Resumo da Linguagem Java

Linguagem e técnica de programação (Universidade Veiga de Almeida)

LINGUAGEM JAVA - RESUMO

1. ESTRUTURAS BÁSICAS

1.1. Estrutura Geral de um Programa

```
public class nome do programa {
      public static void main(String[] args) {

            instruções do programa

      }
}
```

1.2. Tipos Primitivos de Dados em Java

Algoritmos	Equivalentes em Java	Tamanho	Valores Permitidos
inteiro	byte short int long	1 byte 2 bytes 4 bytes 8 bytes	-128...127 -32768...32767 $-2^{31}...2^{31}-1$ $-2^{63}...2^{63}-1$
real	float double	4 bytes 8 bytes	
lógico	boolean	1 byte	**true** e **false**
caractere	char	2 bytes	
literal	String		

Valores do tipo char devem ser informados entre aspas simples (exemplo: 'A', 'c') enquanto valores do tipo String entre aspas duplas (“Teste”, “Lógica de Programação”). Valores do tipo real utilizam o ponto decimal como nos algoritmos.

1.3. Operadores

Operador	Símbolo	Tipo	Para que serve	Precedência
Inverção de sinal	-	Unário	Inverte o sinal de um valor numérico.	7
Manutenção de sinal	+	Unário	Mantém o sinal de um valor numérico.	7
Negação	!	Unário	Inverte o valor de uma expressão lógica.	7
Divisão	/	Binário	Divide dois valores numéricos.	6
Multiplicação	*	Binário	Multiplica dois valores numéricos.	6
Resto da divisão	%	Binário	Dá o resto da divisão entre dois inteiros.	6
Adição	+	Binário	Adiciona dois valores numéricos. Concatena dois literais.	5

Subtração	-	Binário	Subtrai dois valores numéricos.	5
Menor que	<	Binário	Verifica se um valor é menor do que outro.	4
Maior que	>	Binário	Verifica se um valor é maior do que outro.	4
Menor ou igual	<=	Binário	Verifica se um valor é menor ou igual do que outro.	4
Maior ou igual	>=	Binário	Verifica se um valor é maior ou igual do que outro.	4
Igual a	==	Binário	Verifica se um valor é igual a outro.	3
Diferente	!=	Binário	Verifica se um valor é diferente a outro.	3
Conjunção (.e.)	&&	Binário	Conjunção de dois valores lógicos.	2
Disjunção (.ou.)	||	Binário	Disjunção de dois valores lógicos.	1

1.4. Atribuição

O Java apresenta alguns operadores para atribuição citados na tabela abaixo. Nos algoritmos em pseudocódigo, representávamos a atribuição com o símbolo ←.

Operador	Exemplo	Significado
=	`A = B`	Atribuição simples
+=	`A += B`	A = A + B
-=	`A -= B`	A = A - B
*=	`A *= B`	A = A * B
/=	`A /= B`	A = A / B
%=	`A %= B`	A = A % B
++	`A++`	A = A + 1
--	`A--`	A = A - 1

1.5. Variáveis

Variáveis em Java podem ser declaradas em qualquer lugar do programa, desde que dentro do método *main*. Para se declarar uma variável em Java, basta escrever seu tipo seguido do nome da variável, como mostrado abaixo:

```
int a, b;
char caractere;
String texto, msg;
boolean flag;
```

No exemplo são declaradas 6 variáveis: “a” e “b” do tipo int, “caractere” do tipo char, “texto” e “msg” do tipo String e “flag” do tipo boolean. Como mostrado, podemos declarar diversas variáveis de um mesmo tipo em uma única linha, separando seus nomes por vírgulas. A

declaração de variáveis pode vir em qualquer lugar no programa.

Pode-se também incluir na linha de declaração uma atribuição para inicializar a variável (atribuir a ela um valor inicial), como mostrado abaixo:

```
double pi = 3.14159, numero = 10.5;
```

1.6. Conversão de Valores

A conversão entre tipos numéricos pode ser feita diretamente, informando o nome do tipo alvo a frente do valor a ser convertido, entre parênteses. Por exemplo:

```
int a = (int) 10.0;
double b = (double) (a * 4);
byte c = (byte) a;
```

Conversão de String para byte: `Byte.parseByte ( string )`
Conversão de String para short: `Short.parseShort ( string )`
Conversão de String para int: `Integer.parseInt ( string )`
Conversão de String para long: `Long.parseLong ( string )`
Conversão de String para float: `Float.parseFloat ( string )`
Conversão de String para double: `Double.parseDouble ( string )`
Conversão de String para boolean: `Boolean.parseBoolean ( string )`

Conversão de byte para String: `Byte.toString ( byte )`
Conversão de short para String: `Short.toString ( short )`
Conversão de int para String: `Integer.toString ( int )`
Conversão de long para String: `Long.toString ( long )`
Conversão de float para String: `Float.toString ( float )`
Conversão de double para String: `Double.parseDouble ( double )`
Conversão de boolean para String: `Boolean.parseBoolean ( boolean )`

2. Entrada e Saída

2.1. Saída no Console

Com quebra automática de linha:

```
System.out.println( variável ou valor String );
```

Sem quebra automática de linha:

```
System.out.print( variável ou valor String );
```

2.2. Saída em Janela

```
JOptionPane.showMessageDialog(null, variável ou valor String );
```

OBS.: Para utilizar JoptionPane é necessário importar o pacote javax.swing colocando a seguinte linha no início do programa:

```
import javax.swing.*;
```

2.3. Entrada em Janela

```
JOptionPane.showInputDialog( variável ou valor String );
```

O método retorna um valor String correspondendo ao valor digitado pelo usuário na janela mostrada.

OBS.: Para utilizar JOptionPane é necessário importar o pacote javax.swing colocando a seguinte linha no início do programa:

```
import javax.swing.*;
```

2.4. Entrada em Console

```
Scanner entrada = new Scanner(System.in);
int i = entrada.nextInt();
double d = entrada.nextDouble();
String s = entrada.next();
boolean b = entrada.nextBoolean();
```

OBS.: Para utilizar Scanner é necessário importar o pacote java.util colocando a seguinte linha no início do programa:

```
import java.util.*;
```

3. Estruturas de Controle

3.1. Estrutura If

```
if ( condição )
{
      conjunto de instruções 1
}
else
{
      conjunto de instruções 2
}
```

3.2. Estrutura switch

```
switch ( expressão )
{
      case valor1:
            conjunto de instruções 1;
            break;

      case valor2:
```

```
            conjunto de instruções 1;
            break;

      ...

      default:
            conjunto de instruções n;
}
```

3.3. Operador "?"

```
condição : valor_verdadeiro : valor_falso;
```

Retorna *valor_verdadeiro* caso a condição seja verdadeira e *valor_falso* caso a condição seja falsa.

3.4. Estrutura while

```
while ( condição )
{
      conjunto de instruções;
}
```

3.5. Estrutura do...while

```
do
{
      conjunto de instruções;
}while ( condição );
```

3.6. Estrutura for

```
for(início; condição; fim)
{
      conjunto de instruções;
}
```

- *início* : Executado antes de iniciar a repetição do laço;
- *condição* : O laço repete enquanto ela for verdadeira;
- *fim* : Executado após cada repetição.

4. Arranjos em Java (*Arrays*)

4.1. Declaração

```
tipo [] nome_variável;     ou    tipo nome_variável [];
```

4.2. Inicialização e Alocação

```
nome_variável = new tipo[tamanho];
```

4.3. Acesso a um Elemento específico

```
nome_variável[índice]
```

4.4. Número de Elementos do Arranjo

```
nome_variável.length
```

5. Tratamento e Exceções

5.1. Conceito de Exceção

Uma exceção é um evento extraordinário que ocorre durante a execução de uma instrução em um programa em Java. Geralmente está associada com algum erro que ocorreu durante a operação de um método, operador ou comando da linguagem. A exceção é uma forma de dizer ao programa que um erro aconteceu durante a operação.

5.2. Tratamento de uma Exceção

O comportamento padrão de um programa em Java quando uma exceção ocorre é finalizar e mostrar uma mensagem descrevendo a exceção que ocorreu. Existe uma forma de escrevermos nosso próprio tratamento para uma exceção através do bloco **try...catch:**

```
try
{
      instruções com exceções a serem tratadas;
}
catch(TipoDeExcecao1 ReferenciaExcecao1) {
      instruções para tratamento da exceção 1;
}
catch(TipoDeExcecao2 ReferenciaExcecao2) {
      instruções para tratamento da exceção 2;
}
.
.
.
finally {
      instruções;
}
```

Uma exceção que ocorre em um bloco *try* é capturada por um tratador de exceções especificado por um bloco *catch* imediatamente após aquele bloco *try*. Caso não exista um bloco *catch* para uma determinada exceção, ela não será tratada pelo bloco *try*.

O bloco *try* pode ser seguido por zero ou mais blocos *catch*. Se um bloco *try* é executado e nenhuma exceção é disparada, todos os tratadores de exceções são pulados e o controle é retomado na primeira instrução depois do último tratador de exceções. Se um bloco *finally* segue o último bloco *catch*, o código do bloco *finally* é executado independentemente de uma exceção ser ou não disparada.

Os tratadores de exceções estão contidos em blocos *catch*. Cada bloco *catch* inicia com a palavra-chave *catch* seguida por parênteses que contêm um nome de classe (que especifica o tipo de exceção a ser capturada) e um nome de parâmetro. O tratador pode fazer referência ao objeto disparado através desse parâmetro. Depois dele há um bloco que delimita o código de tratamento de exceções. Quando um tratador captura uma exceção, o código no bloco *catch* é executado.

O *catch* que captura um objeto Exception captura todas as exceções. O primeiro tratador de exceções que corresponder ao tipo da exceção é executado – todos os outros tratadores de exceção para o bloco *try* correspondente são ignorados.

6. Leitura de Arquivos com Scanner

6.1. Abrir arquivo

```
Scanner variavel = new Scanner( new File(nome do arquivo) );
```

OBS.: Para usar a classe File é necessário importar o pacote “java.io”. A linha acima pode disparar uma exceção **FileNotFoundException** que deve ser tratada pelo programa.

6.2. Ler uma linha do arquivo

```
String variavel_string = variavel_scanner.nextLine();
```

6.3. Ler outros tipos de dados do arquivo

```
int var_int = variavel_scanner.nextInt();
double var_double = variavel_scanner.nextDouble();
String var_string = variavel_scanner.next();
```

6.4. Verifica se ainda há linhas no arquivo a serem lidas

```
variavel_scanner.hasNextLine();
```

Devolve true se ainda há linhas a serem lidas no arquivo e false caso o scanner já tenha chegado ao fim do arquivo.

6.5. Troca de delimitador de dados

```
variavel_scanner.useDelimiter(novo delimitador);
```