

JAVA: ARRANJOS, LAÇOS DE REPETIÇÃO E COMANDOS BREAK E CONTINUE

Programação Orientada a Objetos – POOS3

Prof. Dr. Lucas Bueno R. Oliveira



INSTITUTO FEDERAL
São Paulo
Câmpus São Carlos

O QUE JÁ VIMOS NAS AULAS ANTERIORES

Uma breve introdução ao IntelliJ IDEA.

Uma breve introdução à plataforma Java.

Tipos de dados, operadores, estruturas de decisão e alguns comandos básicos.

Alguns exemplos de programas em Java.

AO FINAL DESTA AULA, VOCÊ ESTARÁ APTO A...

Trabalhar com arranjos.

Utilizar diferentes tipos de estruturas de repetição.

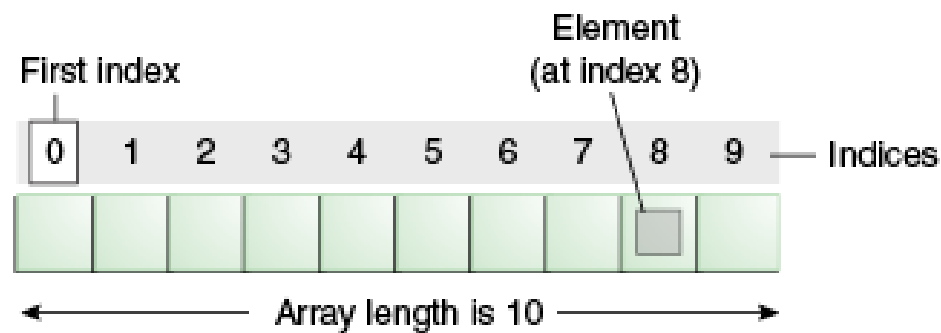
Aplicar corretamente comandos break e continue em estruturas de repetição.

ARRANJOS (ARRAYS)

Arranjos são objetos que armazenam variáveis do mesmo tipo:

- Tipos primitivos; e
- Referências a objetos (veremos mais a diante).

Arranjos são estruturas de dados eficientes, de tamanho fixo, que permitem acesso em tempo constante.



ARRANJOS (ARRAYS)

Formas de declaração de arrays:

```
<tipo>[] <nome>; // forma recomendada
<tipo> <nome>[]; // forma válida, mas não recomendada

<tipo>[] <nome> = new <tipo>[<tamanho>]; // cria um vetor de <tamanho> posições
<tipo>[] <nome> = {3, 4, 1, 4}; // cria de um vetor com 4 elementos

<tipo>[][] <nome_matriz>; // declaração
<nome_matriz> = new <tipo> [5][5]; // instânciação a posteriori
```

“<tipo>[]” também é um tipo, que representa um container de objetos de “<tipo>”.
Objetos arrays precisam ser instanciados para alocar o espaço para o armazenamento.

ARRANJOS (ARRAYS)

Exemplo de uso de arrays:

```
// Declaração com conjunto de valores
String[] nomes = { "Jaime", "Felipe", "Renato" };

// Alocação e posterior atribuição
int[] idades = new int[3];
idades[0] = 60;
idades[1] = 30;
idades[2] = 36;
```

O atributo “length” pode ser utilizado para obter o tamanho do array. No exemplo acima, “idades.length” retorna o valor 3.

LAÇOS DE REPETIÇÃO

Laço while:

```
//Avalia a expressão e depois executa  
while(<expressao_condicional>){  
    <comandos>;  
}
```

Laço do-while:

```
//Executa e depois avalia a expressão  
do{  
    <comandos>;  
}while(<expressao_condicional>;
```



LAÇOS DE REPETIÇÃO

Exemplo de laço while:

```
int count = 1;
while (count < 11) {
    System.out.println("Count is: " + count);
    count++;
}
```

Exemplo de laço do-while:

```
int count = 1;
do {
    System.out.println("Count is: " + count);
    count++;
} while (count < 11);
```


LAÇO FOR

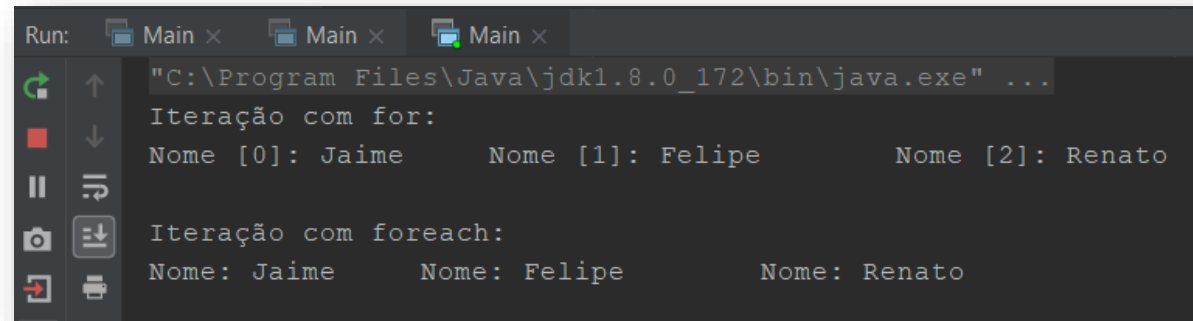
Laço for de propósito geral:

```
//As três opções são opcionais. O comando for( ; ; ) resulta em loop infinito.  
for(<inicializador>;<condicao_de_parada>;<iterador>){//Live Template:fori | itar  
    <comandos>;  
}
```

Laço for para iteração de conjuntos (arrays e coleções):

```
//Lê-se "para cada elemento <tipo> <i> em <conjunto>". Percorre todo o conjunto.  
for (<tipo> <i> : <conjunto>) {//Live Template: iter  
    <comandos>; //a variável de iteração <i> é somente leitura  
}
```

LAÇO FOR



```
Run: Main x Main x Main x
"C:\Program Files\Java\jdk1.8.0_172\bin\java.exe" ...
Iteração com for:
Nome [0]: Jaime      Nome [1]: Felipe      Nome [2]: Renato

Iteração com foreach:
Nome: Jaime      Nome: Felipe      Nome: Renato
```

Exemplo de laços for de propósito geral e para iteração de conjuntos:

```
String[] nomes = { "Jaime", "Felipe", "Renato" };

//println imprime valores individuais em formato padrão
System.out.println("Laço for convencional:");
// Mais complexo de iterar, mas fácil de obter o índice
for (int i = 0; i < nomes.length; i++)
    //printf permite a impressão, com formatação, de múltiplos valores
    System.out.printf("Nome [%d]: %s\t \t", i,  nomes[i]);

System.out.println("\n\nLaço for para iterar conjuntos:");
// Fácil de iterar, mas difícil de obter o índice. Itera todo o conjunto.
for(String s : nomes)
    System.out.printf("Nome: " + s + "\t \t");
```

COMANDO BREAK

O comando break é usado para interromper a execução de laços de repetição ou de um comando switch, essencialmente, por razões de otimização.

A execução do comando break encerra imediatamente a execução do laço.

```
// values possui 1.000.000 valores desordenados
int[] values = {20, 1, 3, 5, ... 2938390};
Scanner scanner = new Scanner(System.in);
int num = scanner.nextInt();

for(int v : values) //Use o debugger para depurar o código quando necessário!
    if (v == num) {
        System.out.println("Valor encontrado.");
        break; // v foi encontrado, não é preciso travar o fluxo até v[999999]
    }
```

COMANDO BREAK

Em alguns casos, a otimização com o comando break fica limitada pelo aninhamento dos laços de repetição. Veja o exemplo:

```
int[][] values = {{1, 3 ... 540}, {20, 23, 54 ... 943} ... {45, 98 ... 1302}};
Scanner scanner = new Scanner(System.in);
int num = scanner.nextInt();

for(int[] i : values) // a matriz funciona como um vetor de vetores
    for(int j : i)
        if (j == num){
            System.out.println("Valor encontrado.");
            // o valor foi encontrado, mas o break interrompe apenas o loop interno
            break;
        }
```

COMANDO BREAK

A solução é utilizar o comando break rotulado:

```
int[][] values = {{1, 3, 5}, {20, 23, 54, 94}, {45, 98, 102} };
Scanner scanner = new Scanner(System.in);
int num = scanner.nextInt();
rotulo:
for(int[] i : values) // a matriz funciona como um vetor de vetores
    for(int j : i)
        if (j == num){
            System.out.println("Valor encontrado.");
            break rotulo; // interrompe todos os laços aninhados sob o "rotulo"
        }
```

Cuidado: break rotulado só deve ser usado quando for indispensável, pois atrapalha a legibilidade do código, assim como os comandos “go to” de linguagens mais antigas.

COMANDO CONTINUE

O comando continue é usado para interromper a execução de uma iteração em laços de repetição, essencialmente, por razões de otimização.

A execução do comando encerra imediatamente a execução da iteração, mas não do loop.

```
while (true) {  
    System.out.printf("Digite um número positivo: ");  
    Scanner scanner = new Scanner(System.in);  
    int num = scanner.nextInt();  
  
    if (num < 0) continue;  
    else if (num % 2 == 0) System.out.println(num + " é par.");  
    else System.out.println(num + " é impar");  
}
```

COMANDO CONTINUE

Comando continue rotulado:

```
int[] values = {3, 9, 5, 20, 1 ... 1283}; // conjunto não ordenado
int[] otherValues = {1, 41, 5, 18, 29 ... 2938390}; // conjunto não ordenado

rotulo:
for(int v : values) // procura cada valor
    for(int o : otherValues) // no outro conjunto de valores
        if (v == o){
            System.out.println("O valor está contido em otherValues.");
            continue rotulo; // quebra a iteração do loop mais externo sob "rotulo"
        }
```

Cuidado: assim como o break, o uso desnecessário do continue rotulado deve ser evitado por questões de legibilidade do código.

BOAS PRÁTICAS: NOMES DE VARIÁVEIS*

Use nomes que revelem o propósito de uma variável:

```
int d; // elapsed time in days
```



```
int elapsedTimeInDays;
```

Evite informações erradas:

```
if(0 == 1) a = 01;  
else 1 = 01;
```

```
// Isso realmente é uma lista?  
double[] listOfGrades;
```

Use nomes pronunciáveis:

```
LocalDate genymdhms;
```



```
LocalDate generationTimestamp;
```

**Do livro Clean Code, de Robert C. Martin*

BOAS PRÁTICAS: NOMES DE VARIÁVEIS*

Faça distinções significativas:

```
void copyChars(char a1[], char a2[]){  
    for (int i = 0; i < a1.length; i++)  
        a2[i] = a1[i];  
}
```



```
void copyChars(char source[], char destination[]){  
    for (int i = 0; i < source.length; i++)  
        destination[i] = source[i];  
}
```

Use nomes longos para escopos grandes:

```
for (int i = 0; i < grades.length; i++) { // 'i' só tem importância no escopo do for  
    if(grades[i] >= bestStudentGrade)  
        bestStudentGrade = grades[i]; // bestStudentGrade é importante ao longo do algoritmo  
}
```

**Do livro Clean Code, de Robert C. Martin*

BOAS PRÁTICAS: NOMES DE VARIÁVEIS*

Substitua números mágicos por constantes com bons nomes.

```
if(numSecs < 86400) {  
    ...  
}
```



```
final int SECONDS_PER_DAY = 86400;  
if(elapsedTimeInSeconds < SECONDS_PER_DAY) {  
    ...  
}
```

Sempre que possível, prefira enuns às constantes:

```
public static final int OPEN = 1, WAITING_PAYMENT = 2, SHIPPED = 3, CONCLUDED = 4;  
public int status = OPEN;
```



```
public enum OrderStatus {OPEN, WAITING_PAYMENT, SHIPPED, CONCLUDED}  
public OrderStatus orderStatus = OrderStatus.OPEN;
```

**Do livro Clean Code, de Robert C. Martin*

RESUMO DA AULA

Arranjos são objetos que permitem armazenar um conjunto fixo de variáveis do mesmo tipo. Existem múltiplas formas de se criar um arranjo e cada uma têm sua utilidade específica. Há duas sintaxes básicas para o laço for, uma geral e outra para iteração de conjuntos. Os comandos break e continue podem ser utilizados para otimizar laços de repetição. Comandos break e continue rotulados são úteis, mas devem ser utilizados com cuidado. Escolher bons nomes leva algum tempo, mas economiza mais no longo prazo.

INFORMAÇÕES ADICIONAIS

Arranjos: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>

Formatação de impressão: <https://docs.oracle.com/javase/tutorial/essential/io/formatting.html>

Laço for: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html>

Break e continue: <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html>