LABORATÓRIO: ARRAYS EM JAVA

Última atualização: 03/05/2023

Exercício 1 – Manipulação de Arrays em Java

Objetivo do exercício – Investigaremos neste exercício atribuição de variáveis de referência em Java.

Tarefas:

Utilizando *arrays* simples:

1. Crie uma classe chamada `TesteArrays`. No método `main`, declare duas variáveis chamadas `array1` e `array2`. Elas devem ser do tipo `int[]`. (*Array* de inteiros)
2. Utilize chaves { }, para inicia `array1` com os oito primeiros números primos: 2, 3, 5, 7, 11, 13, 17, e 19.
3. Exiba o conteúdo de `array1`. Você pode querer utilizar o método `exibirArray`, mostrado abaixo para exibir este *array* de inteiros (**obs:** casso você copie cole o conteúdo deste código no editor do seu IDE e alguns erros aparecerem de caracteres ilegais, certifique-se que as aspas não apresentam problemas. Normalmente basta vocês substituirem as aspas que apresentam problemas no momento de colar o código no editor). Aprenderemos o significado da palavra-reservada `static` em uma outra lista de exercícios. Não deixe de analisar o entendimento do método `exibirArray`.

```
public static void exibirArray(int[] array) {
  System.out.print("<");
  for (int i = 0; i < array.length; i++) {
    // Imprime um elemento
    System.out.print(array[i]);
    // Imprime uma vírgula como delimitador se não for o
    // último elemento
    if ((i+1) < array.length) {
      System.out.print(", ");
     }
  }
  System.out.print(">");
}
```

Listagem 1. Código exibirArray.

4. Execute o programa `TesteArrays`. A saída deverá ser similar à seguinte:

```
Imprimindo array1
<2, 3, 5, 7, 11, 13, 17, 19>
```

5. Atribua a variável `array1` à `array2`. Modifique os elementos de índice par dentro de `array2` para serem iguais aos valores do índices. (Por exemplo, `array2[0] = 0;` `array2[2] = 2;` etc). Imprima `array1`. Execute `TestArrays`. O que ocorreu com `array1`?

Utilizando arrays multidimensionais:

1. Declare uma variável chamada `matriz` com o tipo `int[][]` (um *array* de *arrays* de `int`). Inicie `matriz` com um *array* de cinco *arrays*.

2. Inicie cada um dos *arrays* internos da seguinte maneira: percorra `matriz` de zero até seu tamanho máximo; vamos supor que este índice é `i`. Em cada iteração, atribua `matriz[i]` a um novo `array` de inteiros o tamanho de `i`. Percorra cada elemento neste *array* (de ints), com a variável de índice `j`. Em cada iteração interna atribua `matriz[i][j]` o valor de `(i*j)`.

3. Imprima `matriz` percorrendo o *array* mais externo e imprimindo cada *array* interno em uma linha separada usando o método `exibirArray`.

4. Execute `TesteArrays`. A saída deve ser algo similar à tela abaixo :

```
Imprimindo a matriz
<>
<0>
<0, 2>
<0, 3, 6>
<0, 4, 8, 12>
```

Exercício 2 – Utilizando a classe java.util.Arrays

Objetivo do exercício – Em vez de uasr o método `exibirArray` fornecido até o momento, usar os métodos `toString()` e `deepToString()` da classe utilitária `java.util.Arrays`

Tarefas:

1. Você pode continuar usando o método `main` da classe `TesteArrays` criada anteriormente. Agora, em vez de imprimir `array1`, `array2` e `matriz` usando o método `exibirArray`, vamos usar métodos já existentes no JRE.

2. Use os métodos `toString()` e `deepToString()`, da classe `java.util.Arrays`, para exibição dos conteúdos. A saída deverá ser similar à abaixo:

```
Imprimindo array1, array2 e matriz usando métodos estáticos da classe Arrays
[0, 3, 2, 7, 4, 13, 6, 19]
[0, 3, 2, 7, 4, 13, 6, 19]
[[], [0], [0, 2], [0, 3, 6], [0, 4, 8, 12]]
```

3. Em outras palavras, código similar ao fornecido para o método `exibirArray` para exibir o conteúdo de *arrays* já é fornecido na biblioteca padrão da linguagem Java.

Exercício 3 – Utilizando Arrays para Representar Multiplicidade

Objetivo do exercício – Neste exercício, utilizaremos *arrays* para implementar a multiplicidade na associação entre um banco e seus clientes.

Tarefas:

Adicionaremos ao pacote `banco`, criado em listas anteriores, uma classe chamada `Banco` conforme diagrama de classes abaixo.

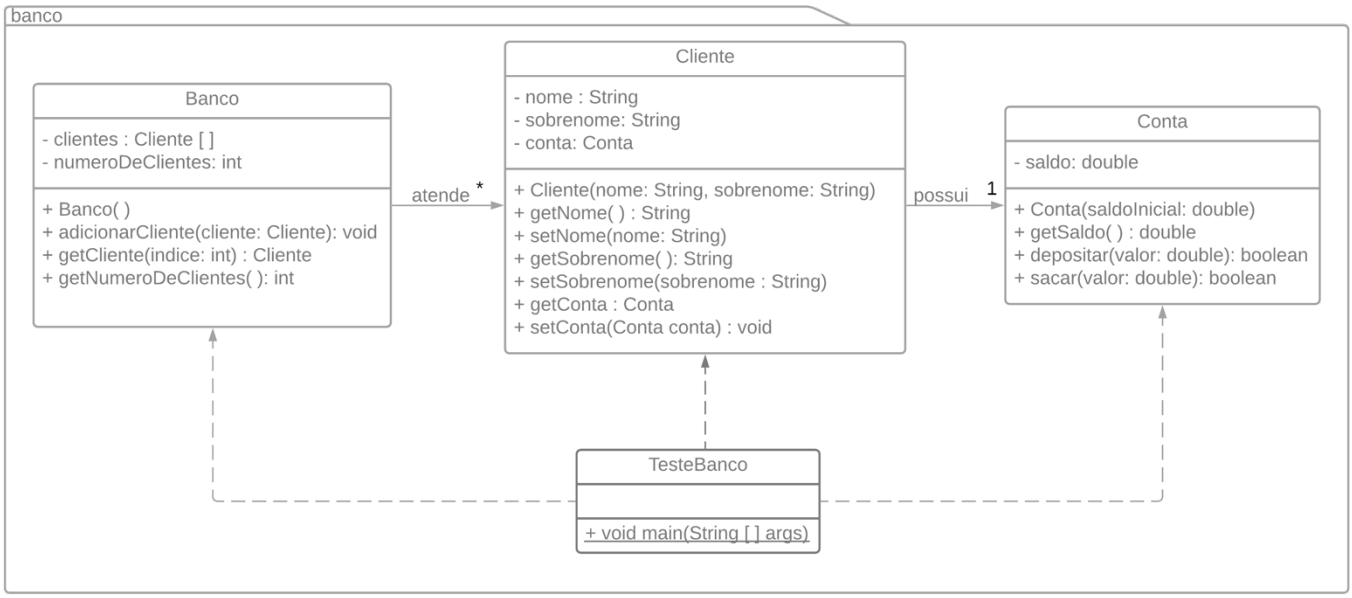

Figura 1. Diagrama de classes para o projeto Banco.

Um sistema bancário armazena associações entre o banco e seus clientes. Implementamos esta associação com um *array* de objetos do tipo `Cliente`. Também necessitamos manter um atributo inteiro que armazena o número de clientes do banco.

Importante: usar *arrays* para representar multiplicidade na associação de objetos é uma prática ruim de programação tendo em vista que precisamos saber de antemão o tamanho do *array*, o que não é sempre possível. No próximo exercício desta lista usaremos uma solução melhor com a classe `ArrayList`.

1. Copie o conteúdo do projeto `banco` do módulo anterior.
2. Adicione dois atributos na classe `Banco`: `clientes` (um array de objetos `Cliente`) e `numeroDeClientes`(um inteiro que armazena número de clientes do banco)
3. Adicione um construtor público que inicia o array de clientes com um tamanho máximo adequado (pelo menos de tamanho 5).
4. Adicione o método `adicionarCliente`. Este método deve construir um novo objeto `Cliente` através dos parâmetros (`nome`, `sobrenome`) e armazená-lo no *array*. Também deve incrementar o atributo `numeroDeClientes`.

5. Adicione o método de acesso `getNumeroDeClientes`, que retorna o atributo `numeroDeClientes`.
6. Adicione o método `getCliente`. Este método retorna o `cliente` associado com o dado parâmetro `indice`.
7. Crie uma classe com um método `main` chamado `TesteBanco`. No método `main` você deverá criar cinco clientes, criar uma conta para cada cliente, adicionar os clientes no banco no final exibir os clientes e seus saldos em conta, um a um, a partir de chamadas ao método `Cliente getCliente(indice i)` presente na classe `Banco` e método `getConta`, presente na classe `Cliente`. Ao executar o programa a seguinte saída deverá ser produzida:

```
Cliente [1] : Bruno Henrique. Saldo R$:50000.0
Cliente [2] : Everton Ribeiro. Saldo R$:45000.0
Cliente [3] : Filipe Luis. Saldo R$:70000.0
Cliente [4] : Gabriel Barbosa. Saldo R$:220000.0
Cliente [5] : Diego Alves. Saldo R$:50000.0
```

8. O que acontece se tentarmos incluir um sexto cliente?

Exercício 4 – Utilizando a classe ArrayList para Representar Multiplicidade

Objetivo do exercício – Neste exercício, evoluiremos nossa solucão do projeto banco para usar a classe ArrayList em vez de *arrays* para representar multiplicidade. Faremos isso, pois arrays são limitados para representar associações entre objetos tendo em vista que são de tamanho fixos sem poder ser redimensionados. Diferentemente de *arrays*, objetos do tipo `ArrayList` possuem tamanhos dinâmicos podendo crescer de tamanho na medida que novos elementes são inseridos.

Tarefas:

- Substitua o uso de *arrays* por `java.util.ArrayList` para representar a multiplicidade entre banco e clientes no conforme diagrama de classes do exercício 3. Abaixo, um trecho de como usar a classe `ArrayList`.

- Criar um objeto `ArrayList`:
  - `ArrayList<Cliente> clientes = new ArrayList<>();`

- Adicionar um objeto no `ArrayList`:
  - `clientes.add(new Cliente("Rodrigo", "Pagliares");`

- Retornar um objeto presente no objeto `ArrayList`:
  - `clientes.get(int indice)`
    - Por exemplo, `clientes.get(0)` para obter o cliente da primeira posição da lista.

- Retornar o número de objetos em um `ArrayList`:
  - `clientes.size();`
- Execute o programa `TesteBanco`. A saída deverá ser idêntica à do exercício anterior.

```
Cliente [1] : Bruno Henrique. Saldo R$:50000.0
Cliente [2] : Everton Ribeiro. Saldo R$:45000.0
Cliente [3] : Filipe Luis. Saldo R$:70000.0
Cliente [4] : Gabriel Barbosa. Saldo R$:220000.0
Cliente [5] : Diego Alves. Saldo R$:50000.0
```

Observe que após substituído o *array* por um objeto `ArrayList` poderemos excluir a variável `numeroDeClientes` usada para sabermos a posição de inserção de um cliente no *array* e todo código que a manipula.

Exercício 5 – Leitura de teclado em linha de comando – Projeto Banco

Nos Exercícios 4 e 5 deste laboratório, incluímos os dados dos clientes, saldos em conta, e valores das transações (*hard-coded*) dentro do método `main`. Neste exercício você vai aperfeiçoar este código, permitindo a leitura dos valores fornecidos via um programa em linha de comando em Java que leia do teclado.

Dica #1: O código-exemplo de leitura do teclado em Java (fornecido na aula #06 disponível no Moodle) tem todo o conteúdo necessário para a realização deste exercício:

- Aula #06 - Pacotes. Wrapper classes. Leitura via teclado para programas em linha de comando em Java

Alternativamente você poderá reutilizar sua solução de leitura de teclado desenvolvida no Laboratório anterior.

Dica #2: Neste e no próximo exercícios você pode assumir que os valores fornecidos pelo usuário via teclado estão corretos com relação a seus tipos e faixa de valores. Por exemplo, se for solicitada a entrada de um valor numérico, supõe-se que o usuário digita um número e não uma `String`, por exemplo. Em Laboratórios futuros iremos aperfeiçoar nosso código com tratamento de exceções e validação de dados.

Dica #3: Você pode usar os mesmos valores usados no exercício e deste laboratório para testar a sua versão com leitura via teclado.