Helder Henrique do Nascimento Peres

Java para Games
O tutorial para sua grande aventura

editora
VIENA
1ª Edição
Bauru/SP
Editora Viena
2015

Sumário

Lista de Siglas e Abreviaturas

DER	*Diagrama de Entidade e Relacionamento.*
MER	*Modelo de Entidade e Relacionamento.*
SQL	*Structured Query Language.*

1
Introdução

1. Introdução

Para programar em **Java**, é necessário entender o conceito de orientação a objetos. No mundo das linguagens de programação, existem dois tipos de paradigmas de programação: o paradigma estruturado (Exemplos: C, Cobol, Fortran, Lua) e o paradigma orientado a objetos (Exemplos: Java, C++, SmallTalk, Python, C#, Objective C). O paradigma estruturado, embora usado ainda hoje, é considerado antigo e, por alguns, obsoleto. Já o paradigma de orientação a objetos é utilizado em quase todas as linguagens atuais devido ao fato de permitir uma visão mais natural do mundo, o que facilita a desenvolver softwares, já que o mundo real é composto por objetos. Outra grande vantagem do paradigma orientado a objetos é a reutilização de código devido ao conceito de herança. Todavia, as aplicações de linguagens orientadas a objetos consomem mais memória (heap).

O **Java** é uma linguagem orientada a objetos, portanto, usam-se os conceitos de orientação a objetos. Outra grande vantagem do **Java** é a sua portabilidade, pois uma mesma aplicação **Java** escrita por um sistema A pode rodar em um sistema B e C sem ter muitas alterações. Isso acontece porque a **Máquina Virtual Java** consiste em criar um emulador do sistema operacional para executar suas aplicações. Por isso, programar em **Java** cria uma vantagem de distribuição de aplicativos, podendo deixar os jogos multiplataforma. Também é possível programar jogos para dispositivos **Android** utilizando a linguagem **Java**.

1.1. Lógica de Programação

Para aprender a programar, é necessário entender a lógica da programação, que consiste em algoritmos, segundo os quais o computador seguirá instruções para realizar tarefas.

1.1.1. Tipos de Dados

Toda linguagem de programação utiliza os símbolos alfanuméricos para realizar seus comandos e manipular dados. Ou seja, tudo o que você vê no seu teclado (letras, número e símbolos) é utilizado para programar.

As letras são chamadas de **caracteres** ou **strings** e os números, de inteiros.

Uma única letra consiste em caractere, e um conjunto de letras é chamado de string.

Os números inteiros são utilizados para realizar cálculos matemáticos (soma, subtração, divisão, multiplicação, raiz quadrada, potenciação).

Se por acaso um inteiro for adicionado a uma string, ele não será mais um número calculável, e sim apenas parte da string.

Veja o exemplo:

- 10 + 10 = 20 (operação de soma com números inteiros).
- “Banana” + 123 = “Banana123” (concatenando uma string com um inteiro).
- “Banana” + “Alface” = “BananaAlface” (concatenando duas strings).
- “Banana” + “123”= “Banana123” (concatenando duas strings).
- “123” + “123”= “123123” (concatenando duas strings).

Lembre-se de que qualquer valor passado dentro de aspas é uma string, independente se for uma letra ou não. Portanto, se for utilizado um valor "123", ele não será "cento e vinte e três", do qual se pode somar, subtrair, dividir, multiplicar ou fazer qualquer outra operação matemática, mas sim apenas uma string de "123".

Vejamos alguns dos tipos de dados mais utilizados em computação:

Tipo de dados	Descrição
Integer (ou int)	Valor numérico inteiro.
Float	Valor numérico com ponto flutuante (número decimal).
Character	Um caractere.
String	Um conjunto de caracteres.
Boolean	Variável lógica que aceita apenas um de dois valores, Verdadeiro ou Falso.
Void	Tipo de dado que aceita qualquer tipo de dado.

A tabela anterior descreve apenas alguns dos vários tipos de dados utilizados em linguagens de programação. Vale lembrar que cada linguagem de programação pode adotar um nome diferente para um tipo de dado. Por exemplo, em **SQL** (linguagem para banco de dados), utiliza-se a notação **integer**. Enquanto em **C**, **Java**, **C++** e várias outras linguagens utiliza-se a notação **int**. Ambas se referem a números inteiros, mas com nomes diferentes.

1.1.2. Ler e Escrever

Toda a linguagem de programação, bem como todo conceito de programação, se resume a uma coisa simples: trocar símbolos.

Basicamente, todo o programa de computador, por mais sofisticado que seja, lê símbolos e os manipula (trocando, modificando). Esses símbolos são dados que, como vimos anteriormente, podem ser compostos de letras, números e operadores lógicos. Caso queira conhecer mais sobre os fundamentos da computação, pesquise sobre a **Máquina de Turing**.

Assim, vamos ao seguinte algoritmo:

```
Int x;
```

Criamos uma variável de nome **x** do tipo inteiro. Isso significa que ela apenas poderá receber números inteiros.

```
X=120;
```

Assim, estamos atribuindo à variável **x** o valor 120.

```
Void y;
LEIA y;
```

O comando **LEIA** vai receber do usuário um valor para guardar dentro da variável **y**.

```
ESCREVA x+y;
```

O comando **ESCREVA** vai imprimir na tela o resultado da soma de x+y.

Se o usuário tivesse passado o valor 35 para a variável y, o resultado de x+y seria 155.

Vejamos alguns dos vários símbolos usados em computação.

Símbolo	Descrição
+	Adição.
-	Subtração.
/	Divisão.
*	Multiplicação.
=	Atribuição.
==	Igualdade.
!=	Não igual, diferente.
&&	E lógico.
||	Ou lógico.
<	Menor que.
>	Maior que.
++variável	Prefixo de incrementação.
--variável	Prefixo de decrementação.
Variável++	Sufixo de incrementação.
Variável--	Sufixo de decrementação.

Embora esses operadores sejam padrões para a grande maioria das linguagens, algumas podem variar desse exemplo. É recomendável estudar a linguagem que se vai utilizar. Todavia, os símbolos acima são usados em **Java**.

Os operadores matemáticos +, -, *, / são usados para fazer operações com números. O operador de adição (+) também é usado para concatenar (juntar) strings.

O operador de atribuição (=) serve para atribuir um valor a uma variável. Quando uma variável tem o seu valor atribuído, o valor anterior é sobrescrito. Ou seja, uma variável x tinha o valor 500 e agora atribuímos para ela o valor 10:

```
Int x;
X=500;
X=10;
```

A variável x vai ficar com o valor 10, pois foi o último valor atribuído a ela.

O operador de igualdade (==) compara os valores de duas variáveis para saber se ambas têm o mesmo valor. Já o operador de desigualdade (!=) compara dois valores para saber se eles não são iguais.

O operador E lógico (&&) faz uma junção de um ou dois valores ou variáveis em uma expressão. Exemplo:

```
Int a;
Int b;
```

```
a=10;
b=35;

50>a&&b;
```

O código 50>a&&b significa que o valor 50 é maior que a e b ao mesmo tempo, pois, 50 é maior que 10 e também maior que 35.

O operador ou lógico (||) vai escolher uma ou outra das opções disponíveis para comparar os valores ou executar ações. Veja o exemplo:

```
Int a;
Int b;

a=100;
b=200;

SE a<110 || a>10000 ENTÃO;
ESCREVA "Teste";
```

No exemplo anterior, a condição **SE** mostra que, se a variável **a** for menor que 110 ou **a** for maior que 10000, então a frase "Teste" vai ser escrita. Como atribuímos o valor 100 para a variável **a**, sabemos que ela é menor que 110, mas não é maior que 10000.

Como uma das condições é verdadeira, então a frase "Teste" vai ser escrita.

O operador lógico ou (||) aceita uma ou outra condição, não necessariamente as duas precisam ser verdadeiras.

Os prefixos de incrementação e decrementação, bem como os sufixos de incrementação e decrementação, são aqueles que aumentam ou diminuem o valor graduativamente de uma variável. Para entender melhor, vejamos o exemplo:

```
Int a;
Int x;

a=0;

PARA x de 0 até 10 FAÇA:
   Escreva a;
   a++;
FIM-PARA;
```

O código anterior fará de um intervalo de 0 a 10 a escrita do valor da variável **a** na tela. Ou seja, será escrito o valor da variável **a** 11 vezes na tela (pois zero conta como uma vez).

E ainda, como o operador de incrementação funciona para aumentar de um em um o valor da variável **a**, o resultado será uma contagem de 0 até 10, sendo impresso na tela. O operador de incrementação vai aumentando de um em um o valor da variável **a**, então, durante as onzes vezes que o comando **PARA** for executado, a variável **a** vai ter um valor diferente. Do mesmo modo, o decrementador iria diminuir o valor da variável **a**, provocando assim uma contagem regressiva.

A diferença de prefixo e sufixo de incrementação e decrementação é que o prefixo vai incrementar ou decrementar a variável antes da verificação do seu valor, e o sufixo, após a verificação do valor da variável.

1.1.3. Laços de Repetição

Como visto anteriormente, existem comandos chamados laços de repetição que repetirão outros comandos que estão dentro dele. Os comandos mais utilizados de laços de repetição são: **PARA**, **ENQUANTO** e **REPITA**.

O comando **PARA** faz uma repetição dentro de uma determinada faixa dada pelo usuário. Vejamos:

```
Int x;
PARA x de 0 até 1000 FAÇA:
   ESCREVA "Frase maneira."
FIM-PARA
```

Ou seja, de 0 até 1000 sua frase será impressa mil e uma vezes! Lembrando que é necessário criar uma variável para se guardar a contagem, por isso a variável **x** foi criada.

O comando **ENQUANTO** também é um laço de repetição, mas ele não atua em uma determinada faixa dada, mas sim enquanto a sua condição for verdadeira. Vejamos um exemplo:

```
Int a;
String b;
b="Teste";
ENQUANTO b=="Teste" FAÇA:
   ESCREVA a++;
FIM-ENQUANTO;
```

Enquanto a variável **b** for igual a "Teste", **a** vai continuar sendo incrementado e escrito. Ou seja, até o conteúdo de **b** não mudar, a variável **a** continuará sendo exibida. Nesse caso, como no laço de repetição não há opção de mudar o conteúdo de **b**, então o processamento fica para sempre dentro do laço de repetição, criando assim um loop infinito. É preciso prestar atenção para esse tipo de situação não acontecer, pois o programa ficará eternamente nesse trecho do código se não for interrompido.

O comando **REPITA** funciona de maneira semelhante:

```
Int a;
Int b;

a=10;
b=20;

REPITA:
   ESCREVA a+b;
   ESCREVA "Eis o resultado da soma de A+B: "
FIM-REPITA;
```

1.1.4. Condição

Uma condição é um comando em que se determina se uma condição é verdadeira ou não. Vejamos um exemplo para entender melhor:

```
Int a;
a=10;
SE a<15 ENTÃO:
   ESCREVA "A é menor que quinze, cara!"
FIM-SE;
```

O comando acima verifica se **a** é menor do que 15. Se a condição for verdadeira, então o comando **ESCREVA**, e qualquer outro que esteja dentro da condição, será executado. Se não for verdadeira, caso **a** não fosse menor que 15, a condição seria falsa e nenhum comando dentro da condição seria executado. Para tratar essa exceção, utiliza-se o comando **SE NÃO**.

Vejamos outro exemplo:

```
Int a;
a=20;
SE a<15 ENTÃO:
   ESCREVA "A é menor que quinze, cara!"
FIM-SE;

SE NÃO:
   ESCREVA "A não é menor que quinze, cara!"
FIM-SE NÃO;
```

Na primeira condição, é verificado se **a** é menor que quinze. Se **a** for menor que quinze, então a condição é verdadeira, mas, se ela for falsa, como é o caso, o comando **SE NÃO** será executado e a frase "A não é menor que quinze, cara!" vai ser exibida na tela.

1.1.5. Matriz e Vetor

Vetores e matrizes são estruturas de dados que servem para armazenar vários dados.

Um vetor é uma faixa de uma dimensão, onde em cada posição se guarda um valor. Todo vetor e toda matriz começam na posição zero. Vejamos um exemplo bem simples, mas que vai ajudá-lo a entender:

```
Int vetor[10];
```

No comando acima, criamos um vetor de 10 posições, sendo que o primeiro valor do vetor fica na posição 0, e o último valor, na posição 9.

No comando a seguir, vamos atribuir o valor 15 na primeira posição do vetor:

```
vetor[0]=15;
```

Agora vamos atribuir o valor 22 na segunda posição do vetor:

```
vetor[1]=22;
```

Isso pode confundir um pouco, por isso alguns programadores simplesmente criam um vetor com uma posição a mais para pularem a posição zero e não se confundirem.

Matrizes são semelhantes a vetores, todavia, elas podem ter duas ou mais dimensões. Vejamos um exemplo:

```
Int matrix[10][2];
```

Criamos uma matriz de 10x2, o que significa que em uma dimensão ela possui 10 posições (de 0 a 9) e em outra dimensão ela possui 2 (0 e 1). Pense em uma matriz como tendo linhas e colunas. Nossa matriz tem 10 linhas e 2 colunas. Bem simples, né?

No comando a seguir, estamos atribuindo o valor 125 para a primeira posição da matriz, a posição 0x0 (lembrando que a matriz também começa no zero!).

```
matrix[0][0]=125;
```

Agora vamos atribuir o valor 12 na segunda posição da primeira linha:

```
matrix[0][1]=12;
```

Como nossa matriz só tem 2 colunas, já atingimos todas as posições possíveis na primeira linha.

Então, se quisermos continuar acessando a matriz, temos de acessar a segunda linha, pois a primeira já está cheia.

```
matrix[1][0]=1555;
```

Na primeira posição da segunda linha, estamos adicionando o valor 1555, ou seja, estamos acessando a segunda linha na primeira coluna.

1.1.6. Função

Função é um bloco de código que pode ser chamado a qualquer momento dentro da aplicação. Vejamos um exemplo:

```
int função_somar (int x, int y){
   int resultado;
   resultado = x+y;
   return resultado;
}
```

Acima, foi criada uma função chamada **função_somar**.

O tipo de retorno da função é **int**, ou seja, ela vai nos retornar um dado do tipo inteiro.

Os parâmetros da função são as variáveis **x** e **y**. Essas variáveis são criadas nos parâmetros e são somente visíveis dentro da função. Esses parâmetros servem para que, quando a função for chamada, os valores passados como parâmetros sejam utilizados na função. Como nossa função é uma função que soma dois valores, os valores de **x** e **y** serão somados e o resultado será retornado para nós.

Toda a função deve ter um retorno compatível com o seu tipo de retorno. Não se pode criar uma função inteira e retornar uma string. Não dará certo.

Todas as variáveis criadas dentro da função só existem dentro da função. Caso queira acessar alguma variável dentro da função, deve-se passá-la como retorno. O exemplo a seguir ilustra melhor:

```
String elogiar(string elogio){
   elogio = elogio+" Parabéns! Muito sucesso!";
   return elogio;
}
```

Criamos a função **elogiar**. Ela recebe a variável **elogio** por parâmetro e ainda adiciona "Parabéns! Muito sucesso!".

Então, vamos chamar a função **elogiar** em nosso programa:

```
String meu_elogio;
meu_elogio="Você é muito bonita.";

elogiar(meu_elogio);

ESCREVA meu_elogio;
```

O resultado será "Você é muito bonita.". O acréscimo de "Parabéns! Muito sucesso!" foi feito dentro da função, mas, como não foi retornado, o conteúdo de **meu_elogio** continuou o mesmo. Para a função retornar um valor, precisa-se ter o comando return (necessário em todos os tipos de funções, menos em **void**).

Para receber o valor alterado de elogio, o correto deverá ser:

```
String meu_elogio;
meu_elogio="Você é muito bonita.";

meu_elogio=elogiar(meu_elogio);

ESCREVA meu_elogio;
```

Desse jeito, **meu_elogio** foi passado como parâmetro para a função **elogiar**, a função alterou o seu valor, acrescentando "Parabéns! Muito sucesso!" e retornou o resultado para a variável **meu_elogio**. Assim, podemos escrever o resultado alterado do jeito que queríamos com a função.

Por uma questão apenas de nomenclatura, função é uma parte do código que retorna algum valor e rotina são funções que não retornam valor, que são as de tipo **void**.

Em **Java**, chama-se de **método** tanto função quanto rotina, pois essa é a nomenclatura correta em linguagens orientadas a objetos, e às variáveis de **atributos**.

Exercícios Práticos

1. Qual é o nome correto de uma letra em programação? E um conjunto de letras?

2. Qual é o nome dado a um valor numérico utilizado para fazer operações matemáticas?

3. O que acontece se eu somar um valor Inteiro com um valor String?

4. O que faz um programa de computador?

5. Ao se criar a seguinte variável em um programa "Int x", qual é o nome da variável e qual é o seu tipo?

6. O que estes símbolos matemáticos (+, -, *, /, =, == e !=) significam em programação?

7. O que os seguintes símbolos lógicos significam (&&, ||, <, >)?

8. Quais são os dois principais comandos para laços de repetição?

9. Considere a seguinte situação:

```
Int a;
a=10;
SE a<15 ENTÃO:
   ESCREVA "A é menor que quinze, cara!"
FIM-SE;
```

Sabemos que a variável do tipo inteiro chamada "a" vale 10. O comando ESCREVA vai ser executado se a condição for verdadeira. Se "a" vale 10, a condição é verdadeira? Por quê?

10. Um vetor contém apenas uma dimensão enquanto a matriz pode ter duas ou mais. Essa frase é verdadeira? Explique.

11. Considere a seguinte função:

```
int função_somar (int x, int y){
   int resultado;
   resultado = x+y;
   return resultado;
}
```

a) Qual é o tipo de retorno da função?

b) Qual é o nome da função?

c) Quais são os parâmetros da função?

d) Qual é o retorno da função?

12. Usando seus conhecimentos, crie um algoritmo para trocar de camisa. Veja o exemplo:

```
Pegar sapato;
Pegar meia;
Colocar meia;
Colocar sapato;
```

13. Usando seus conhecimentos, crie um algoritmo para escovar os dentes.

14. Usando seus conhecimentos, crie um algoritmo para trocar o pneu do carro.

15. Crie um algoritmo para comprar ovos. Se a dúzia do ovo for menor que 5 reais, compre duas dúzias. Não se esqueça de levar uma ou mais bandejas, caso o ovo esteja barato.

16. Crie um algoritmo para jogar vídeo game. Considere que você só tem a televisão e que o vídeo game está em uma caixa. Se você não tiver nenhum jogo em sua casa, terá de ir à casa de algum amigo buscar.