

Arrays

Roland Teodorowitsch

Fundamentos de Programação - Escola Politécnica - PUCRS

27 de outubro de 2022

Objetivos

- Coletar elementos usando *arrays*
- Aprender os algoritmos comuns para processamento de *arrays*
- Trabalhar com *arrays* bidimensionais

Conteúdos

- *Arrays*
- Algoritmos Comuns para *Arrays*
- Usando *Arrays* com Métodos
- Tópicos Especiais
- Solução de Problemas
 - Combinando Algoritmos
 - Usando Objetos Reais
- *Arrays* Bidimensionais
- Sumário
- Humor
- Tópicos complementares
- Referências

Arrays

- Um programa de computador frequentemente necessita armazenar uma lista de valores para então processá-los
- Por exemplo:
 - Cálculo de variância ou desvio padrão
 - Divisão das despesas de uma festa entre um grupo de amigos
 - etc.
- Se você tiver uma lista de valores (por exemplo, 32, 54, 67,5, 29, 35, 80, 115, 44,5, 100, 65), quantas variáveis seriam necessárias?
 - `double input1, input2, input3, ...`
- *Arrays* resolvem este problema
- Um *array* armazena sequências de valores do mesmo tipo

Declarando um *Array*

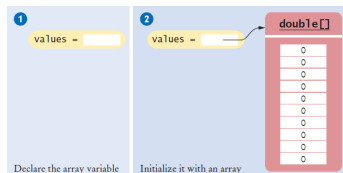
- Declarar um *array* envolve 2 etapas

- 1 Declarar a variável *array*

```
double[] values;
```

- 2 Inicializar o *array*

```
values = new double[10];
```



- O *array* não pode ser utilizado até que o compilador saiba qual o tamanho do *array* na etapa 2

Declarando um *Array* (Etapa 1)

- As seguintes partes devem ser especificadas

Type	Square Braces	Array name	semicolon
<code>double</code>	<code>[]</code>	<code>values</code>	<code>;</code>

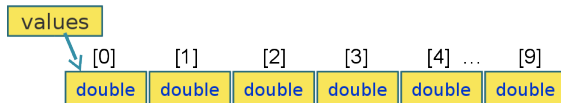
- Esta declaração especifica que
 - Há um *array* chamado `values`
 - Que os seus elementos são do tipo `double`
 - E que (AINDA) não foi definido quantos elementos ele poderá armazenar
- Outras considerações
 - Arrays* podem ser declarados em qualquer lugar onde também seja possível declarar uma variável
 - Não use palavras-reservadas ou nomes que já estejam em uso

Declarando um *Array* (Etapa 2)

- Reserva-se memória para todos os elementos

Array name		Keyword	Type	Size	semicolon
values	=	new	double	[10]	;

- Agora o compilador sabe que o *array* chamado `values` necessita de `[10]` elementos do tamanho do tipo `double`
- O *array* também está sendo inicializado: cada elemento do *array* recebe o valor 0
- Não se pode alterar o tamanho do *array* depois da sua declaração!



Declaração de um *Array* em uma Linha

- Declaração e criação em 1 linha

Type	Braces	Array name	Keyword	Type	Size	semi
<code>double</code>	<code>[]</code>	<code>values</code>	<code>=</code>	<code>new</code>	<code>double</code> <code>[10]</code>	<code>;</code>

- Está sendo feita a declaração de um *array* chamado `values` para armazenar elementos do tipo `double`
- Está sendo reservada memória para armazenamento de `[10]` elementos do tipo `double`
- Os elementos estão sendo inicializados com 0

Declaração e Inicialização de um *Array*

- Pode-se declarar e definir os conteúdos iniciais de todos os elementos de um *array* usando:

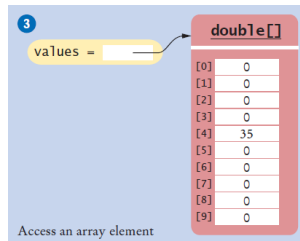
Type	Braces	Array name	contents list	semi
<code>int</code>	<code>[]</code>	<code>primes =</code>	<code>{ 2, 3, 5, 7 }</code>	<code>;</code>

- Está sendo declarado que:
 - O *array* `primes` conterá elementos do tipo `int`
 - Haverá espaço para 4 elementos (automaticamente contados pelo compilador) que conterão os seguintes valores iniciais: 2, 3, 5 e 7
 - O par de chaves determina uma lista de valores iniciais para o *array*

Acessando Elementos de um *Array*

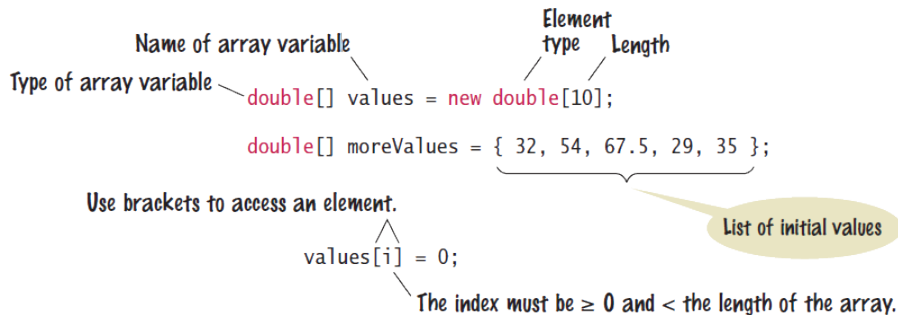
- Cada elemento de um *array* é numerado:
 - Este número é chamado de índice
 - Acessa-se um elemento especificando o nome do *array* e o índice numérico
- Elementos no *array* `values` são acessados por um índice inteiro `i`, usando a notação `values[i]`

```
public static void main(String[] args) {  
    double[] values;  
    values = new double[10];  
    values[4] = 35;  
}
```



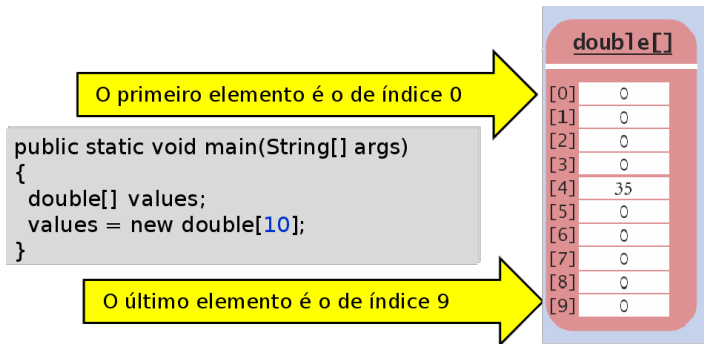
Sintaxe de *Arrays*

- Para declarar um *array*, especifique
 - O nome da variável *array*
 - O tipo de seus elementos
 - O tamanho (número de elementos)



Números de Índices de *Arrays*

- Números de índices de *arrays* iniciam em 0 e os demais são números inteiros positivos
- Um *array* com 10 elementos tem índices de 0 até 9: **NÃO há elemento 10**



Verificação de Limites de *Arrays*

- Um *array* sabe quantos elementos ele pode armazenar
 - `values.length` é o tamanho de um *array* chamado `values`
 - Trata-se de um valor inteiro que corresponde ao índice do último elemento + 1
- Usa-se isto para verificar e prevenir erros de acesso fora dos limites
- *Strings* e *arrays* usam sintaxes diferentes para encontrar os seus tamanhos
 - *Strings*: `name.length()`
 - *Arrays*: `values.length`

```
public static void main(String[] args) {  
    int i = 10, value = 34;  
    double[] values;  
    values = new double[10];  
    if (0 <= i && i < values.length) { // length is 10  
        values[i] = value;  
    }  
}
```

Exemplos de Declaração Arrays (1/2)

Declaração	Explicação
<pre>int[] numeros = new int[10];</pre>	Um <i>array</i> de 10 inteiros. Todos os elementos inicializados com zero.
<pre>final int TAMANHO = 10; int[] numeros = new int[TAMANHO];</pre>	É uma boa ideia usar uma constante, em vez de um número constante.
<pre>int tamanho = in.nextInt(); double[] dados = new double[tamanho];</pre>	O tamanho não precisa ser uma constante.

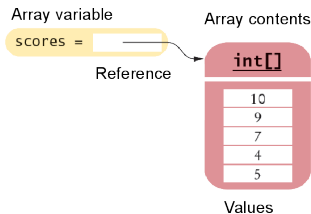
Exemplos de Declaração *Arrays* (2/2)

Declaração	Explicação
<code>int[] quadrados = { 0, 1, 4, 9, 16 };</code>	Um <i>array</i> de 5 inteiros, com valores iniciais.
<code>String[] amigos = { "Joao", "Maria", "Paulo" };</code>	Um <i>array</i> com 3 <i>strings</i> .
<code>double[] data = new int[10]; // ERRO</code>	ERRO! Não se pode inicializar uma variável <code>double[]</code> com um <i>array</i> do tipo <code>int[]</code> .

Referências a *Arrays*

- É preciso perceber que há uma diferença entre:
 - Variável *array*: chamada de “manipulador” do *array*
 - Conteúdos do *array*: memória onde os valores estão armazenados
- Uma variável *array* contém uma referência aos conteúdos do *array*
- A referência é a localização dos conteúdos do *array* (na memória)

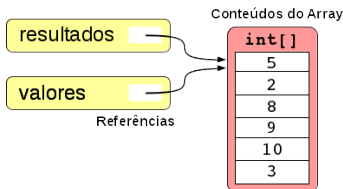
```
int[] scores = { 10, 9, 7, 4, 5 };
```



“Apelidos” para Arrays

- Pode-se fazer uma variável *array* referenciar os mesmos conteúdos de outra variável *array*
- Uma variável *array* especifica a localização de um *array*
- Copiar uma referência corresponde a se ter uma segunda referência para o mesmo conteúdo

```
int[] resultados = { 5, 2, 8, 9, 10, 3 };  
int[] valores = resultados; // Cópia da referência do array
```



Arrays Parcialmente Preenchidos

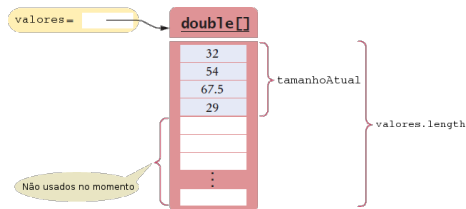
- Um *array* não pode ter o seu tamanho alterado durante a execução
 - O programador pode necessitar obter valores até o número máximo de elementos necessário
 - É uma boa ideia usar uma constante para o número máximo escolhido
 - Usa-se uma variável (`tamanhoAtual` no exemplo a seguir) para controlar quantos elementos já foram obtidos

```
final int TAMANHO = 100;
double[] valores = new double[TAMANHO];
int tamanhoAtual = 0;
Scanner in = new Scanner(System.in);
while (in.hasNextDouble()) {
    if (tamanhoAtual < valores.length) {
        valores[tamanhoAtual] = in.nextDouble();
        tamanhoAtual++;
    }
}
```

Percorrendo um *Array* Parcialmente Preenchido

- No exemplo, usa-se `tamanhoAtual` (e não `valores.length`) para determinar qual o último elemento
- Um laço `for` é uma escolha natural para percorrer um *array*

```
for (int i = 0; i < tamanhoAtual; i++) {  
    System.out.println(valores[i]);  
}
```



Erros Comuns (1)

Erro nos limites de *arrays*

- Índices de *arrays* iniciam em 0 e terminam em tamanho - 1
- Acessar um elemento que não existe é um erro bastante comum
- Resultado: exceção lançada em tempo de execução
(`java.lang.ArrayIndexOutOfBoundsException`)

```
public class OutOfBounds {  
    public static void main(String[] args) {  
        double[] values;  
        values = new double[10];  
        values[10] = 100; // ERRO=EXCECAO  
    }  
}
```

Erros Comuns (2)

Arrays não inicializados

- Não se esqueça de inicializar variáveis *array*
- O compilador vai gerar um erro como “variable values might not have been initialized”

```
double[] values;  
...  
values[0] = 29.95; // ERRO!
```

```
double[] values;  
values = new double[10];  
values[0] = 29.95; // Sem erro!
```

Algoritmos Comuns para Arrays

- Preencher um *array*
- Soma e média de valores
- Encontrar máximo e mínimo
- Saída de elementos com separadores
- Busca linear
- Remoção de um elemento
- Inserção de um elemento
- Troca de elementos
- Cópia de *arrays*
- Aumento de tamanho de *arrays*
- Leitura da entrada

Preencher um *Array*

- Inicializar um *array* com um conjunto de valores calculados
- Por exemplo: preencher um *array* com os quadrados de 0 até 10

```
int[] quadrados = new int[11];  
for (int i = 0; i < quadrados.length; i++) {  
    quadrados[i] = i * i;  
}
```

Soma e Média de Valores

- Nunca se esqueça de evitar a divisão por zero

```
if (valores.length > 0) {  
    double total = 0, media = 0;  
    for (int i=0; i<valores.length; ++i)  
        total = total + valores[i];  
    media = total / valores.length;  
}
```


Encontrar máximo e mínimo

- Defina que o maior ou menor é o primeiro elemento e teste os outros elementos usando `for`

```
// Laco tipico para encontrar o maximo
double maior = valores[0];
for (int i = 1; i < valores.length; i++)
    if (valores[i] > maior)
        maior = valores[i];
```

```
// Laco tipico para encontrar o minimo
double menor = valores[0];
for (int i = 1; i < valores.length; i++)
    if (valores[i] < menor)
        menor = valores[i];
```

- Às vezes também é desejável localizar o índice do maior ou menor elemento

```
// Localizar o indice do maior
int iMaior = 0;
for (int i = 1; i < valores.length; i++)
    if (valores[i] > valores[iMaior])
        iMaior = i;
```

```
// Localizar o indice do menor
int iMenor = 0;
for (int i = 1; i < valores.length; i++)
    if (valores[i] < valores[iMenor])
        iMenor = i;
```

Saída de Elementos com Separadores

- Imprime-se o separador antes de todos os elementos, com exceção do primeiro

```
double[] valores = {32, 54, 67.5, 29, 35};  
for (int i = 0; i < valores.length; i++) {  
    if (i > 0) {  
        System.out.print(" | ");  
    }  
    System.out.print(valores[i]);  
}
```

- Resultado:

32 | 54 | 67.5 | 29 | 35

- Ou usa-se o método `Arrays.toString()` (muito útil para depuração)

```
import java.util.*;  
// ...  
double[] valores = {32, 54, 67.5, 29, 35};  
System.out.println(Arrays.toString(valores));
```

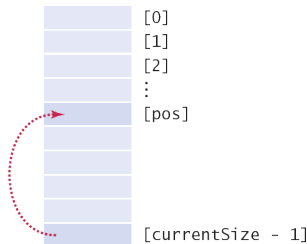
Busca Linear

- Busca-se um valor específico em um *array*
- Inicia-se pelo primeiro elemento e para-se quando/se o valor for encontrado
- Usa-se uma variável booleana `achou` para controlar o final do laço

```
int valorBuscado = 100; int pos = 0;
boolean achou = false;
while (pos < valores.length && !achou) {
    if (valores[pos] == valorBuscado) {
        achou = true;
    }
    else {
        pos++;
    }
}
if (achou)
    System.out.println("Encontrado na posicao: " + pos);
else
    System.out.println("Nao encontrado");
```

Remoção de um Elemento

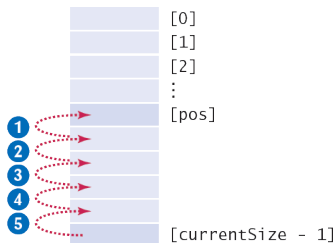
- Exige que se mantenha uma variável com o tamanho atual (número de elementos válidos)
- Não se pode deixar um “buraco” no *array*
- Se NÃO é preciso manter o *array* ordenado: copie o último elemento sobre o elemento atual e atualize o tamanho atual



```
if ( pos >= 0 && pos <= valores.length - 1 ) {  
    if ( tamanhoAtual > 1 ) {  
        valores[pos] = valores[tamanhoAtual - 1];  
    }  
    tamanhoAtual--;  
}
```

Remoção de um Elemento (Continuação)

- Se é preciso manter o *array* **ordenado**: mova todos os elementos que estão após *pos* uma posição (em direção ao início do *array*) e atualize o tamanho atual



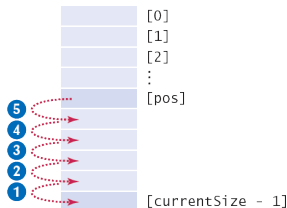
```
if ( pos >= 0 && pos <= valores.length-1 ) {  
    for (int i = pos; i < tamanhoAtual - 1; i++) {  
        valores[i] = valores[i + 1];  
    }  
    tamanhoAtual--;  
}
```

Inserção de um Elemento

- Se não é preciso manter a ordenação, apenas adiciona-se o novo valor no final e atualiza-se o tamanho

```
if ( tamanhoAtual < valores.length ) {
    valores[tamanhoAtual] = novoValor;
    tamanhoAtual++;
}
```

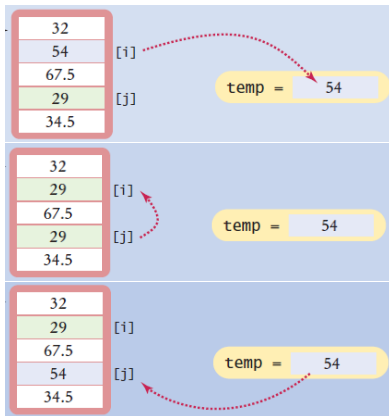
- Se é preciso manter a ordem, localiza-se a posição correta para o novo elemento, move-se todos os elementos válidos uma posição (em direção ao final do *array*) e atualiza-se o tamanho



```
if (tamanhoAtual < valores.length) {
    tamanhoAtual++;
    for (int i = tamanhoAtual - 1; i > pos; i--) {
        valores[i] = valores[i - 1]; // move p/ inicio
    }
    valores[pos] = novoValor; // preenche buraco
}
```

Troca de Elementos

- São usados 3 passos com uma variável auxiliar

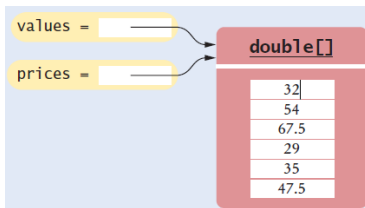


```
double temp = valores[i];  
valores[i] = valores[j];  
valores[j] = temp;
```

Cópia de *Arrays*

- Copiar *arrays* não é a mesma coisa que copiar apenas a referência
 - A cópia de *arrays* cria 2 conjuntos de conteúdos
 - Exemplo de cópia de referência:

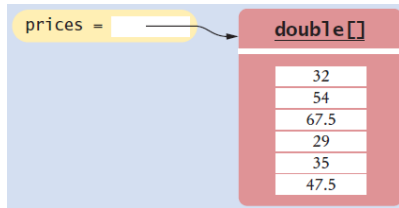
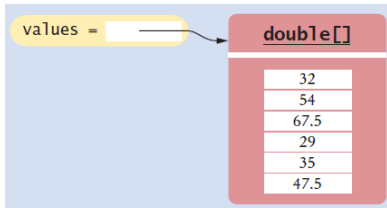
```
double[] values = { 32, 54, 67.5, 29, 35, 47.5 };  
// Cópia de referencia  
double[] prices = values;
```



Cópia de *Arrays* (2)

- Pode-se usar o método `Arrays.copyOf` (Java 6):

```
double[] values = { 32, 54, 67.5, 29, 35, 47.5 };  
// copyOf cria uma nova copia, retornando a referencia  
double[] prices = Arrays.copyOf(values, values.length);
```



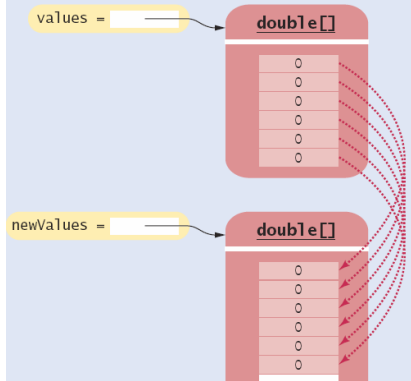
Aumento de Tamanho de *Arrays*

- Não é possível mudar o atributo correspondente ao tamanho de um *array*, mas é possível criar outra área de memória com `copyOf` e fazer a variável *array* apontar para ela
- Por exemplo, para duplicar o tamanho de um *array* existente:

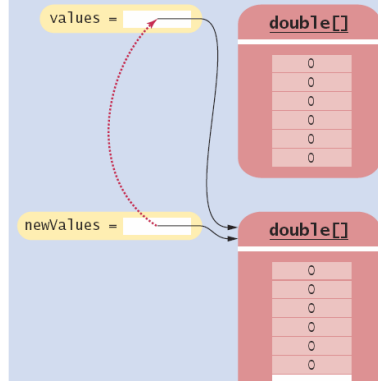
```
double[] values = { 32, 54, 67.5, 29, 35, 47.5 };  
double[] newValues = Arrays.copyOf(values, 2 * values.length);  
values = newValues;
```

Aumento de Tamanho de *Arrays* (2)

1 Move elements to a larger array



2 Store the reference to the larger array in `values`



Leitura da Entrada

- Sabendo-se previamente o tamanho exato do *array*:

```
double[] entradas = new double[NUM_ENTRADAS];  
for (i = 0; i < entradas.length; i++) {  
    entradas[i] = in.nextDouble();  
}
```

- Sem saber o tamanho exato, pode-se usar um tamanho máximo, mantendo-se o *array* parcialmente preenchido:

```
double[] entradas = new double[MAX_ENTRADAS];  
int tamanhoAtual = 0;  
while (in.hasNextDouble() && tamanhoAtual < entradas.length) {  
    entradas[tamanhoAtual] = in.nextDouble();  
    tamanhoAtual++;  
}
```

LargestInArray.java (HORSTMANN, 2013, p. 265-266)

```
import java.util.Scanner;

/**
 * This program reads a sequence of values and prints them, marking the largest value.
 */
public class LargestInArray {
    public static void main(String[] args) {
        final int LENGTH = 100;
        double[] values = new double[LENGTH];
        int currentSize = 0;
        // Read inputs
        System.out.println("Please enter values, Q to quit:");
        Scanner in = new Scanner(System.in);
        while (in.hasNextDouble() && currentSize < values.length) {
            values[currentSize] = in.nextDouble();
            currentSize++;
        }
        // Find the largest value
        double largest = values[0];
        for (int i = 1; i < currentSize; i++) {
            if (values[i] > largest) {
                largest = values[i];
            }
        }
    }
}
```

LargestInArray.java (HORSTMANN, 2013, p. 265-266)

```
// Print all values, marking the largest
for (int i = 0; i < currentSize; i++) {
    System.out.print(values[i]);
    if (values[i] == largest) {
        System.out.print(" <== largest value");
    }
    System.out.println();
}
}
```

Resultado da execução:

Please enter values, Q to quit:

35 80 115 44.5 Q

35

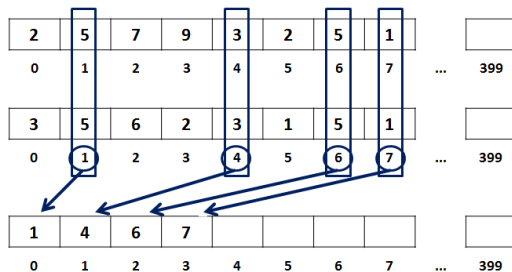
80

115 <== largest value

44.5

Exercícios (1/2)

- Construa um programa em Java que leia dados para dois vetores do tipo inteiro de 400 posições cada (`vetorA` e `vetorB`) e preencha um terceiro vetor (`vetorRes`), também de 400 posições e também do tipo inteiro, da seguinte maneira: quando o conteúdo dos elementos de mesma posição dos vetores `vetorA` e `vetorB` for igual, a posição (índice) destes elementos deve ser armazenada no vetor `vetorRes`, conforme ilustração abaixo. Devem ser utilizadas posições consecutivas de `vetorRes` para inserção dos valores desejados e, ao final do programa, devem ser escritas somente as posições preenchidas de `vetorRes`. [Adaptado do material da professora Milene Selbach Silveira]



Exercícios (2/2)

- Considere um *array* parcialmente preenchido

```
double[] valores = new double[100];
```

com valores até

```
int tamAtual;
```

E escreva um trecho de programa em Java para eliminar todos os valores zero deste *array*, sem criar um novo *array*.

- Considere um *array* de inteiros chamado `valores` e escreva um programa em Java para inverter este *array* (ou seja, troca o primeiro elemento pelo último, o segundo pelo penúltimo, e assim sucessivamente).

Nunca esquecer

- Subestimar o tamanho de um conjunto de dados é um erro comum
 - O programador não pode prever como as pessoas usarão o seu programa
 - Deve-se ter o cuidado escrever código que rejeita o excesso de dados na entrada de forma elegante, principalmente quando se usa tamanhos fixos

Usando *Arrays* com Métodos

- Métodos podem ser declarados para receber referências a *arrays* como variáveis paramétricas
- Isto permite declarar, por exemplo, um método para somar todos os elementos de um vetor: o método recebe a referência ao *array* e consegue somar os valores dos elementos de qualquer *array* (do mesmo tipo, é claro)
 - Método:

```
public static double sum(double[] values) {  
    double total = 0;  
    for (int i=0; i<values.length; ++i)  
        total = total + values[i];  
    return total;  
}
```

- Chamada:

```
double[] scores = { 32, 54, 67.5, 29, 35, 47.5 };  
double scoresTotal = sum(scores);
```

Passando Referências

- Quando se passa uma referência, o método pode **alterar** os valores do *array*
- Exemplo: multiplicar cada elemento de um *array* por um fator
 - Método:

```
public static void multiply(double[] values, double factor) {  
    for (int i = 0; i < values.length; i++)  
        values[i] = values[i] * factor;  
}
```

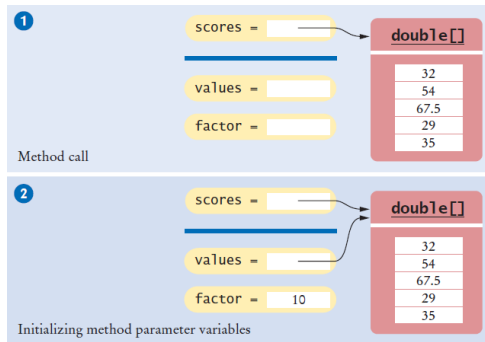
- Chamada:

```
double[] scores = { 32, 54, 67.5, 29, 35, 47.5 };  
multiply(scores, 10);
```

- Em Java, a passagem de parâmetros é sempre por valor (cópia do valor da chamada para a variável paramétrica), mas quando se passa uma referência (o que é o caso de *arrays*) é possível alterar os conteúdos desta referência (mas nunca a referência propriamente dita)

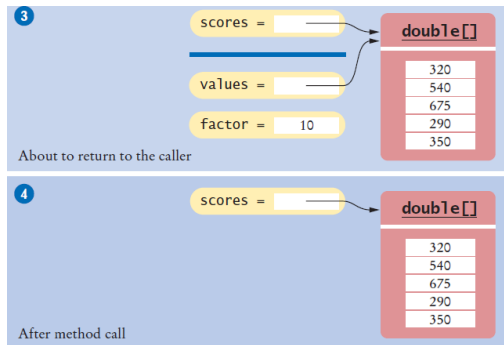
Passagem por Referência (1)

- As variáveis paramétricas são inicializadas com os argumentos que são passados na chamada. Neste exemplo, `values` recebe a mesma referência que `scores` e `factor` é inicializado com 10



Passagem por Referência (2)

- O método multiplica todos os elementos por 10
- O método retorna e suas variáveis paramétricas são destruídas. Entretanto, os valores do *array* permanecem alterados



Método Retornando um *Array*

- Métodos podem ser declarados para retornar um *array*

```
public static int[] squares(int n) {  
    int[] result = new int[n];  
    for (int i = 0; i < n; i++) {  
        result[i] = i * i;  
    }  
    return result;  
}
```

- Para chamar o método, deve-se criar uma referência a *array* que seja compatível

```
int[] numbers = squares(10);
```

Exercício

- Implemente uma classe chamada `Vetor` com uma biblioteca de métodos para processamento de vetores de inteiros
- Nesta classe procure implementar todos os algoritmos citados até aqui
- Os algoritmos que ainda serão vistos poderão ser incorporados posteriormente
- Use sobrecarga de métodos para permitir que cada método, quando possível, seja executado sobre um vetor completo ou sobre um vetor parcialmente preenchido

```
public class Vetor { // ESBOÇO
    public static void preenche(int[] v, int n) {}
    public static long soma(int[] v) {}
    public static double media(int[] v) {}
    public static int menor(int[] v) {}
    public static int maior(int[] v) {}
    public static boolean busca(int[] v, int n) {}
    public static int conta(int[] v, int n) {}
    public static void le(Scanner in, int [] v) {}
    public static void mostra(int [] v) {}
}

public static void preenche(int[] v, int t, int n) {}
public static long soma(int[] v, int t) {}
public static double media(int[] v, int t) {}
public static int menor(int[] v, int t) {}
public static int maior(int[] v, int t) {}
public static boolean busca(int[] v, int t, int n) {}
public static int conta(int[] v, int t, int n) {}
public static void le(Scanner in, int [] v, int t) {}
public static void mostra(int [] v, int t) {}
```

Tópico Especial: Ordenação de *Arrays*

- Quando se armazena valores em um *array*, pode-se:

- Manter os dados desordenados (ordem aleatória)

[0][1][2][3][4]

11 9 17 5 12

- Ordená-los (de forma ascendente ou descendente)

[0][1][2][3][4]

5 9 11 12 17

- Em um *array* ordenado é muito mais fácil encontrar um valor
- A Java API provê um método de ordenação eficiente:

```
Arrays.sort(values);           // Ordena todo o array  
Arrays.sort(values, 0, currentSize); // Parcialm. preen.
```


Tópico Especial: Algoritmos de Ordenação

- Alguns dos algoritmos de ordenação mais conhecidos são:
 - *Bubble Sort*: percorre os dados comparando os elementos da posição i com a posição $i+1$, e, se o primeiro for maior do que o segundo, inverte eles; simples; pouco eficiente
 - *Selection Sort*: procura o menor elemento e coloca na primeira posição, depois procura o segundo menor e coloca na segunda posição, e assim sucessivamente
 - *Insertion Sort*: percorre os dados da segunda posição até o final, procurando onde o elemento se encaixaria no conjunto à esquerda da posição atual (que já está ordenado); é simples e eficiente quando aplicado a pequenas listas
 - *Quick Sort*: escolhe um pivô e organiza os dados de forma que à esquerda do pivô estejam todos os valores menores do que ele, e à direita, todos os valores maiores do que o pivô, organizando a seguir as duas metades (recursivamente); é considerado o algoritmo de ordenação mais eficiente

Tópico Especial: Pesquisa

- Já vimos um método de **pesquisa linear**
 - Ele funciona sobre *arrays* ordenados ou desordenados
 - Cada elemento é visitado, partindo do início, até que o valor procurado seja encontrado ou até que se chegue ao fim do *array*

Pesquisa Binária

- Só funciona se o *array* estiver ordenado
- Compara o valor procurado com o elemento do meio
 - Se for igual, encontrou
 - Se for menor, desconsidera a metade superior
 - Se for maior, desconsidera a metade inferior
- Repete-se o procedimento até que o valor procurado seja encontrado ou até que não se consiga dividir o *array*

Pesquisa Binária: Exemplo

- Encontrar o valor 15

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	5	8	9	12	17	20	32

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	5	8	9	12	17	20	32

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	5	8	9	12	17	20	32

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
1	5	8	9	12	17	20	32

Sorry, 15 is not in this array.

Pesquisa Binária: Implementação

```
boolean found = false;
int pos = 0, low = 0, high = values.length - 1;

while (low <= high && !found) {
    pos = (low + high) / 2; // Ponto central
    if (values[pos] == searchedValue)
        found = true; // Encontrou!
    else if (values[pos] < searchedValue)
        low = pos + 1; // Busca na primeira metade
    else
        high = pos - 1; // Busca na segunda metade
}
if (found)
    System.out.println("Found at position " + pos);
else
    System.out.println("Not found. Insert before position " + pos);
```

Pesquisa Binária: Exercício

Considerando o teste a seguir, monte uma tabela com colunas para cada uma das variáveis envolvidas, apresentando nas linhas os valores que estas variáveis assumem ao longo da execução. Mostre também a saída gerada. Faça 4 execuções, para `valorProc` igual a 2, 5, 18 e 20.

```
boolean achou = false;
int[] valores = { 3, 5, 8, 9, 15, 16, 17, 19};
int pos = 0, inf = 0, sup = valores.length - 1;

int valorProc = in.nextInt();
while (inf <= sup && !achou) {
    pos = (inf + sup) / 2;
    if (valores[pos] == valorProc)
        achou = true;
    else if (valores[pos] < valorProc)
        inf = pos + 1;
    else
        sup = pos - 1;
}
if (achou)
    System.out.println("pos = " + pos);
else
    System.out.println("NAO encontrado");
```

Combinando Algoritmos

- Considere o seguinte problema: o cálculo do escore de um aluno num questionário é a soma dos pontos em cada questão sem contar o menor valor.
- Por exemplo, para os seguintes valores:

8 7 8.5 9.5 7 5 10

o escore final será 50.

Abordagem

- Decompor a tarefa em passos:
 - Ler as entradas
 - Calcular o somatório
 - Encontrar o menor valor
 - Subtrair o menor valor do somatório
- Determinar algoritmos para cada passo e implementá-los em métodos
- Montar a solução:

```
double[] scores = readInputs();  
double total = sum(scores) - minimum(scores);  
System.out.println("Final score: " + total);
```

- Revisar o código tratando soluções especiais: dados do enunciado, duas notas com o menor valor (apenas uma deve ser desconsiderada), uma única nota (score deve ser zero), nenhuma entrada, etc.

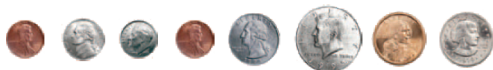
Usando Objetos Reais

- Considere o seguinte problema: você tem um *array* cujo tamanho é par, e você deve trocar a primeira metade com a segunda metade
- Por exemplo, para um *array* de 8 posições

9	13	21	4	11	7	1	3
---	----	----	---	----	---	---	---

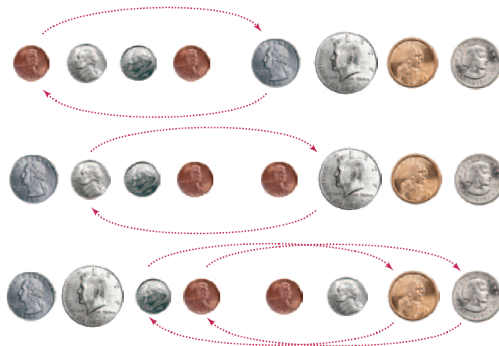
11	7	1	3	9	13	21	4
----	---	---	---	---	----	----	---

- Uma técnica bastante útil para ajudar no desenvolvimento de algoritmos é usar objetos reais
- Pode-se, por exemplo, usar uma linha de objetos (moedas, cartas, pequenos brinquedos, peças de um jogo, etc.) para representar um *array* e assim estudar operações necessárias na implementação de um algoritmo



Manipulando Objetos Reais

- Para resolver o problema proposto, os movimentos (operações) seriam::



Criando o Algoritmo

- É importante identificar:
 - Quantas trocas serão necessárias?
 - Quais os índices dos elementos que deverão ser trocados?
- Estas informações devem estar relacionadas ao tamanho do *array*
- Algoritmo:

```
i = 0  
j = size / 2  
While (i < size / 2)  
    Swap elements at positions i and j  
    i++  
    j++
```

Exercício

Escreva um método em Java que troque a primeira metade de um *array* de inteiros pela sua segunda metade. Caso o tamanho do *array* seja ímpar, o método não deve fazer nada com o *array*.

Exercício (Solução)

Escreva um método em Java que troque a primeira metade de um *array* de inteiros pela sua segunda metade. Caso o tamanho do *array* seja ímpar, o método não deve fazer nada com o *array*.

```
public static void trocaMetades(int[] vetor) {  
    int tam = vetor.length;  
    if ( tam % 2 == 0 ) {  
        tam = tam / 2;  
        int j = tam;  
        for (int i=0; i < tam; ++i) {  
            int aux = vetor[i];  
            vetor[i] = vetor[j];  
            vetor[j] = aux;  
            j++;  
        }  
    }  
}
```

Arrays Bidimensionais

- Arrays também podem ser usados para armazenar dados em duas dimensões, como dados de uma tabela
- Neste caso tem-se uma matriz com linhas e colunas
- A declaração é feita com dois pares de colchetes:
 - Usando `new`:

```
const int COUNTRIES = 7;  
const int MEDALS = 3;  
int[][] counts = new int[COUNTRIES][MEDALS];
```

- Usando valores inicializados e chaves (neste caso com dois “níveis” de chaves):

```
int[][] counts = {  
    { 0, 0, 1 },  
    { 1, 0, 0 },  
    { 0, 1, 1 },  
    { 0, 1, 1 },  
    { 1, 1, 0 }  
};
```

Sintaxe de *Arrays* Bidimensionais

Name Element type Number of rows Number of columns
`double[][] tableEntries = new double[7][3];`

All values are initialized with 0.

Name
`int[][] data = {`
 `{ 16, 3, 2, 13 },`
 `{ 5, 10, 11, 8 },`
 `{ 9, 6, 7, 12 },`
 `{ 4, 15, 14, 1 }`
 `};`

List of initial values

- O nome do *array* continua a ser a referência para os conteúdos do *array*

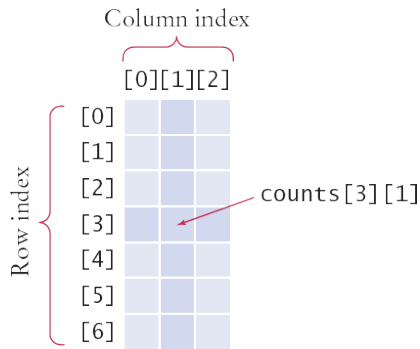
Acessando Elementos

- Usa-se dois valores de índice: linha e coluna

```
int value = counts[3][1];
```

- Para imprimir são usados 2 laços aninhados: o mais externo para linhas (i) e o mais interno para colunas (j)

```
for (int i = 0; i < COUNTRIES; i++) {  
    // Process the ith row  
    for (int j = 0; j < MEDALS; j++) {  
        // Process the jth column in the ith row  
        System.out.printf("%8d", counts[i][j]);  
    }  
    // Start a new line at the end of the row  
    System.out.println();  
}
```



Localizando Elementos Vizinhos

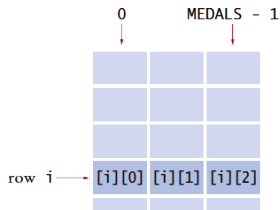
- Alguns programas que trabalham com *arrays* bidimensionais necessitam localizar elementos que estão em posições adjacentes, o que é muito comum em jogos
- Na posição $[i][j]$, os vizinhos são:

$[i - 1][j - 1]$	$[i - 1][j]$	$[i - 1][j + 1]$
$[i][j - 1]$	$[i][j]$	$[i][j + 1]$
$[i + 1][j - 1]$	$[i + 1][j]$	$[i + 1][j + 1]$

- Cuidado com cantos e bordas, pois não há índices negativos (posições fora do tabuleiro, por exemplo)

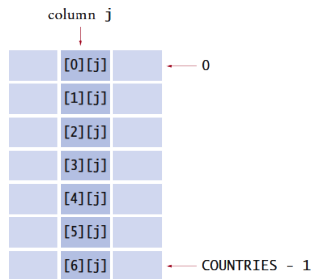
Somando Linhas e Colunas

• Soma linha i



```
int total = 0;
for (int j = 0; j < MEDALS; j++) {
    total = total + counts[i][j];
}
```

• Soma coluna j



```
int total = 0;
for (int i = 0; i < COUNTRIES; i++) {
    total = total + counts[i][j];
}
```

Medals.java (HORSTMANN, 2013, p. 286-287)

```
/**
 * This program prints a table of medal winner counts with row totals.
 */
public class Medals {
    public static void main(String[] args) {
        final int COUNTRIES = 7;
        final int MEDALS = 3;
        String[] countries = {
            "Canada",
            "China",
            "Germany",
            "Korea",
            "Japan",
            "Russia",
            "United States"
        };
        int[][] counts = {
            { 1, 0, 1 },
            { 1, 1, 0 },
            { 0, 0, 1 },
            { 1, 0, 0 },
            { 0, 1, 1 },
            { 0, 1, 1 },
            { 1, 1, 0 }
        };
    }
}
```

Medals.java (HORSTMANN, 2013, p. 286-287)

```

System.out.println("      Country    Gold  Silver  Bronze  Total");
// Print countries, counts, and row totals
for (int i = 0; i < COUNTRIES; i++) {
    // Process the ith row
    System.out.printf("%15s", countries[i]);
    int total = 0;
    // Print each row element and update the row total
    for (int j = 0; j < MEDALS; j++) {
        System.out.printf("%8d", counts[i][j]);
        total = total + counts[i][j];
    }
    // Display the row total and print a new line
    System.out.printf("%8d\n", total);
}
}
}

```

Resultado da execução:

Country	Gold	Silver	Bronze	Total
Canada	1	0	1	2
China	1	1	0	2
Germany	0	0	1	1
Korea	1	0	0	1
Japan	0	1	1	2
Russia	0	1	1	2
United States	1	1	0	2

Dica

- Se um *array* tem uma única dimensão, `length` corresponde ao tamanho desta dimensão
- E se *array* for bidimensional? Qual o valor de `length`? Como obter o tamanho da segunda dimensão?
 - Para *arrays* bidimensionais, `length` retorna o número de linhas
 - Para obter o número de colunas de uma linha, deve-se usar o `length` para a linha

```
int mat[][] = new int[2][3];  
System.out.println(mat.length); // 2  
System.out.println(mat[0].length); // 3  
System.out.println(mat[1].length); // 3
```

Dica avançada

- Pode-se criar um *array* com dimensões “irregulares” (cada linha com um número variável de elementos)

```
int mat[][] = new int[2][];  
mat[0] = new int[3];  
mat[1] = new int[4];  
System.out.println(mat.length);    // 2  
System.out.println(mat[0].length); // 3  
System.out.println(mat[1].length); // 4
```

Exercício 1

Considere matrizes de inteiros com l linhas e c colunas e implemente métodos para:

- a) Ler todos os elementos da matriz
- b) Escrever todos os elementos da matriz
- c) Trocar a linha i_1 pela linha i_2 da matriz
- d) Trocar a coluna c_1 pela coluna c_2 da matriz
- e) Inicializar todos os elementos da matriz com valores aleatórios

Exercício 2

Considere matrizes quadradas de inteiros e implemente métodos para calcular:

- a) Somatório de todos os elementos da matriz
- b) Somatório dos elementos da linha i da matriz
- c) Somatório dos elementos da coluna j da matriz
- d) Somatório dos elementos da diagonal principal da matriz
- e) Somatório dos elementos acima da diagonal principal da matriz
- f) Somatório dos elementos abaixo da diagonal principal da matriz
- g) Somatório dos elementos da diagonal secundária da matriz
- h) Somatório dos elementos acima da diagonal secundária da matriz
- i) Somatório dos elementos abaixo da diagonal secundária da matriz

Sumário: *Arrays*

- Um *array* armazena uma sequência de valores do mesmo tipo
- Elementos individuais em um *array* são acessados por um índice inteiro i , usando a notação `values[i]`
- Um elemento de *array* pode ser usado como qualquer outra variável
- Um índice de *array* deve ser no mínimo 0 e menor do que o tamanho do *array*
- Um erro de limite, que ocorre se você não especificar um índice válido de *array*, pode abortar a execução do programa
- Use a expressão `array.length` para encontrar o número de elementos de um *array*
- Uma referência de *array* armazena o endereço do local do conteúdo do *array*
- Copiar uma referência cria uma segunda referência para o mesmo *array*
- Com *arrays* parcialmente preenchidos, deve-se manter uma variável auxiliar com o tamanho atual do *array*

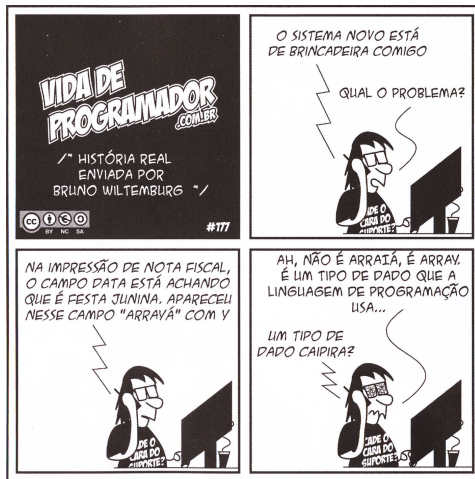
Sumário: *Arrays*

- Uma pesquisa linear inspeciona os elementos em sequência até encontrar o elemento procurado
- Usa-se uma variável temporária quando elementos devem ser trocados
- Usa-se o método `Arrays.copyOf` para copiar os elementos de um *array* para um novo *array*
- *Arrays* podem ser usados como variáveis paramétricas de métodos e podem ser retornados por métodos

Sumário: *Arrays*

- Pode-se resolver tarefas complexas combinando algoritmos básicos
- Deve-se conhecer os algoritmos básicos para reaproveitá-los e, quando necessário, adaptá-los
- Pode-se usar objetos reais para auxiliar no desenvolvimento de algoritmos
- Usa-se *arrays* bidimensionais (matrizes) para armazenar dados de tabelas
- Elementos individuais de um *array* bidimensional são acessados usando dois valores de índice: `values[i][j]`

Vida de Programador (NOEL, 2016, p. 37)



O Laço `for` Abreviado

- Usar laços `for` para percorrer um *array* é bastante comum:
 - O `for` abreviado simplifica este processo
 - Este laço também é chamado de *for each* (“para cada ”)
 - Este código pode ser lido como: “Para cada elemento do *array*”
- À medida que o laço avança, ele:
 - Acessará cada elemento sequencialmente (de 0 até o tamanho -1)
 - Copiará o seu valor para a variável de indução
 - Executará o corpo do laço
- Não é possível para:
 - Alterar elementos
 - Obter erros de limite

```
double[] values = ...;
double total = 0;
for (double element : values) {
    total = total + element;
}
```

Sintaxe do `for` abreviado

- Use o `for` abreviado quando:
 - For necessário acessar cada elemento de um *array*
 - Não for necessário alterar qualquer elemento do *array*

This variable is set in each loop iteration.
It is only defined inside the loop.

An array

```
for (double element : values)  
{  
    sum = sum + element;  
}
```

These statements
are executed for each
element.

The variable
contains an element,
not an index.

Referências

HORSTMANN, C. **Java for Everyone – Late Object**. 2. ed. Hoboken: Wiley, 2013. xxxiv, 589 p.

NOEL, Andre. **Vida de Programador.com.br - Volume 1**. São Paulo: Novatec, 2016. 118 p.