

Simples e Multidimensionai

Em boa parte dos programas, existem situações onde será necessário agrupar uma sequência de dados de um mesmo tipo. Como por exemplo, uma sequência de valores que representam os preços de produtos que estão em uma sacola de compras de um cliente em um *e-commerce*. Se cada valor for tratado com uma respectiva variável, existe o risco da pessoa programadora ter de lidar com uma quantidade muito grande de variáveis no código no caso da manipulação de muitos dados simultaneamente. Para evitar este tipo de situação e, principalmente, organizar a forma como os agrupamentos podem ser feitos, o Java possibilita a formação do que é chamado de **agregados**. Esses agregados são popularmente chamados de **vetores** ou **arrays**.



Os agregados podem ter várias características diferentes. O mais simples deles é o chamado **agregado homogêneo**, ou seja, uma conjunto de dados de um mesmo tipo. Um agregado deste tipo é feito da seguinte forma:

```
float precos [] = new float[3]; // array com 3 posições
```

```
precos[0] = 10.50;  
precos[1] = 5.60;  
precos[2] = 3.30;  
  
// o mesmo array pode ser declarado da seguinte forma:  
float precos[] = {10.5, 5.60, 3.30};
```

Neste exemplo, um novo *array* é criado usando os colchetes ([]). No lado esquerdo da atribuição, é informado o tipo de dados que irá compor este agregado. Do lado direito, é informado a quantidade de dados que serão alocados para este conjunto (lembrando que todos os conjuntos começam a partir do índice zero).

Os agregados também podem ter mais de uma dimensão. O mais utilizado é o **agregado bidimensional**, também conhecido como **matriz**. No exemplo abaixo, a matriz de

números é representada usando um  array bidimensional:

matriz = 1 2

3 4

```
int matriz[][] = new int [2][2];  
matriz[0][0] = 1;  
matriz[0,1] = 2;  
matriz[1,0] = 3;  
matriz[1,1] = 4;  
  
// o mesmo array pode ser declarado da seguinte forma:  
int matriz[][] = { {1,2}, {3,4} };
```

De forma análoga aos agregados unidimensionais, as matrizes são representadas por dois conjuntos de colchetes. O primeiro indica a posição da linha e o segundo a posição da coluna.

Desde os *arrays* mais simples até os mais complexos, a forma de se navegar por eles não é muito diferente. Para os tipos mais simples, um código com um laço de repetição *for* costuma ser o suficiente.





Veja o exemplo abaixo onde um *array* de números naturais é varrido item a item é exibido no *console*:

```
int jogadores [] = { 9, 10, 22, 31, 4 };  
for(int i = 0; i < jogadores.length; i++) {  
    System.out.println(jogadores[i]);  
}
```

Neste código, a variável *i* é criada para assumir o papel de índice (*index*), ou seja, ela indica em qual passo a iteração está. Em seguida, a expressão *i < jogadores.length* representa a condição para que a iteração continue, ou seja, quanto o valor da variável *i* for menor que a quantidade de itens no *array* jogadores, a iteração avança. O avanço da iteração é representado pelo *i++*, que indica que a variável *i* avançará um número por vez - seu equivalente seria $i = i + 1$.

Se um *array* for bidimensional basta incrementar o código acima inserindo uma nova iteração dentro da iteração. Isso fará com que a matriz seja lida linha a linha.

```
int matriz [][] = { {1,1}, {2,2}, {3,3} };  
for(int i = 0; i < matriz.length; i++) {  
    for(int j = 0; j < matriz[i].length; j++) {  
        System.out.println(matriz[i][j]);  
    }  
}
```



No Java, existem um conjunto de classes que abstraem as operações com *arrays* e facilitam a vida da pessoa programadora. Essas estruturas são chamadas de Listas (*Lists*). As três estruturas mais comuns para lidar com listas são a *ArrayList*, *LinkedList* e *Vector*. Apesar de serem bem parecidas, as três implementações são diferentes e apresentam performances distintas nas operações de inserção, remoção e iteração. A não ser que a questão seja otimização, a estrutura *ArrayList* geralmente é a mais recomendada.

No entanto, tipos primitivos não são aceitos nesta estrutura pois estes tipos não estendem a classe *Object*, base das demais classes do Java. Ou seja, para o uso das listas é necessário o uso de tipos complexos ou classes *wrapper*. Abaixo segue a criação de uma lista usando a classe *ArrayList*.



```
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Integer> numeros = new ArrayList<Integer>();
        numeros.add(1);
        numeros.add(2);
        numeros.add(3);

        System.out.println(numeros);
    }
}
```



Ao utilizar estas classes, é necessário indicar de que tipo serão formadas as listas. Isso é representado através dos símbolos de menor que (<) e maior que (>). Isso faz que o Java garanta que itens de outros tipos, por exemplo String, não sejam inseridos na lista.

Para realizar operações nesta lista, basta usar os métodos listados abaixo.

Assinatura do método	O que faz?
add()	Adicione um valor no final da lista.
size()	Retorna ao tamanho da lista.
get(int i)	Obtém o valor que corresponde ao índice passado por parâmetro.
remove(int i)	Remove da lista o valor que corresponde ao índice passado por parâmetro.

Tabela 1 - Métodos da interface List do Java.



Por fim, uma forma prática de alterar listas é usando o laço de repetição *for*. Como a classe *List* é um iterável, é possível usar o laço *for* de maneira ainda mais prática, conforme mostra o código seguir:

```
import java.util.ArrayList;
import java.util.List;

public class Main {


    public static void main(String[] args) {
        List<Integer> numeros = new ArrayList<Integer>();
        numeros.add(1);
        numeros.add(2);
        numeros.add(3);

        for(Integer numero: numeros) {
            System.out.println(numero);
        }
    }
}
```

```
}
}
```

Atividade Extra



Artigo: Orientação a Objetos: Um pouco de Arrays 



Artigo: Java Multidimensional Arrays



Exercícios: Java Array: Exercises, Practice, Solution

Referência Bibliográfica

BARNES, D. J.; KOLLING, M. **Programação orientada a objetos com java: um**

introdução prática usando o bluej. ¹ ed.  Pearson: 2009.

FELIX, R. (Org.). Programação orientada a objetos. Pearson: 2017.

MEDEIROS, L. F. de. Banco de dados: princípios e prática. Intersaberes: 2013;

ORACLE. Java Documentation, 2021. Documentação oficial da plataforma Java. Disponível em:
<<https://docs.oracle.com/en/java/>>. Acesso em 12 de Jul. de 2021.

Ir para questão

