

Orientação a Obejtos Classica

Namom Alves Alencar



ARRAY

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

```
int idade1;
```

```
int idade2;
```

```
int idade3;
```

```
int idade4;
```

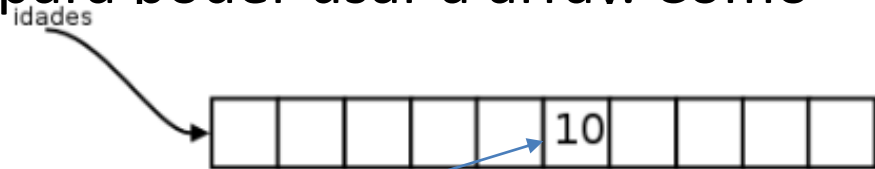
- Isso pode se tornar um problema quando precisamos mudar a quantidade de variáveis a serem declaradas de acordo com um parâmetro. Esse parâmetro pode variar, como por exemplo, a quantidade de números contidos num bilhete de loteria. Um jogo simples possui 6 números, mas podemos comprar um bilhete mais caro, com 7 números ou mais.
- Para facilitar esse tipo de caso podemos declarar um vetor (array) de inteiros:

```
int[] idades;
```

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. Como criamos o objeto-array?

`idades = new int[10];`



- O que fizemos foi criar uma array de `int` de 10 posições e atribuir o endereço no qual ela foi criada. Podemos ainda acessar as posições do array:

`idades[5] = 10;`

- O código acima altera a sexta posição do array. No Java, os índices do array vão de 0 a $n-1$, onde n é o tamanho dado no momento em que você criou o array. Se você tentar acessar uma posição fora desse alcance, um erro ocorrerá durante a execução.

ARRAY

- No caso do bilhete de loteria, podemos utilizar o mesmo recurso. Mais ainda, a quantidade de números do nosso bilhete pode ser definido por uma variável. Considere que n indica quantos números nosso bilhete terá, podemos então fazer:

```
int numerosDoBilhete[] = new int[n];
```

- E assim podemos acessar e modificar os inteiros com índice de 0 a $n-1$.

ARRAY DE OBJETOS

- 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? Na verdade, nenhuma. Foram criados 10 espaços que você pode utilizar para guardar uma referência a uma Conta. Por enquanto, eles se referenciam para lugar nenhum (null). Se você tentar:

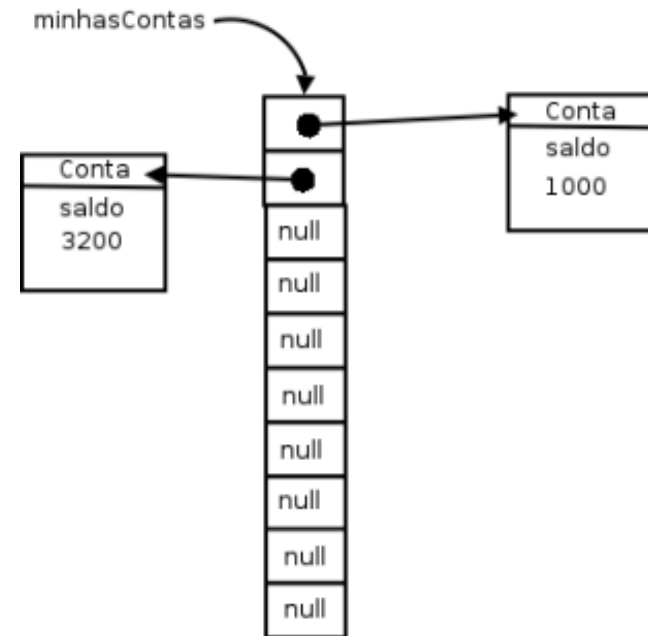
```
System.out.println(minhasContas[0].saldo);
```

Está correto?! Qual o meu retorno? O que a tela imprimirá?

ARRAY DE OBJETOS

- Um erro durante a execução ocorrerá! Pois, na primeira posição da array, não há uma referência para uma conta, nem para lugar nenhum. Você deve popular sua 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.

ARRAY DE OBJETOS

- Percorrer uma array é muito simples quando fomos nós que a criamos:

```
public static void main(String args[]) {  
    int[] idades = new int[10];  
    for (int i = 0; i < 10; i++) {  
        idades[i] = i * 10;  
    }  
    for (int i = 0; i < 10; i++) {  
        System.out.println(idades[i]);  
    }  
}
```

ARRAY DE OBJETOS

- Porém, em muitos casos, recebemos uma array como argumento em um método:

```
void imprimeArray(int[] array) {  
    // não compila!!  
    for (int i = 0; i < ???; i++) {  
        System.out.println(array[i]);  
    }  
}
```

- Até onde o for deve ir? Toda array em Java tem um atributo que se chama length, e você pode acessá-lo para saber o tamanho do array ao qual você está se referenciando naquele momento:

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


ARRAY DE OBJETOS

- No caso de você não ter necessidade de manter uma variável com o índice que indica a posição do elemento no vetor (que é uma grande parte dos casos), podemos usar o enhanced-for.

```
class AlgumaClasse{  
    public static void main(String args[]) {  
        int[] idades = new int[10];  
        for (int i = 0; i < 10; i++) {  
            idades[i] = i * 10;  
        }  
        // imprimindo toda a array  
        for (int x : idades) {  
            System.out.println(x);  
        }  
    }  
}
```

ARRAY DE OBJETOS

- Não precisamos mais do `length` para percorrer matrizes cujo tamanho não conhecemos:

```
class AlgumaClasse {  
    void imprimeArray(int[] array) {  
        for (int x : array) {  
            System.out.println(x);  
        }  
    }  
}
```

- O mesmo é válido para arrays de referências. Esse `for` nada mais é que um truque de compilação para facilitar essa tarefa de percorrer arrays e torná-la mais legível.

EXERCÍCIOS

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

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

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

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

- Você deve inserir o Funcionario em uma posição da array que esteja livre. Existem várias maneiras para você fazer isso: guardar um contador para indicar qual a próxima posição vazia ou procurar por uma posição vazia toda vez. O que seria mais interessante?

EXERCÍCIOS

2. Crie uma classe TestaEmpresa que possuirá um método main. Dentro dele crie algumas instâncias de Funcionario 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):

```
Empresa empresa = new Empresa();  
empresa.empregados = new Funcionario[10];
```

```
// ....
```

- Ou você pode construir a array dentro da própria declaração da classe Empresa, fazendo com que toda vez que uma Empresa é instanciada, a array de Funcionario que ela necessita também é criada.
- Crie alguns funcionários e passe como argumento para o adiciona da empresa:

```
Funcionario f1 = new Funcionario();  
f1.salario = 1000;  
empresa.adiciona(f1);  
...
```

EXERCÍCIOS

- 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);  
}
```

- Repare que temos de instanciar Funcionario dentro do laço. Se a instanciação de Funcionario ficasse acima do laço, estaríamos adicionado cinco vezes a **mesma** instância de Funcionario nesta Empresa e apenas mudando seu salário e nome a cada iteração, que nesse caso não é o efeito desejado.

3. No método adiciona gere uma mensagem de erro indicando quando o array já está cheio.

EXERCÍCIOS

4. Percorra o atributo empregados da sua instância da Empresa e imprima as informações 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  
        //chamar o metodo em funciona que mostra os seus atributos  
    }  
}
```

- Cuidado ao preencher esse método: alguns índices do seu array podem não conter referência para um Funcionario construído, isto é, ainda se referirem para null. Se preferir, use o foreach.
- Aí, através do seu main, depois de adicionar alguns funcionários, basta fazer: empresa.mostraEmpregados();

EXERCÍCIOS

5. Crie um método para verificar se um determinado Funcionario se encontra ou não como funcionário desta empresa:

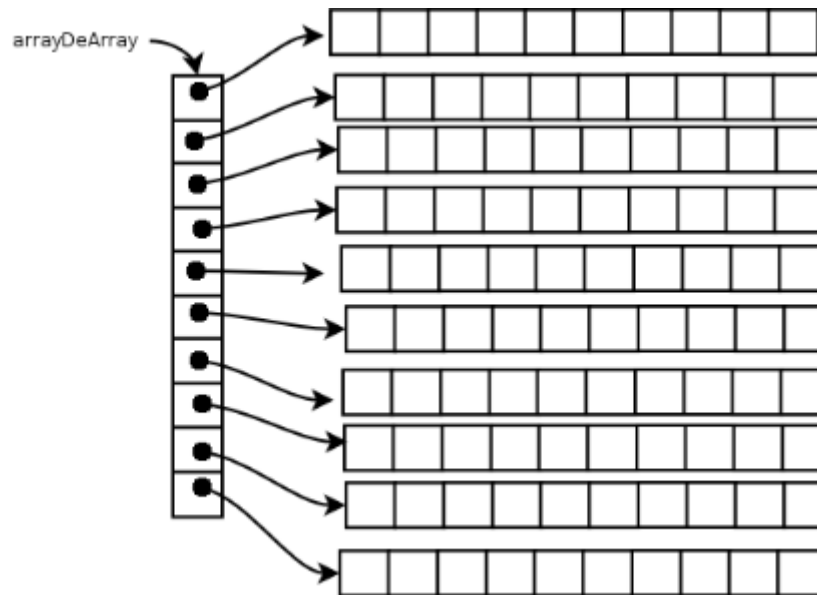
```
boolean contem(Funcionario f) {  
    // ...  
}
```

- Você vai precisar fazer um for na sua array e verificar se a referência passada como argumento se encontra dentro da array. Evite ao máximo usar números hard-coded, isto é, use o `.length` ou o atributo livre.

6. Caso a array já esteja cheia no momento de adicionar um outro funcionário, criar uma nova maior e copiar os valores. Isto é, fazer a realocação já que java não tem isso: uma array nasce e morre com o mesmo length. Implementação do ArrayList.

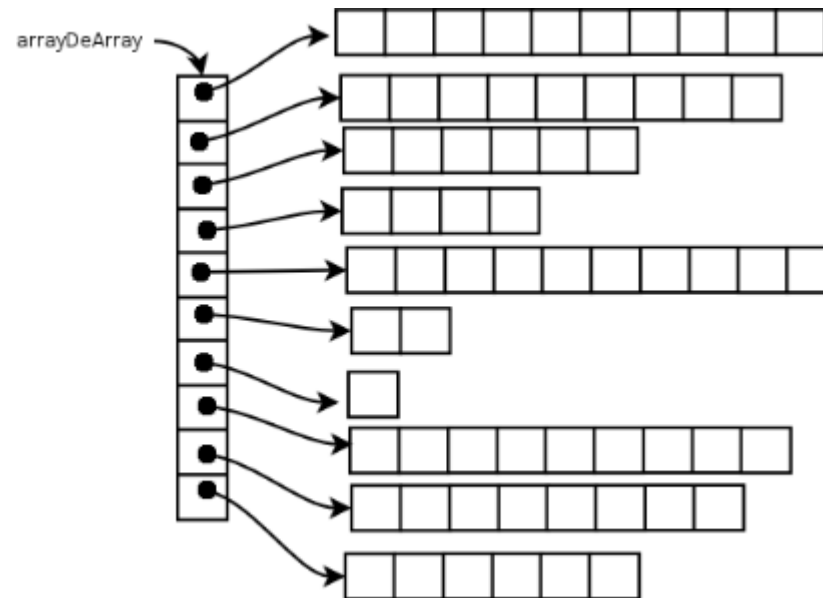
ARRAY N-DIMENSIONAL DE OBJETOS

- Arrays podem ter mais de uma dimensão. Isto é, em vez de termos uma array de 10 contas, podemos ter uma array de 10 por 10 contas e você pode acessar a conta na posição da coluna x e linha y. Na verdade, uma array bidimensional em Java é uma array de arrays.



ARRAY N-DIMENSIONAL DE OBJETOS

- Uma array bidimensional não precisa ser retangular, isto é, cada linha pode ter um número diferente de colunas. Como? Porque?



ARRAY CURIOSIDADES

1. O que acontece se criarmos uma array de 0 elementos? e -1?
2. O método main recebe uma array de Strings como argumento. Essa array é passada pelo usuário quando ele invoca o programa:

- \$ java Teste um dois tres

```
class Teste {  
    public static void main (String[] args) {  
        for(String argumento: args) {  
            System.out.println(argumento);  
        }  
    }  
}
```

Bons Estudos

Namom Alves Alencar

