Métodos
void imprimeDados() {
    JOptionPane.showMessageDialog(null, "Dados da Conta Corrente: "+
        "\nAgência: "+agencia +
        "\nNúmero: "+ numero +
        "\nSaldo: "+ saldo +
        "\nTaxa: "+ taxa);
}

void depositar(float valor){
    saldo = saldo + valor;
}

float calculaRendimento() {
    float rendimento;
    rendimento = saldo * taxa/100;
    return rendimento;
}
```



Métodos – Exemplo 1

```
public class TestePoupanca {  
  
    public static void main(String[] args) {  
        ContaPoupanca cp;  
  
        float valorDep, rend, saldo, taxa;  
        String agencia, num;  
  
        agencia = JOptionPane.showInputDialog("Digite o número da agência");  
        num = JOptionPane.showInputDialog("Digite o número da Conta Poupança");  
        saldo = Float.parseFloat(JOptionPane.showInputDialog("Digite o saldo"));  
        taxa = Float.parseFloat(JOptionPane.showInputDialog("Digite a taxa de juros"));  
  
        cp = new ContaPoupanca(agencia, num, saldo, taxa);  
  
        //Utilização dos métodos para o objeto do tipo Conta Poupança  
        cp.imprimeDados();  
        rend=cp.calculaRendimento();  
        JOptionPane.showMessageDialog(null,"O rendimento é: " +rend);  
        valorDep=Float.parseFloat(  
            JOptionPane.showInputDialog("Digite o valor a ser depositado"));  
        cp.depositar(valorDep);  
        cp.imprimeDados();  
    }  
}
```



Métodos – Exemplo 2

Produto
marca: string valor: float
Produto() Produto(m: string, v: float) imprimeDados(): void calculaImposto(p: float): float



Métodos – Exemplo 2

```
public class Produto {  
    //Atributos  
    String marca;  
    float valor;  
  
    //Construtores  
    public Produto() {  
    }  
  
    public Produto(String marca, float valor) {  
        this.marca = marca;  
        this.valor = valor;  
    }  
  
    void imprimeDados() {  
        JOptionPane.showMessageDialog(null, "Dados do Produto: " +  
            "\nMarca: " + marca +  
            "\nValor: " + valor);  
    }  
  
    float calculaImposto(float p) {  
        return valor*p/100;  
    }  
}
```



Métodos – Exemplo 2

```
public class TesteProduto {  
  
    public static void main(String[] args) {  
        Produto p;  
  
        String marca;  
        float valor, valorImp, porc;  
        marca = JOptionPane.showInputDialog("Digite a marca do produto");  
        valor = Float.parseFloat(JOptionPane.showInputDialog("Digite o valor do produto"));  
  
        p = new Produto(marca, valor);  
  
        p.imprimeDados();  
        porc = Float.parseFloat(  
            JOptionPane.showInputDialog("Digite a porcentagem de imposto"));  
        valorImp = p.calculaImposto(porc);  
        JOptionPane.showMessageDialog(null, "Imposto a pagar: " + valorImp);  
    }  
}
```



Exercícios de aplicação



Exercícios de aplicação

1- Implemente as classes disponíveis com o métodos descritos nos diagramas UML abaixo. Crie a classe Java principal, instancie 2 objetos usando os construtores criados, também utilize os outros métodos.

Triangulo
base: float altura: float
Triangulo() Triangulo(b: float, a: float) calculaArea(): float imprimeDados(): String

Data
dia: int mes: int ano: int
Data() Data(d: int, m: int, a: int) cadastraDados(): void imprimeData(): void



Exercícios de aplicação

ContaCorrente
nome: string saldo: float limite: float tipo: char
ContaCorrente(n: string, s: float, l: float, t: char) ContaCorrente(n: string, s: float, t: char) ContaCorrente() cadastraDados(): void imprimeDados(): string depositar(valor: float): void sacar(valor: float): void

2- Implemente as classes disponíveis com o métodos descritos no diagrama UML ao lado. Crie a classe Java principal, instancie 2 objetos usando os construtores criados, também utilize os outros métodos.





That's all Folks!