Revisão – Programação em Java

Conteúdo

- Variáveis e constantes
- Tipos primitivos
- Operadores e expressões lógicas
- Estruturas condicionais
- Laços de repetição
- Funções
- Arrays
- Matrizes
- Manipulação de Strings

Variáveis e constantes

- Variável
  - É um mapeamento para um espaço alocado de memória, no qual se pode armazenar valores de um determinado tipo.
- Variáveis em Java

```
int idade;                  //Declaração
idade = 10;                 //Atribuição
int valor = idade;          //Recuperação
System.out.println(idade); //Recuperação e apresentação
```

Variáveis e constantes

- Variável
  - É um mapeamento para um espaço alocado de memória, no qual se pode armazenar valores de um determinado tipo.
- Variáveis em Java

```
int idade;                  //Declaração
idade = 10;                 //Atribuição
int valor = idade;          //Recuperação
System.out.println(idade); //Recuperação e apresentação
```

- Constante
  - É uma variável que possui um valor predeterminado, não permitindo sua alteração.
- Constantes em Java

```
final int matricula = 512010681; //Declaração e atribuição
int aluno = matricula;           //Recuperação
System.out.println(matricula);   //Recuperação
```

Tipos primitivos em Java

- Existem quatro tipos básicos para variáveis em Java.
  - Numéricas: **int, long, float, double**.
  - Caracteres / alfanuméricas: **char**, [**String**].
  - Lógicas: **boolean**.

Categoria	Tipo	Tamanho
Numérica	byte	8 bits
Numérica	short	16 bits
Numérica	int	32 bits
Numérica	long	64 bits
Numérica	float	32 bits
Numérica	double	64 bits
Alfanumérica	char	16 bits
Lógica	boolean	1 bit

Tipos primitivos em Java

- Existem quatro tipos básicos para variáveis em Java.
  - Numéricas: **int, long, float, double**.
  - Caracteres / alfanuméricas: **char**, [**String**].
  - Lógicas: **boolean**.

- Exemplo

```
String nome = "João";
int idade = 15;
char sexo = 'M';
boolean aprovado = true;

System.out.println("Dados do acadêmico"
          + "\nNome: " + nome
          + "\nIdade: " + idade
          + "\nSexo: " + sexo
          + "\nAprovado: " + aprovado);
```

Leitura de valores do usuário

- Para a leitura de valores a partir do console, é utilizada a classe **Scanner**.

- Exemplo

```
Scanner scanner = new Scanner(System.in);

int valorInteiro = scanner.nextInt();
String valorTextual = scanner.next();
double d = scanner.nextDouble();
```

- Para a leitura através de uma caixa de diálogo, é utilizada a classe **JOptionPane**.

- Exemplo

```
int valorInteiro = Integer.parseInt(JOptionPane.showInputDialog("Digite um valor inteiro: "));
String valorTextual = JOptionPane.showInputDialog("Digite uma String: ");
double d = Double.parseDouble(JOptionPane.showInputDialog("Digite um valor double: "));
```

- Para a apresentação de texto, pode-se utilizar o método **showMessageDialog**.

Operadores aritméticos

- São funções e operadores predefinidos para realização de cálculos matemáticos.
- Atuam sobre valores ou variáveis numéricas.

Operador	Exemplo	Comentário
=	x = y	O conteúdo da variável y é atribuído à variável x.
+	x + y	Soma o conteúdo de x e de y.
-	x - y	Subtrai o conteúdo de y do conteúdo de x.
*	x * y	Multiplica o conteúdo de x com o conteúdo de y.
/	x / y	Divide o conteúdo de x pelo conteúdo de y.
%	x % y	Obtém o resto da divisão inteira de x por y.
++	x++	Equivale a x = x + 1.
--	x--	Equivale a x = x - 1.

Operadores aritméticos

- São funções e operadores predefinidos para realização de cálculos matemáticos.
- Atuam sobre valores ou variáveis numéricas.

- Exemplo

```
int a, b, resultado;
a = 10;
b = 3;

resultado = a + b;    //resultado -> 13
resultado = a - b;    //resultado -> 7
resultado = a * b;    //resultado -> 30
resultado = a / b;    //resultado -> 3
resultado = a % b;    //resultado -> 1
resultado = a++;      //resultado -> ???
resultado = b--;      //resultado -> ???
```

Operadores aritméticos

- São funções e operadores predefinidos para realização de cálculos matemáticos.
- Atuam sobre valores ou variáveis numéricas.

- Exemplo

```
int a, b, resultado;
a = 10;
b = 3;

resultado = a + b;    //resultado -> 13
resultado = a - b;    //resultado -> 7
resultado = a * b;    //resultado -> 30
resultado = a / b;    //resultado -> 3
resultado = a % b;    //resultado -> 1
resultado = a++;      //resultado -> ???
resultado = b--;      //resultado -> ???
```

Resultado é 10 e 3, pois o incremento/decremento é feito depois da atribuição.

Para ter o efeito desejado, deve-se usar ++a e --b.

Operadores lógicos

- São funções ou operadores que atuam sobre valores ou variáveis booleanas (lógicas), avaliando seu conteúdo.
- Retornam um valor booleano (true / false).

Operador	Exemplo	Comentário
E - AND - &&	x && y	Representa o conteúdo de x E o conteúdo de y.
OU - OR - ||	x || y	Representa o conteúdo de OU o conteúdo de y.
NÃO - NOT - !	!x	Representa o valor inverso de x (não x).

Operadores lógicos

- Aplicação – tabelas verdade

Valor de x	Operador	Valor de y	Resultado
TRUE	&&	TRUE	TRUE
TRUE	&&	FALSE	FALSE
FALSE	&&	TRUE	FALSE
FALSE	&&	FALSE	FALSE

Valor de x	Operador	Valor de y	Resultado
TRUE	||	TRUE	TRUE
TRUE	||	FALSE	TRUE
FALSE	||	TRUE	TRUE
FALSE	||	FALSE	FALSE

Operador	Valor de y	Resultado
!	TRUE	FALSE
!	FALSE	TRUE

Operadores lógicos

- São funções ou operadores que atuam sobre valores ou variáveis booleanas (lógicas), avaliando seu conteúdo.
- Retornam um valor booleano (true / false).

- Exemplo

```
boolean a, b, resultado;
a = true;
b = false;

resultado = a && b;   //resultado -> false
resultado = a || b;   //resultado -> true
resultado = !a;       //resultado -> false
```

Operadores relacionais

- São funções ou operadores que relacionam dois ou mais valores ou variáveis, comparando-os.
- Retornam um valor booleano (true / false).

Operador		Exemplo	Comentário
Igual	==	x == y	Conteúdo de x é igual ao conteúdo de y?
Diferente	!=	x != y	Conteúdo de x é diferente do conteúdo de y?
Maior	>	x > y	Conteúdo de x é maior que o conteúdo de y?
Menor	<	x < y	Conteúdo de x é menor que o conteúdo de y?
Maior ou igual	>=	x >= y	Conteúdo de x é maior ou igual ao conteúdo de y?
Menor ou igual	<=	x <= y	Conteúdo de x é menor ou igual ao conteúdo de y?

Operadores relacionais

- São funções ou operadores que relacionam dois ou mais valores ou variáveis, comparando-os.
- Retornam um valor booleano (true / false).

- Exemplo

```
int a, b;
boolean resultado;
a = 5;
b = 2;

resultado = (a == b);          //resultado -> false
resultado = (a != b);          //resultado -> true
resultado = (a > b);           //resultado -> true
resultado = (a < b);           //resultado -> false
resultado = (a >= b);          //resultado -> true
resultado = (a <= b);          //resultado -> false
```

Estruturas condicionais

- Também conhecidas como estruturas de seleção, permitem definir um trecho de código que é executado apenas em condições determinadas.
- Permitem desviar o fluxo de execução de acordo com condições específicas.

- **Estruturas condicionais simples**

```
int a = 10;
int b = 5;

if(a > b) {
    System.out.println("Este código só é executado se a > b");
}
```

Estruturas condicionais

- Também conhecidas como estruturas de seleção, permitem definir um trecho de código que é executado apenas em condições determinadas.
- Permitem desviar o fluxo de execução de acordo com condições específicas.

- **Estruturas condicionais aninhadas**

```
int a = 10;
int b = 5;

if(a > b) {
    if(b < 50) {
        System.out.println("Este código só é executado se a > b e b < 50");
    }
} else {
    if(a > 30) {
        System.out.println(
            "Este código é executado quando a condição (a > b) não é satisfeita e a > 30");
    }
}
```

Estruturas condicionais

- Também conhecidas como estruturas de seleção, permitem definir um trecho de código que é executado apenas em condições determinadas.
- Permitem desviar o fluxo de execução de acordo com condições específicas.

- Estruturas condicionais - **múltiplas escolhas com SWITCH - CASE**

```
int a = 2;

switch(a) {
    case 1:
        System.out.println("O valor de a é 1");
        break;
    case 2:
        System.out.println("O valor de a é 2");
        break;
    case 3:
        System.out.println("O valor de a é 3");
        break;
    default:
        System.out.println("O valor de a não é nenhum dos valores testados");
        break;
}
```

Laços de repetição

- São utilizadas quando um trecho de código precisa ser executado várias vezes.
- Permitem definir a forma como um processo é realizado e replicá-lo repetidas vezes.

- **Laço de repetição contado – for**

```
for(int i = 0; i < 10; i++) {
    System.out.println("Este texto será impresso 10 vezes!");
}

for(int i = 10; i > 0; i--) {
    System.out.println("Esse texto será impresso 10 vezes!");
}

int inicio = 1, fim = 10;
for(int i = inicio; i <= fim; i++) {
    System.out.println("Executando...");

        if(i > 100)
        break;
}
```

Laços de repetição

- São utilizadas quando um trecho de código precisa ser executado várias vezes.
- Permitem definir a forma como um processo é realizado e replicá-lo repetidas vezes.

- **Laço de repetição condicional com teste no início – while**

```
int i = 0;
while (i < 10) {
    System.out.println("Este texto será impresso 10 vezes");
    i++;
}

boolean continua = true;
while(continua) {
    i *= 2;
    if(i > 1000) {
        System.out.println("i -> " + i);
        continua = false;
    }
}
```

Laços de repetição

- São utilizadas quando um trecho de código precisa ser executado várias vezes.
- Permitem definir a forma como um processo é realizado e replicá-lo repetidas vezes.

- **Laço de repetição condicional com teste no final – do...while**

```
int i = 0;
do {
    System.out.println("Este texto será impresso 10 vezes");
    i++;
} while (i < 10);

boolean continua = false;
do {
    System.out.println("Esse texto será exibido uma vez, pois o teste é realizado no final!");
} while(continua);
```

Métodos

- Um processo complexo pode ser dividido em problemas menores (simplificado).
  - Conceito da divisão e conquista.
- Uma função é um trecho de código que resolve um problema específico, por isso é identificado e pode ser utilizado em diferentes partes do código mediante sua chamada.

- **Exemplo – método sem retorno**

```
public static void main(String[] args) {
    nomeMetodo();
}

public static void nomeMetodo() {
    System.out.println("Este código é executado quando o método é chamado");
}
```

Métodos

- Um processo complexo pode ser dividido em problemas menores (simplificado).
  - Conceito da divisão e conquista.
- Uma função é um trecho de código que resolve um problema específico, por isso é identificado e pode ser utilizado em diferentes partes do código mediante sua chamada.

- **Exemplo – método com retorno**

```
public static void main(String[] args) {
    int resultado = nomeMetodo();
    System.out.println("O resultado retornado é: " + resultado);
}

public static int nomeMetodo() {
    return 10;
}
```

Métodos

- Um processo complexo pode ser dividido em problemas menores (simplificado).
  - Conceito da divisão e conquista.
- Uma função é um trecho de código que resolve um problema específico, por isso é identificado e pode ser utilizado em diferentes partes do código mediante sua chamada.

- **Exemplo – método com passagem de parâmetros**

```
public static void main(String[] args) {
    int resultado = nomeMetodo(50);
    System.out.println("O resultado retornado é: " + resultado);
}

public static int nomeMetodo(int parametro) {
    return parametro * 2;
}
```

Vetores (arrays unidimensionais)

- Conjunto de variáveis do mesmo tipo, que possuem o mesmo identificador (nome) e se distinguem por um índice.
- São alocados sequencialmente na memória.

- **Exemplo – declaração e inicialização**

```
int[] primeiroVetor = new int[5];          //Declaração de vetor de inteiros com 5 posições
int[] segundoVetor = {1, 2, 3, 4, 5};      //Declaração e inicialização na mesma instrução
```

- **Exemplo – atribuição e recuperação de valores**

```
primeiroVetor[0] = 10;          //Atribuição de valor
int valor = segundoVetor[2];    //Recuperação de valor
```

Vetores (arrays unidimensionais)

- Conjunto de variáveis do mesmo tipo, que possuem o mesmo identificador (nome) e se distinguem por um índice.
- São alocados sequencialmente na memória.

- **Exemplo – laços de repetição para percorrer um vetor**

```
for(int i = 0; i < primeiroVetor.length; i++) {
    primeiroVetor[i] = i * 10;
}

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

Matrizes (arrays multidimensionais)

- Conjunto de variáveis do mesmo tipo, que possuem o mesmo identificador (nome) e se distinguem por um **par de índices**.
- Também são alocados sequencialmente na memória.
- Logo, são vetores com duas dimensões (similar a uma tabela).

- **Exemplo – declaração e inicialização**

```
String[][] primeiraMatriz = new String[5][5];          //Declaração de uma matriz de Strings 5x5
String[][] segundaMatriz = {{"a", "b"},{"c", "d"}};    //Declaração e inicialização
```

- **Exemplo – atribuição e recuperação de valores**

```
primeiraMatriz[2][4] = "Texto";              //Atribuição de valor
String letra = segundaMatriz[1][1];          //Recuperação de valor
```

Matrizes (arrays multidimensionais)

- Conjunto de variáveis do mesmo tipo, que possuem o mesmo identificador (nome) e se distinguem por um **par de índices**.
- Também são alocados sequencialmente na memória.
- Logo, são vetores com duas dimensões (similar a uma tabela).

- **Exemplo – laços de repetição para percorrer um vetor**

```
for(int i = 0; i < primeiraMatriz.length; i++) {
    for(int j = 0; j < primeiraMatriz.length; j++) {
        primeiraMatriz[i][j] = "***";
    }
}
```

Manipulação de Strings

- É possível manipular variáveis textuais como cadeias de caracteres. O método que recupera um caractere de uma String é o **charAt(i),** onde **i** é o índice buscado.
  - **Exemplo:** charAt(2) na palavra "escola" retorna o caractere 'c'.

- **Exemplo**

```
String palavra = "programação";

System.out.println(palavra.charAt(0)); //p
System.out.println(palavra.charAt(2)); //o
System.out.println(palavra.charAt(3)); //g
System.out.println(palavra.charAt(7)); //a
```

- Com o uso de laços de repetição e a função charAt(i), é possível analisar e manipular qualquer cadeia de caracteres.

Manipulação de Strings

- Outras funções disponíveis para o tratamento de Strings.

Método	Comentário
substring(int inicio, int fim)	Retorna a substring a partir do caractere da posição 'inicio' até a posição 'fim'.
equals(String s)	Verifica se a String é igual a s.
equalsIgnoreCase(String s)	Identico ao anterior, mas sem diferenciar entre caracteres maiúsculos e minúsculos.
contains(String s)	Verifica se s é uma substring do objeto que chama a função.
startsWith(String s)	Verifica se a String inicia com a substring s.
endsWith(String s)	Verifica se a String termina com a substring s.
toUpperCase()	Transforma todos os caracteres em maiúsculos.
toLowerCase()	Transforma todos os caracteres em minúsculos.
replace(String s1, String s2)	Substitui os caracteres indicados em 's1' pelos indicados em 's2'.
trim()	Remove espaços no início e final da String.

Referências

DEITEL, H. M. **Java: como programar**. H. M Deitel e P. J. Deitel - 8a ed. Porto Alegre: Prentice-Hall, 2010.

Leitura complementar

TutorialsPoint Java (http://www.tutorialspoint.com/java).