Arrays e *ArrayLists*

Prof. Ítalo Assis

Ajude a melhorar este material =]

Encontrou um erro? Tem uma sugestão?

Envie e-mail para italo.assis@ufersa.edu.br

Agenda

- *Arrays* Unidimensionais;
- *Arrays* Multidimensionais;
- *ArrayList*.

Motivação

- Considere uma classe que deva encapsular 1440 valores de ponto flutuante correspondentes a medidas de temperatura obtidas a cada minuto em um dia inteiro em uma estação meteorológica.
  - Claramente usar uma variável para cada medida seria inviável
  - O problema é simplificado dando um nome único às variáveis e usando índices para identificar as diferentes medidas.
- Considere agora o problema de representar uma turma de alunos através de um conjunto de instâncias de uma classe *Aluno*.
  - Uma classe *Turma* poderia conter um número pré-definido e limitado de referências à instâncias de *Aluno*, mas novamente a declaração e manipulação de muitas referências independentemente seria complexa e poderia levar a erros.

Arrays

- *Arrays* são estruturas de dados que permitem o armazenamento de um conjunto de variáveis de um mesmo tipo ou instâncias de uma mesma classe usando uma única referência e um índice de acesso
- Cada um dos elementos do *array* pode ser acessado individualmente
- O *array* inteiro pode ser processado como uma única entidade caso seja desejado, simplificando bastante algumas tarefas

temp[0]	temp[1]	temp[2]	temp[3]
20.1	22.3	25.8	23.7

Arrays unidimensinais

- Declaração:
  - tipo[] nomeDaReferencia;
  - tipo nomeDaReferencia[];
- Exemplos:
  - *char[] letrasDoAlfabeto*;
  - *double medidasDeTemperatura[]*;
- Não basta somente declarar as referências a *arrays*, estas devem ser inicializadas

Arrays unidimensinais

- Um *array* deve ser inicializado com a palavra-chave *new*, seguida do tipo de dado a ser alocado e do número de elementos a alocar, entre colchetes

```
int[] posicoesDeMemoria = new int[1024];
int quantidadeNecessaria = 32768;
byte[] vetorNumerico = new byte[quantidadeNecessaria];
char[] letrasDoAlfabeto = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};
double[] medidasDeTemperatura;
medidasDeTemperatura = new double[24*60*60];
double[] duplicata = medidasDeTemperatura;
```

Arrays unidimensinais

- Após a inicialização, podemos acessar seus elementos usando o nome da referência seguida do índice do elemento entre colchetes
- O primeiro índice é o zero

```
int[] posicoesDeMemoria = new int[1024];
System.out.println(posicoesDeMemoria[0]); // correto
System.out.println(posicoesDeMemoria[1023]); // correto
System.out.println(posicoesDeMemoria[-1]); // errado
System.out.println(posicoesDeMemoria[1024]); // errado
System.out.println(posicoesDeMemoria[5.4]); // errado
```

Arrays unidimensinais

- Um *array*, depois de inicializado, não pode ter seu tamanho modificado
- Porém, é possível usar a referência para apontar para outro *array*

```
boolean[] respostas = new boolean[12];
respostas = new boolean[144];
```

  - os valores do primeiro *array* (de 12 posições) serão perdidos quando a referência apontar para o *array* de 144 posições
- Quando *arrays* de tipos nativos são inicializados, os elementos automaticamente recebem valores *default*

Arrays unidimensinais

- Os elementos de um *array* inicializado podem ser utilizados como variáveis independentes
  - *medidasDeTemperatura[5] = 45.8;*
- Geralmente um *array* será populado com um laço cuja variável de controle cobre os valores aceitáveis para índices do *array*, modificando o elemento naquele índice
  - Em Java, *Arrays* unidimensionais possuem um atributo *length* que armazena a sua quantidade de elementos

```
int[] naturais = new int[11];
for (int i = 0; i < naturais.length; i++) {
  naturais[i] = i;
  System.out.println("naturais[" + i + "] = " + naturais[i]);
}
```

Arrays de instâncias de classes

- Podem ser declaradas de forma similar à *arrays* de valores de tipos nativos
- A diferença principal é que a inicialização dos elementos do *array* deve ser feita através da palavra-chave *new* e da chamada ao construtor da classe.

  Funcionario[] equipe = new Funcionario[5];

  equipe[2] = new Funcionario("Davi Matias", 876451, 27, 12, 1940, 38200f);

Passando *arrays* para métodos

- Para passar um argumento de *array* para um método, especifique o nome do *array* sem nenhum colchete
  - *double[] temperaturaHora = new double[24];*
  - *modificaArray(temperaturaHora);*
- Para um método receber uma referência de *array*, a lista de parâmetros do método deve especificar um parâmetro de *array* incluindo tipo, colchetes e nome do parâmetro
  - *void modificaArray(double[] b)*

Prática

$$\frac{\pi^2}{6} = \sum_{i=1}^{i \leq n} \frac{1}{i^2}$$

- Vamos fazer um programa que calcule $\pi^2/6$ através da fórmula acima

  Para isso:

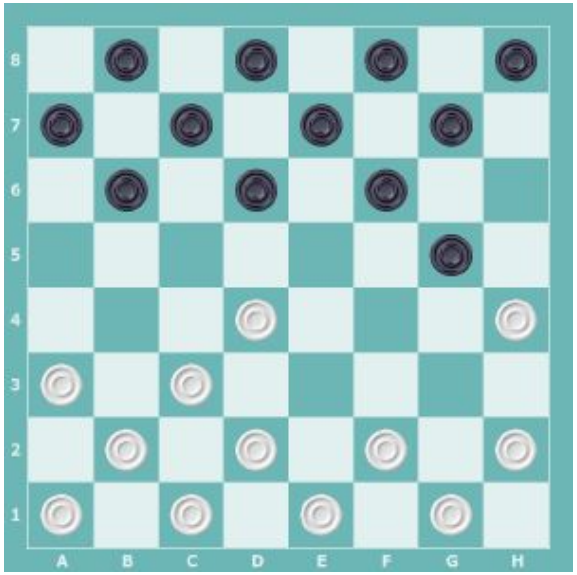
  - Escreva no método *main* o código para calcular os primeiros 1 milhão de elementos do somatório acima e armazená-los em um *array termosDaSerie*
  - Escreva também o método *calculaEMostraSomatoria* para calcular e imprimir a soma dos *num* primeiros elementos de um *array*
  - Por fim, utilize o método *main* para chamar o método *calculaEMostraSomatoria* passando o array *termosDaSerie* e crescentes valores de *num* (10, 100, ..., 1000000)
  - Os valores obtidos se aproximam da resposta exata (1.64493406685)?

Arrays multidimensionais

- Até agora vimos *arrays* de uma única dimensão: para acessar os elementos destes *arrays* basta usar um único índice
- Podemos criar *arrays* multidimensionais em Java, onde a posição de cada elemento dentro do *array* será indicada por dois ou mais índices
  - *char tabuleiro[][] = new char[8][8];*
  - *tabuleiro[5][4] = 'x';*
  - *int[][] b = {{1, 2}, {3, 4, 5}};*
- Um exemplo comum de *arrays* multidimensionais são matrizes matemáticas, que representam valores tabulados em linhas e colunas

Prática

- Escreva a classe *JogoDeDamas*, que encapsula um tabuleiro de jogo de damas usando um *array*. Casas vazias no tabuleiro são representadas pelo caracter '.' e casas com peças pelos caracteres 'o' e 'x'.
  - Esta versão do jogo não prevê a representação de damas
  - Implemente um método construtor e um método *toString* (retorna uma *String* que representa os atributos de uma instância)



Arrays irregulares

- Arrays multidimensionais em Java não precisam ter o mesmo número de valores para cada dimensão
- O Java trata *arrays* multidimensionais como *arrays* de *arrays*
- Somente o tamanho da primeira das dimensões precisa ser especificado - as outras dimensões devem ser alocadas em passos subsequentes do método
    - *int[][] b = {{1, 2}, {3, 4, 5}};*
    - *int[][] matriz = new int[5][];*
    - *matriz[3] = new int[4];*

matriz[0][]	7	1	5	8	3	6		
matriz[1][]	5	11	2					
matriz[2][]	7							
matriz[3][]	12	7	6	9				
matriz[4][]	6	9	4	5	5	10	4	1

Prática

- Escreva uma classe executável cujo método principal apresente o triângulo de pascal
  - O triângulo de Pascal é uma série de valores onde cada elemento pode ser calculado como a soma do elemento acima com o elemento acima e à esquerda
  - O primeiro e último elementos de cada linha do triângulo são sempre iguais a um
  - O triângulo de Pascal de cinco linhas é:

```
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
...
```

A instrução *for* aprimorada

- A instrução *for* aprimorada (*for each*) itera pelos elementos de um *array* sem usar um contador
- Não pode ser usada para modificar elementos

```
for ( parâmetro : nomeDoArray ) {
        instruções;
}
```

```
public class ForAprimorado {
        public static void main(String[] args) {
                int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
                int total = 0;
                for (int num : array) {
                        total += num;
                }
                System.out.printf("Soma dos elementos: %d%n", total);
        }
}
```

Listas de argumentos de comprimento variável

- Você pode criar métodos que recebem um número não especificado de argumentos
- Um tipo seguido por reticências (...) na lista de parâmetros de um método indica que o método recebe um número variável de argumentos desse tipo particular
  - *double media(int n, double... numeros)*
  - *numeros* é um *array* unidimensional com tamanho definido a partir da quantidade de parâmetros
- Só pode ocorrer uma vez e ao final de uma lista de parâmetros
  - *double media(int... pesos, double... numeros)*

Prática

- Em uma classe executável, escreva um método que receba uma quantidade variável de números *double* e calcule a média deles

Classe *Arrays*

- Fornece métodos estáticos para manipulações comuns de *array*
  - *sort*: organiza os elementos em ordem crescente
  - *binarySearch*: determina se um *array* contém um valor específico e, se contiver, onde o valor está localizado
  - *equals*: compara *arrays*
  - *fill*: insere valores em um *array*
- Outro método estático útil é o *arraycopy* da classe *System*
- Referência: https://www.tutorialspoint.com/java/util/java_util_arrays.htm

Coleções

- A Java API fornece várias estruturas de dados predefinidas, chamadas **coleções**, usadas para armazenar grupos de objetos relacionados
- Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados de forma **transparente**
- Isso reduz o **tempo de desenvolvimento** de aplicativos

ArrayList

- *Arrays* não mudam automaticamente o tamanho em tempo de execução
- Objetos da classe de coleção *ArrayList<T>* (pacote *java.util.ArrayList*) podem alterar dinamicamente seu tamanho para acomodar mais elementos
- O T representa um espaço reservado: ao declarar um novo *ArrayList*, substitua-o pelo tipo dos elementos que você deseja que o *ArrayList* armazene
    - *ArrayList<String> list;*
- Como vimos, classes com esse tipo de espaço reservado que podem ser usadas com qualquer tipo são chamadas **classes genéricas**

Classes genéricas

- Somente tipos não primitivos podem ser usados para declarar variáveis e criar objetos das classes genéricas
- Mas o Java fornece um mecanismo conhecido como *boxing*, que permite que valores primitivos sejam empacotados como objetos para uso com classes genéricas
- Exemplo:
  - *ArrayList<Integer> inteiros;*
    - declara *inteiros* como um *ArrayList* que só pode armazenar *Integers*
    - Ao inserir um valor *int* em um *ArrayList<Integer>*, o valor *int* é empacotado como um objeto *Integer*
    - Quando você obtém um objeto Integer de um *ArrayList<Integer>*, e então atribui o objeto a uma variável *int*, o valor *int* dentro do objeto é desempacotado
- Classes empacotadoras de tipo: *Boolean*, *Byte*, *Character*, *Double*, *Float*, *Integer*, *Long* e *Short*

ArrayList

- Alguns métodos úteis da classe *ArrayList<T>*

Método	Descrição
add	Adiciona um elemento ao final do *ArrayList*
clear	Remove todos os elementos do *ArrayList*
contains	Retorna *true* se o *ArrayList* contém o elemento especificado; caso contrário, retorna *false*
get	Retorna o elemento no índice especificado
indexOf	Retorna o índice da primeira ocorrência do elemento especificado no *ArrayList*
remove	Remove a primeira ocorrência do valor especificado ou o elemento no índice especificado
size	Retorna o número de elementos armazenados em *ArrayList*

ArrayList

- Inicialização de um *ArrayList*
  - *ArrayList<String> itens = new ArrayList<String>();*
  - *ArrayList<String> itens = new ArrayList<>();*
    - Cria um novo *ArrayList* vazio de *Strings* com uma capacidade inicial padrão de 10 elementos
    - A capacidade indica quantos itens o *ArrayList* pode armazenar sem crescer
    - Um *ArrayList* é implementado usando um *array* convencional
    - Para crescer, ele deve criar um array interno maior e copiar cada elemento

Prática

- Escreva uma classe para representar o extrato de uma conta bancária
- O extrato é composto por transações (depósitos e saques), além do saldo final. As possíveis operações sobre um extrato são a inicialização, a transação e a visualização
- Escreva uma classe executável com um menu onde o usuário pode escolher entre depositar, sacar, visualizar o extrato ou encerrar
- Após a escolha do usuário, o programa deve realizar a ação solicitada. Caso não tenha sido escolhida opção de encerrar, o menu deve ser apresentado novamente

Referências

SANTOS, R. **Introdução à programação orientada a objetos usando JAVA**. 2. ed. Rio de Janeiro: Campus, 2013. 336p.

DEITEL, Paul; DEITEL, Harvey. **Java: como programar**. 10. ed. São Paulo: Pearson Education do Brasil, 2017.