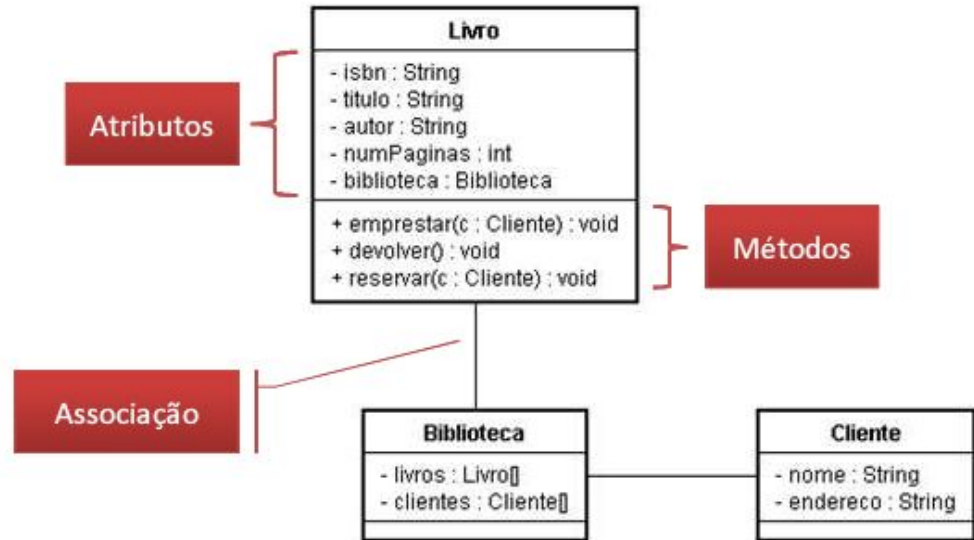


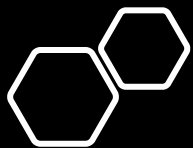
DCC205 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Aula 05

Carlos Bruno Oliveira Lopes
carlosbrunocb@gmail.com

A Notação UML: Diagrama de Classes

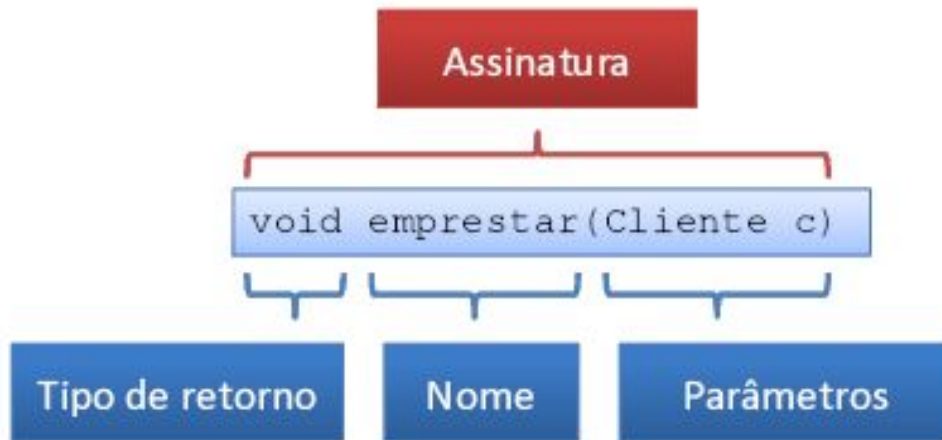




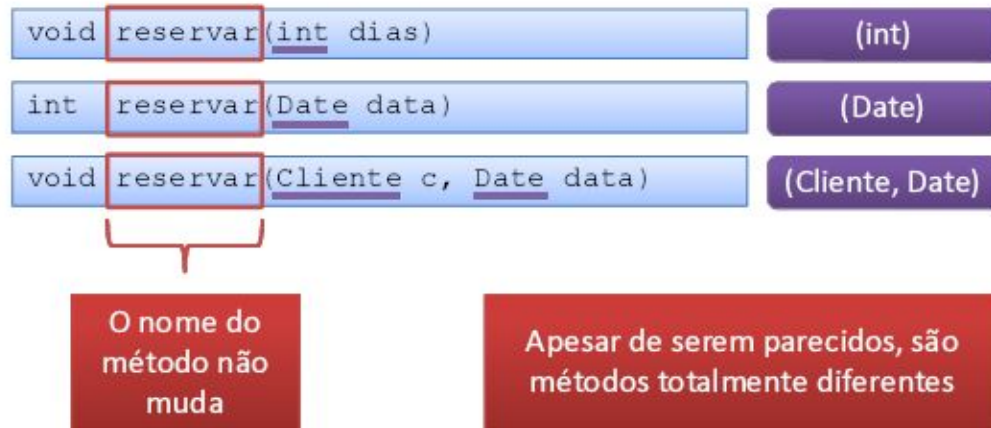
Assinatura de um método

Se um método não retornar valores, é utilizado o **void**

- Um método pode ter zero ou mais parâmetros, e todo parâmetro deve ter um tipo definido



Sobrecarga de métodos (overload)



- Sobrecarregar um método significa criar outros métodos com o mesmo nome, mas com **assinatura** diferente.

Sobrecarga de métodos (overload)



Overload

- Sobrecarga (Overload)
 - Métodos (procedimentos) com o mesmo nome, mas com argumentos diferentes.
 - Pode ser na mesma classe ou em subclasses.
- **void** println()
 - **void** println(**boolean** x)
 - **void** println(**char** x)
 - **void** println(**char[]** x)
 - **void** println(**double** x)
 - **void** println(**float** x)
 - **void** println(**int** x)
 - **void** println(**long** x)
 - **void** println(Object x)
 - **void** println(String x)

Objetos e Referências Relembrando

- Cada objeto criado com o **new** é único
- Os atributos de objetos diferentes pertencem apenas ao objeto

```
Livro livro1 = new Livro();  
livro1.isbn = "1234";
```

```
Livro livro2 = new Livro();  
livro2.isbn = "4321";
```

```
Livro livro3 = new Livro();  
livro3.isbn = "1212";
```

Cada livro possui o
seu próprio ISBN

Criando e Manipulando Objetos (relembrando)

- Um objeto é sempre instância de uma classe
- Para instanciar objetos, é utilizado o new

```
Livro livro1 = new Livro();  
Cliente clientel = new Cliente();
```

- O objeto possui acesso ao que foi definido na sua estrutura (classe) através do "."

```
livro1.titulo = "Aprendendo Java";  
livro1.emprestar(clientel);
```


Objetos e Referências Relembrando

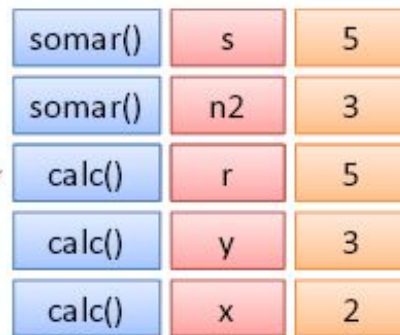
- Uma variável cujo tipo é uma classe não guarda o objeto diretamente.
- A variável guarda uma referência ao objeto.
- O **new** aloca uma área de memória e retorna a referência da área de memória alocada.
- As variáveis declaradas em métodos são criadas numa área de memória chamada **stack**.
- Os objetos são criados numa área de memória chamada **heap**.

Como funciona a Stack

```
void calc() {  
    int x = 2;  
    int y = 3;  
    int r = somar(x, y);  
}
```

```
int somar(int n1, int n2) {  
    int s = n1 + n2;  
    return s;  
}
```

Em Java, os valores dos
parâmetros são sempre copiados



somar()	s	5
somar()	n2	3
calc()	r	5
calc()	y	3
calc()	x	2

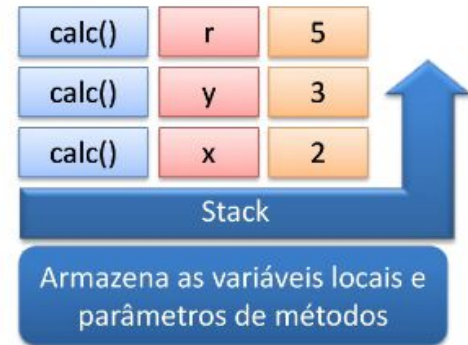
Stack

Armazena as variáveis locais e
parâmetros de métodos

Como funciona a Stack

depois do return

```
void calc() {  
    int x = 2;  
    int y = 3;  
    int r = somar(x, y);  
}  
  
int somar(int n1, int n2) {  
    int s = n1 + n2;  
    return s;  
}
```



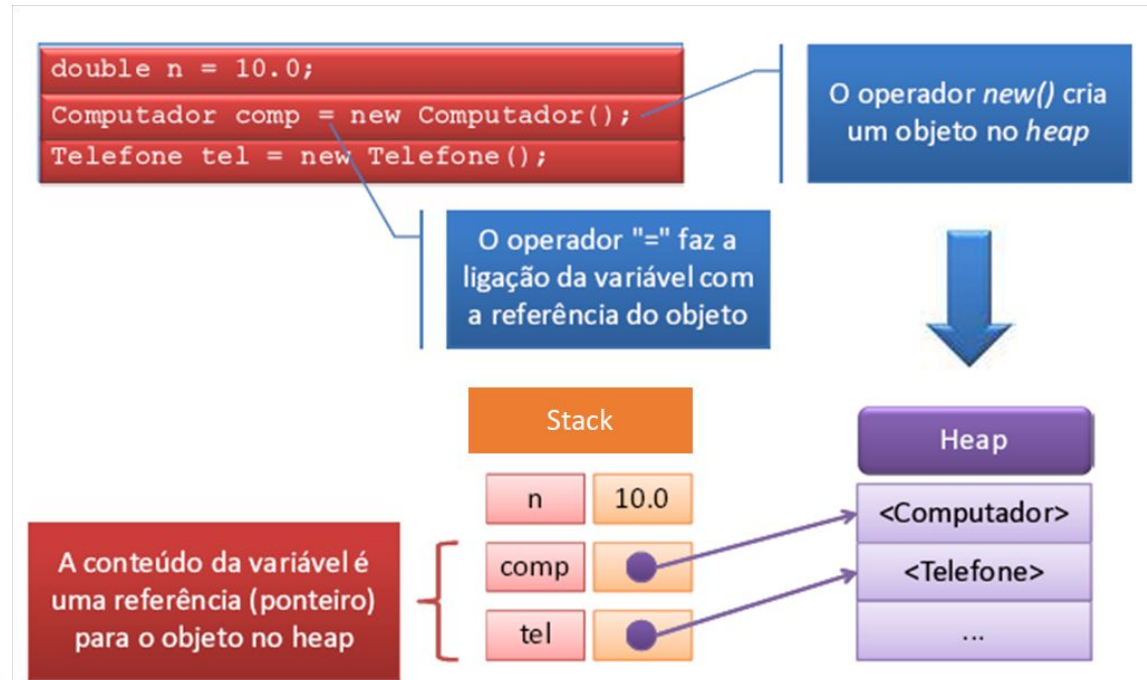
```
void calc() {  
    int x = 2;  
    int y = 3;  
    int r = somar(x, y);  
}  
  
int somar(int n1, int n2) {  
    int s = n1 + n2;  
    return s;  
}
```



Como funciona a Stack

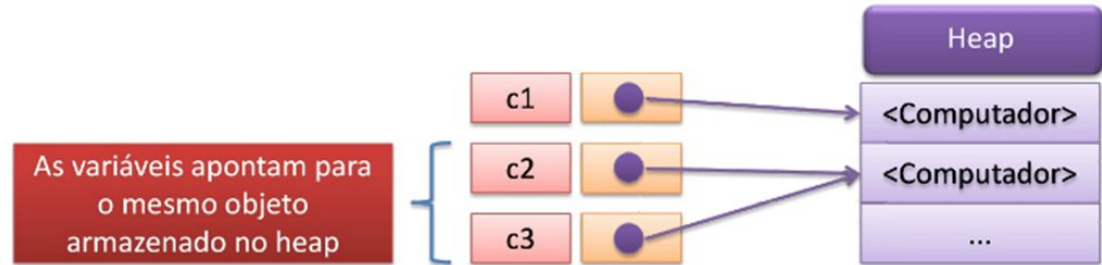
Depois que `calc()` acabar

Como Funciona o Heap



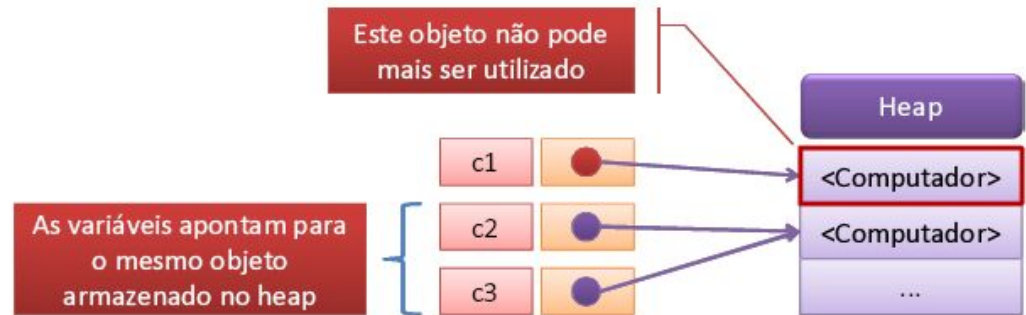
Como Funciona o Heap

```
Computador c1 = new Computador();  
Computador c2 = new Computador();  
Computador c3 = c2;
```



Como Funciona o Heap

```
Computador c1 = new Computador();  
Computador c2 = new Computador();  
Computador c3 = c2;  
c1 = null;
```

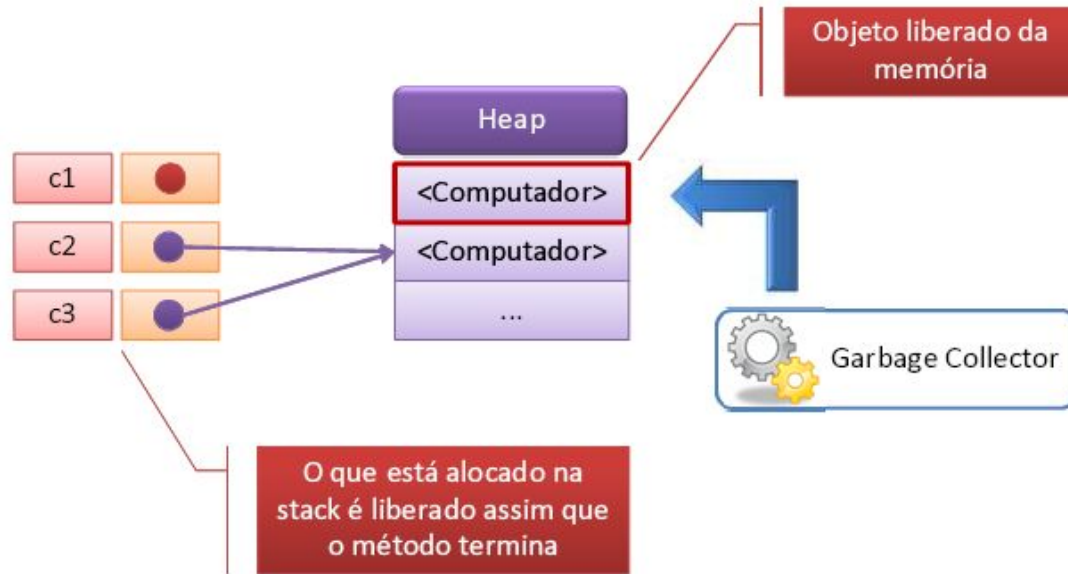


Garbage Collector

- Serviço da JVM que executa em segundo plano
- Procura objetos no heap que não são mais utilizados pela aplicação e os remove
- Não pode ser controlado pelo desenvolvedor.

Obs.: você pode sugerir que ele atue: `System.gc();` ou `Runtime.gc();`

Garbage Collector



Exercício em sala



Duração máxima: 20 minutos



Crie uma classe chamada Matematica e faça alguns métodos oriundos dessa área, como: potencia, soma, subtração, multiplicação e divisão.



O métodos soma poderão receber 2, 3, 4, ou 5 argumentos.

Um problema

- Dentro de um bloco, podemos declarar diversas variáveis e usá-las:

```
int idade1;  
int idade2;  
int idade3;  
int idade4;
```

- Para facilitar esse tipo de caso podemos declarar um **vetor (array)** de inteiros:

```
int[] idades;
```

Arrays

Arrays são utilizados para agrupar dados de um mesmo tipo

```
int[] distancias;  
distancias = new int[8];
```

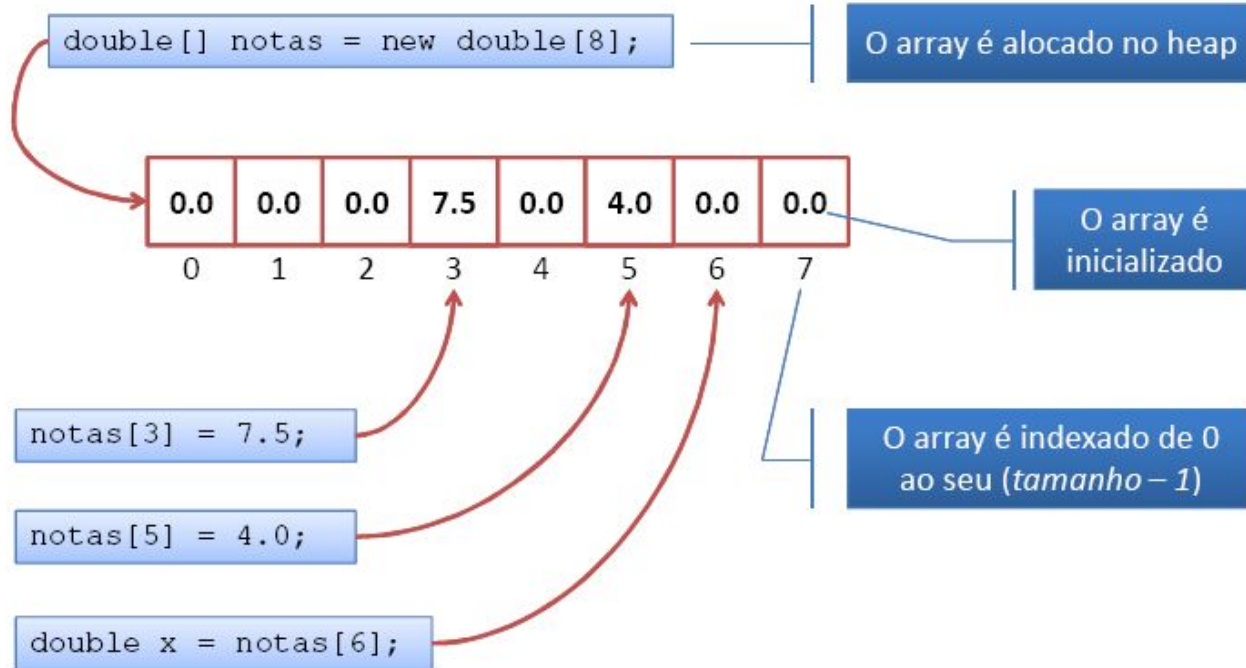
Array de int com 8 posições

```
double[] notas = new double[5];
```

Array de double com 5 posições

Em Java, array é
um objeto

Acessando Elementos do Array



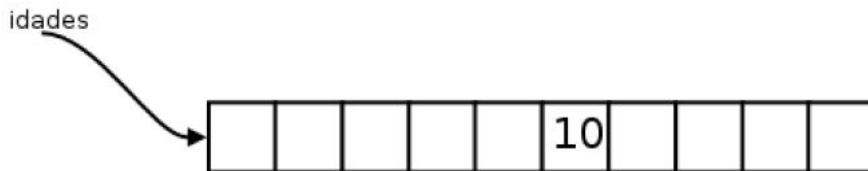
Array

- O `int[]` é um tipo. Uma array é sempre um objeto, portanto, a variável `idades` é uma referência. Vamos precisar criar um objeto para poder usar a array:

```
idades = new int[10];
```

- Criamos uma array de `int` de 10 posições e atribuímos o endereço no qual ela foi criada. Podemos ainda acessar as posições do array:

```
idades[5] = 10;
```



Considerações Sobre Arrays

- Os índices do array vão de 0 a $n - 1$ (onde n é o tamanho do array)
 - Acessos fora deste intervalo resultam em erro.
- Não é possível declarar arrays com tamanho negativo

```
int[] array = new int[-5];
```

- Arrays podem ter tamanho 0

```
int[] array = new int[0];
```

Inicialização de Arrays

```
int[] array = new int[5];
```

```
int array[] = new int[5];
```

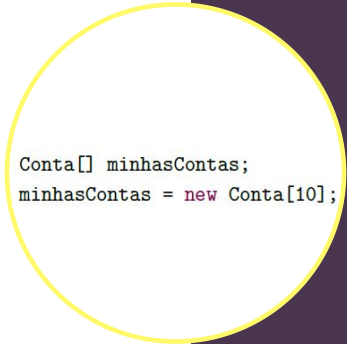
```
int[] array = { 1, 2 };
```

```
int[] array = new int[]{ 1, 2 };
```

Formas de inicializar
os arrays

Arrays de Referência

- É comum ouvirmos “array de objetos”. Porém quando criamos uma array de alguma classe, ela possui referências. O objeto, como sempre, está na memória principal e, na sua array, só ficam guardadas as referências (endereços).



```
Conta[] minhasContas;  
minhasContas = new Conta[10];
```

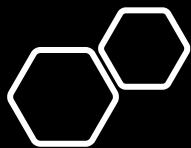
Quantas contas foram criadas aqui?

- Você deve **popular** seu array antes:

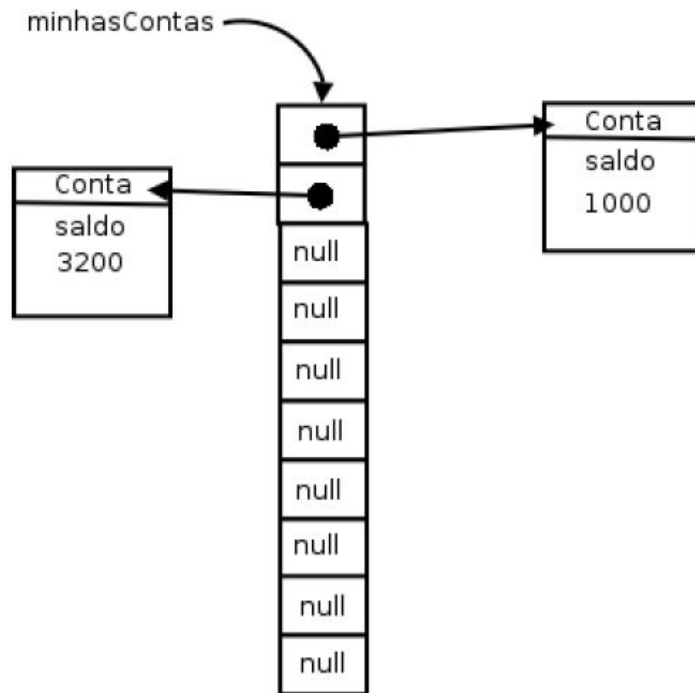
```
Conta contaNova = new Conta();  
contaNova.saldo = 1000.0;  
minhasContas[0] = contaNova;
```

- Ou você pode fazer isso diretamente:

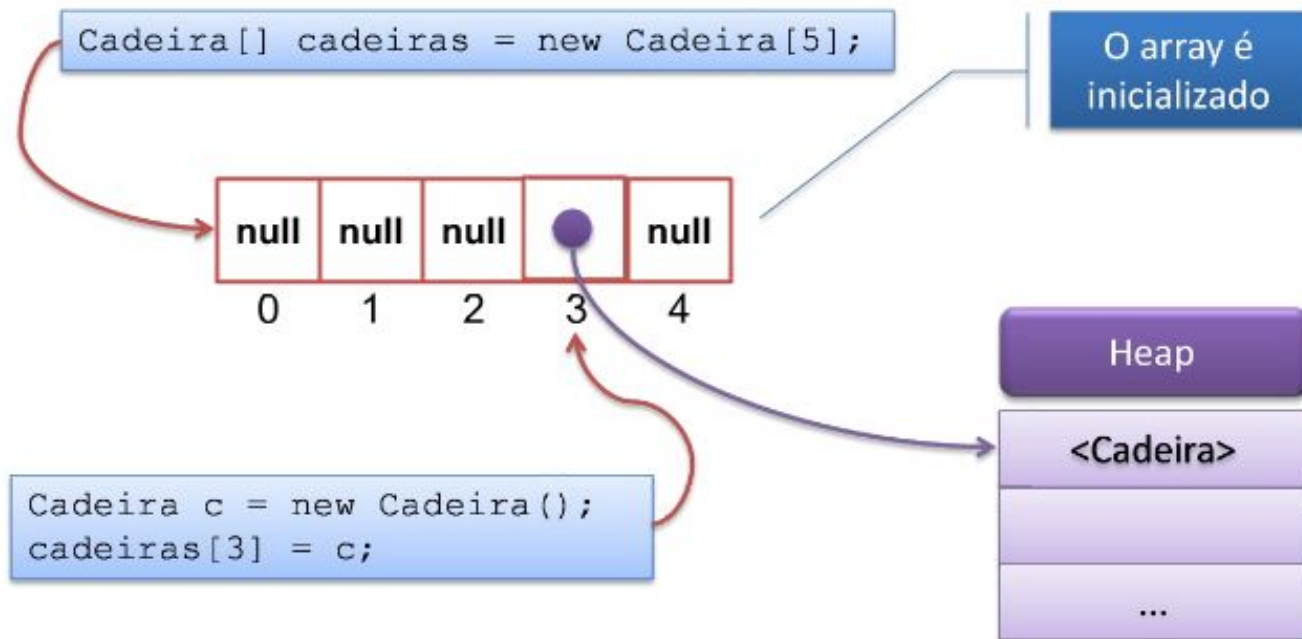
```
minhasContas[1] = new Conta();  
minhasContas[1].saldo = 3200.0;
```



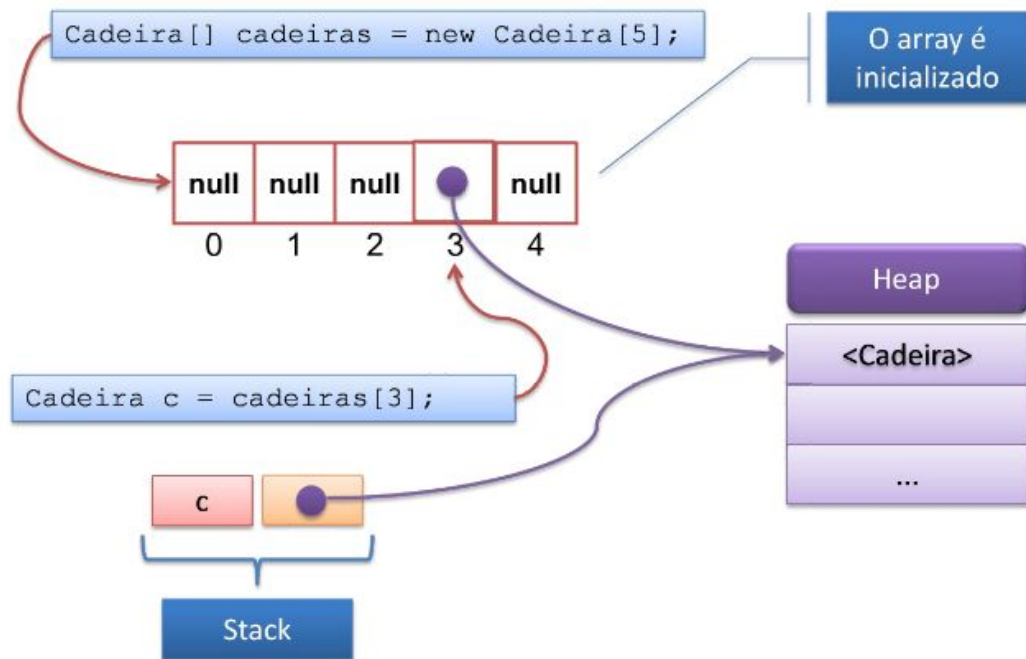
Uma array de tipos primitivos guarda valores, uma array de objetos guarda referências.



Arrays de Referências



Arrays de Referências



Percorrendo uma Array

- Utilizando o for

```
int[] array = new int[10];  
  
for(int i = 0; i < array.length; i++) {  
    System.out.println(array[i]);  
}
```

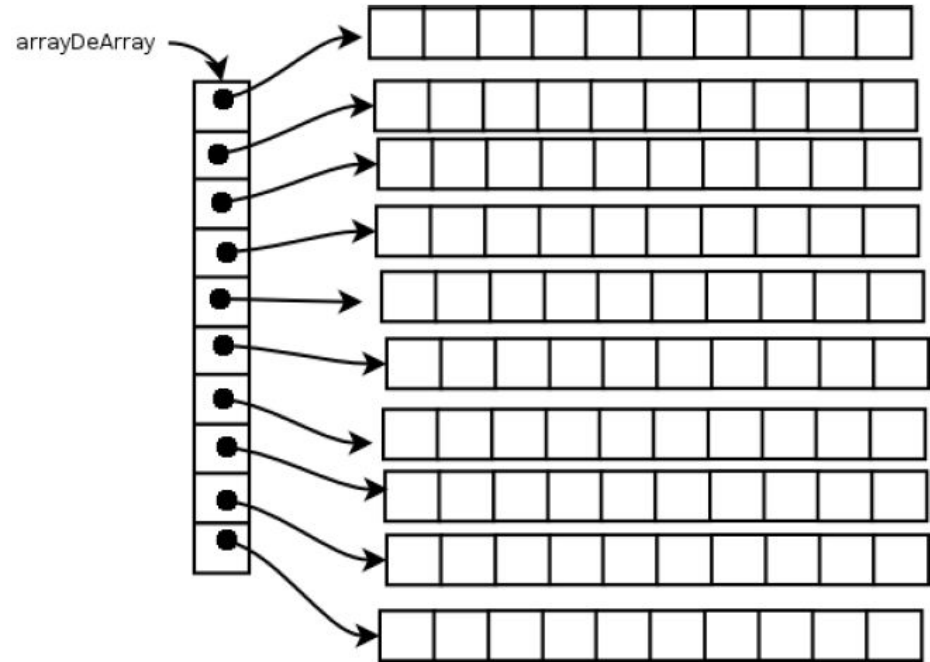
- Utilizando o enhanced-for

```
int[] array = new int[10];  
  
for(int i : array) {  
    System.out.println(i);  
}
```

O mesmo é válido para arrays de referências.

Um pouco mais...

- Arrays podem ter mais de uma dimensão.
- Podemos ter uma array de 10 por 10 contas;
- Ele pode ser acessado pela posição da coluna x e linha y.
 - Na verdade, um array bidimensional é uma array de arrays.



Outros Exemplos...

```
int[] array = new int[5];

array[1] = 6;
array[4] = 7;

for (int i = 0; i < array.length; i++) {
    System.out.println(array[i]);
}

for (int x : array) {
    System.out.println(x);
}
```




Outros Exemplos...

```
for (int x : array) {  
    System.out.println(x);  
}
```

```
String[] array2 = new String[7];  
array2[0] = "a";  
array2[1] = "b";
```

```
for (String s : array2) {  
    System.out.println(s);  
}
```

```
Object[] array3 = new Object[7];  
array3[0] = "a";  
array3[1] = new Object();  
array3[2] = new int[3];  
array3[3] = 10;  
  
String e1 = (String) array3[0];  
int e2 = (int) array3[3];
```

Outros Exemplos...

Outros Exemplos...

```
public class ArrayMultiDimensional {  
  
    public static void main(String[] args) {  
        int[][] array = new int[2][2];  
  
        Scanner leitor = new Scanner(System.in);  
  
        for (int i=0; i<array.length; i++) {  
            for (int j=0; j < array[i].length; j++) {  
                System.out.printf("Entre com o elemento array[%d][%d]\n", i+1, j+1);  
                array[i][j] = leitor.nextInt();  
            }  
        }  
        System.out.println("Impressao de valores");  
        for (int i=0; i<array.length; i++) {  
            for (int j=0; j<array[i].length; j++) {  
                System.out.printf("%d ",array[i][j]);  
            }  
        }  
    }  
}
```

Outros Exemplos...

```
//são três array[2][4]
int [][][] array2 = new int[3][2][4];
array2[2][1][2] = 3;
array2[1][1][3] = 2;

for (int i = 0; i<array2.length; i++) {
    for (int j = 0; j<array2[i].length; j++) {
        for (int k=0; k<array2[i][j].length;k++) {
            System.out.print(array2[i][j][k] + " ");
        }
        System.out.println();
    }
    System.out.println();
}
```

Saída

0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 2
0 0 0 0
0 0 3 0

Desvantagens dos arrays

- Depois de criado, não é possível modificar o tamanho de um array
- Dificuldade em encontrar elementos dentro do array quando o índice não é conhecido
- Ao remover elementos, sobram “buracos” no array

Varargs

- O uso de varargs permite que métodos possam receber um número variável de parâmetros.

Varargs

```
public int somar(int... valores) {  
    //...  
}
```

```
somar(10, 20, 30);
```

```
somar(10, 20);
```

```
somar(10);
```

```
somar();
```

Lendo os Parâmetros do Varargs

```
public int somar(int... valores) {  
    int soma = 0;  
    for (int valor : valores) {  
        soma += valor;  
    }  
    return soma;  
}
```

- Os parâmetros passados via varargs são lidos como arrays

39

```
int[] array = { 10, 20, 30 };  
somar(array);
```

- É possível passar o parâmetro diretamente como um array

Exemplo Prático sem varargs

40

```
public class Aplicacao {  
  
    public static void main(String[] args) {  
  
        imprimir("a");  
        imprimir("a", "b");  
        imprimir("a", "b", "c");  
  
    }  
  
    public static void imprimir(String s1) {  
        System.out.println(s1);  
    }  
  
    public static void imprimir(String s1, String s2) {  
        System.out.println(s1);  
        System.out.println(s2);  
    }  
  
    public static void imprimir(String s1, String s2, String s3) {  
        System.out.println(s1);  
        System.out.println(s2);  
        System.out.println(s3);  
    }  
}
```


Exemplo Prático com varargs

41

```
public class Aplicacao {  
  
    public static void main(String[] args) {  
  
        imprimir("a");  
        imprimir("a", "b");  
        imprimir("a", "b", "c");  
        imprimir();  
  
        String[] array = { "a", "b", "c" };  
        imprimir(array);  
  
    }  
  
    public static void imprimir(String... textos)  
        for (int i = 0; i < textos.length; i++) {  
            System.out.println(textos[i]);  
        }  
    }  
}
```

Ordem dos Parâmetros do Varargs

42

```
public void metodo(int x, boolean y, String... params) {  
    //...  
}
```

- Parâmetros do tipo varargs podem ser misturados com parâmetros “normais”
- Parâmetros varargs devem ser sempre os últimos definidos no método

Exercício em sala

- **Modifique a Classe Matemática:**

- Relembrando: Crie uma classe chamada Matemática e faça alguns métodos oriundos dessa área, como: potencia, soma, subtração, multiplicação e divisão.
- O métodos soma poderão receber um número indefinido de argumentos. Use varargs.

43

Atividade para entregar

44

1. Volte ao nosso sistema de Funcionário e crie uma classe Empresa dentro do mesmo arquivo .java. A Empresa tem um nome, cnpj e uma referência a uma array de Funcionário, além de outros atributos que você julgar necessário.

```
class Empresa {  
    // outros atributos  
    Funcionario[] empregados;  
    String cnpj;  
}
```

Atividade para entregar

2. A Empresa deve ter um método adiciona, que recebe uma referência a Funcionário como argumento e guarda esse funcionário. Algo como:

```
void adiciona(Funcionario f) {  
    // algo tipo:  
    //   this.empregados[ ??? ] = f;  
    // mas que posição colocar?  
}
```

Atividade para entrega

3. Crie uma classe TestaEmpresa que possuirá um método main. Dentro dele crie algumas instâncias de Funcionário e passe para a empresa pelo método adiciona. Repare que antes você vai precisar criar a array, pois inicialmente o atributo empregados da classe Empresa não referencia lugar nenhum (seu valor é null):
- Dica: Você pode criar esses funcionários dentro de um loop e dar a cada um deles valores diferentes de salários:

```
for (int i = 0; i < 5; i++) {  
    Funcionario f = new Funcionario();  
    f.salario = 1000 + i * 100;  
    empresa.adiciona(f);  
}
```

Atividade para entregar

4) Percorra o atributo empregados da sua instância da Empresa e imprima os salários de todos seus funcionários. Para fazer isso, você pode criar um método chamado mostraEmpregados dentro da classe Empresa:

```
void mostraEmpregados() {  
    for (int i = 0; i < this.empregados.length; i++) {  
        System.out.println("Funcionário na posição: " + i);  
        // preencher para mostrar outras informacoes do funcionario  
    }  
}
```

Atividade para entregar

5) Em vez demonstrar apenas o salário de cada funcionário, você pode chamar o método `mostra()` de cada `Funcionario` da sua array.

Dica: Cuidado com os campos não contextualizados.

6) Crie um método ou uma solução para verificar se um determinado `Funcionário` se encontra ou não como funcionário desta empresa:

Atividade para entregar

- Classe Porta

Classe: Porta

Atributos: aberta, cor, dimensaoX, dimensaoY, dimensaoZ

Métodos: void abre()

void fecha()

void pinta(String s)

boolean estaAberta()

Crie uma porta, abra e feche a mesma, pinte-a de diversas cores, altere suas dimensões e use o método estaAberta para verificar se ela está aberta.

Atividade para entregar

O objetivo deste exercício é fixar os conceitos vistos. Aproveite e treine tudo o que vimos até agora no pequeno programa abaixo:

- Programa: Classe: Casa
- Atributos: cor, totalDePortas, portas[]
- Métodos: void pinta(String s), int quantasPortasEstaoAbertas(), void adicionaPorta(Porta p), int totalDePortas()

Crie uma casa, pinte-a. Crie três portas e coloque-as na casa através do método adicionaPorta, abra e feche-as como desejar. Utilize o método quantasPortasEstaoAbertas para imprimir o número de portas abertas e o método totalDePortas para imprimir o total de portas em sua casa.