FUNDAMENTOS DA LINGUAGEM JAVA

Símbolos:

Comentários de uma linha: //

//Isto é um comentário

Comentários de mais de uma linha: /* */

/

Isto também é um comentário

Mas tem mais de uma linha...

**/*

Terminador de instrução: **;**

TODA INSTRUÇÃO DEVE TERMINAR COM “;” (PONTO-E-VÍRGULA)!!!

Tipos de dados primitivos:

Tipo	**Tamanho**
boolean	true / false
byte	8-bit
char	16-bit
short	16-bit
int	32-bit
long	64-bit
float	32-bit
double	64-bit

Declaração de variáveis:

<tipo de dado> <nome da variável>

int x; *//declara uma variável do tipo inteiro chamada “x”*

float y; *//declara uma variável do tipo ponto flutuante chamada “y”*

Operadores:

Operador	Função	Exemplo
=	Atribuição	X = 10;
==	Igualdade	X == y; //não funciona com classes (ver mais a frente na seção dedicada a classes)
>	Maior	X > 3
<	Menor	X < 3
>=	Maior ou igual	X >= 3
<=	Menor ou igual	X <= 3
&&	E (and)	(X > 2) && (X < 10)
||	Ou (or)	(X < 2) || (X > 10)
!	Negação	!estaVazio; *//retorna a negação do valor da variável. Se esta vale true, retorna false e vice-versa.*
!=	Diferente de	X != 0 //X possui qualquer valor diferente de zero. Também não funciona com classes
+	Adição	X + 4; //serve para concatenar Strings. String s = “João” + “ da Silva”;
-	Subtração	X – 4
/	Divisão	22 / 4 //retorna 5 22.0 / 4 //retorna 5.5
%	Resto da divisão	22 / 4 //retorna 2 22.0 / 4 //retorna 2.0
+=	Auto-incremento	X += 3; //x tem seu valor acrescido de 3. O resultado dessa operação é armazenado no próprio x. É o mesmo que x = x + 3.
-=	Auto-decremento	X -= 3; //x tem seu valor decrescido de 3. O resultado dessa operação é armazenado no próprio x. É o mesmo que x = x - 3.
*=	Auto-multiplicação	X *= 2; //x tem seu valor multiplicado por 2. O resultado dessa operação é armazenado no próprio x. É o mesmo que x = x * 2.
/=	Auto-divisão	X /= 2; //x tem seu valor dividido por 2. O resultado dessa operação é armazenado no próprio x. É o mesmo que x = x / 2.
%=	Auto-divisão (resto)	X %= 2; //x tem seu valor dividido por 2. O valor do resto dessa divisão é armazenado no próprio x. É o mesmo que x = x % 3.

++	Incremento	Incrementa de 1 uma variável numérica e armazena nela o novo valor. Pode ser usado antes ou depois da variável, porém, dependendo da posição em que for utilizado, o comportamento será modificado. cont++ *//Lê o valor de cont e depois incrementa 1* int cont = 5; System.out.println(cont++)*//Imprime 5* System.out.println(cont)*//Imprime 6* ++cont *//Incrementa 1 e depois lê o valor de cont* int cont = 5; System.out.println(++cont)*//Imprime 6* System.out.println(cont)*//Imprime 6*
--	Decremento	Decrementa de 1 uma variável numérica e armazena nela o novo valor. Pode ser usado antes ou depois da variável, porém, dependendo da posição em que for utilizado, o comportamento será modificado. Cont-- *//Lê o valor de cont e depois decrementa 1* int cont = 5; System.out.println(cont--)*//Imprime 5* System.out.println(cont)*//Imprime 4* --cont *//Decrementa 1 e depois lê o valor de cont* int cont = 5; System.out.println(--cont)*//Imprime 4* System.out.println(cont)*//Imprime 4*

Desvios condicionais:

IF:

Sintaxe:

```
if( <lista de condições> ){
  <lista de comandos>
}
else{
  <lista de comandos>
} //o else é opcional
```

Exemplos:

```
if( x > 2 ){
   X = 5;
}
```

```
if( y == 0 && x > 3 ){
   z = 9;
}
else{
   z = 10;
}
```

Existe um operador que substitui o IF para condições mais simples, que é o **?** (conhecido como operador ternário). Esse operador é utilizado em atribuições ou retorno de funções, quando as mesmas dependem de um teste condicional.

Sintaxe:

<Variável> = <condição booleana> ? <resultado se for true> : <resultado se for false>;

Exemplo:

```
X = (y > 5) ? 2 : y;
```

O que é equivalente a:

```
if ( y > 5 ){
  X = 2;
}
else{
  X = y;
}
```

Obs.: IF's com apenas uma linha no seu bloco de comandos podem omitir as chaves:

```
if ( y > 5 )
  X = 2;
else
  X = y;
```

Já em:

```
if ( y > 5 ){
  X = 2;
  y = 5;
}
else{
  X = y;
  Y = 3;
}
```

as chaves são obrigatórias.

SWITCH: Usado para se executar uma ou mais ações quando várias alternativas são possíveis de acordo com o valor de uma variável inteira ou caracter. Semelhante ao “case” do Pascal.

Sintaxe:

switch(<variável>){
	case <valor 1>:
		<lista de comandos>
		[break;]
	case <valor 2>:
		<lista de comandos>
		[break;]
	[default:
		<lista de comandos>]
  }

Exemplo:

```
switch( x ){
	case 100:
		y = 1000;
		break; //Executa apenas esse bloco e pára a execução do switch, se x estiver valendo 100.
	case 150:
		x += 3; //Aqui como não tem o break ele vai executar os comandos abaixo (se x  estiver valendo 100) até encontrar um break, mesmo se o valor de x não for igual ao que está sendo testado (1000, neste caso).
	case 1000: //Executa apenas esse bloco e pára a execução do switch, se x  estiver valendo 1000.
		x += 3;
		break;
	default: //Executa se a variável não estiver valendo um dos valores testados
		x += x;
}
```

Estruturas de repetição:

for: Normalmente é utilizado quando sabe-se o número exato de repetições ou quando há a necessidade de um incremento automático de uma variável contadora e/ou de controle.

Sintaxe:

for(<inicialização variável de controle>; <condições a testar>; <incremento variável de controle>)
{
	<lista de comandos>
}

Exemplo:

```
for( int i = 1; i <= 500; i++ ){
	System.out.println( “Estou passando aqui pela ” + i + “a vez...” );
}
```

No caso acima, quando o programa chegar no ponto de execução do **for**, acontecerá:

1. A inicialização da variável “i” com o valor 1 (int i = 1);
2. O teste se a mesma é menor ou igual a 500 (i <= 500);
3. Se o teste retornar **true**, a lista de comandos será executada (neste caso, uma mensagem será escrita no console) (System.out.println(“Estou passando aqui pela ” + i + “a vez...”););
4. Após o fim da execução da lista de comandos, a variável de controle será incrementada (i++) e o loop retorna ao passo 2, até que a condição de teste retorne **false**, quando o mesmo acaba.

Ao contrário do Pascal, no Java o **for** pode ter condições de teste mais complexas, como abaixo:

```
for( int i = 1; i <= 20 && x < 1000; i++ ){
	x *= i;
}
```

while: Normalmente utilizado quando não se tem certeza de quantas iterações serão executadas. O teste condicional é feito antes de entrar na execução da lista de comandos.

Sintaxe:

while(<condições a testar>){
	<lista de comandos>
}

Exemplo:

```
boolean achou = false;
while( i < vet.length && !achou ){
	achou = ( vet[i] == param );
	i++;
}
```

No caso acima, quando o programa chegar no ponto de execução do **while**, acontecerá:

1. O teste se ”i” é menor do que o tamanho da vetor declarado na variável “vet” (i < vet.length).
2. Se “i” for menor do que o tamanho de “vet”, a negação do valor da variável “achou” declarada como **false** (!achou);

3. O teste se o resultado de ambas as condições é verdadeiro (i < vet.length && !achou);
4. Enquanto o teste retornar **true**, a lista de comandos será executada (neste caso, uma comparação de um determinado elemento de um vetor com uma variável para ver se ambos são iguais. Se forem, "achou" valerá **true**, senão, "achou" valerá **false**. Depois disso, o incremento da variável i);
5. Após o fim da execução da lista de comandos, o programa volta ao passo 1 até que a condição seja **false**.

É importante observar que o Java, assim como o Pascal, faz o que chamamos de "curto-circuito" em testes condicionais. No exemplo acima o primeiro teste é feito (i < vet.length). Por ser um teste do tipo "E", se essa condição já não for aceita (retornar **false**), o segundo teste não é feito, pois em casos de "E" ambas as condições devem ser verdadeiras. Se o resultado for **true**, aí sim a outra condição é testada. Isso é muito útil e importante, pois ajuda a montarmos condições compostas na ordem correta, evitando possíveis erros.

do..while: Normalmente utilizado quando não se tem certeza de quantas iterações acontecerão e quando se deseja que a lista de comandos seja executada pelo menos uma vez. O teste condicional é feito ao final da execução da lista de comandos.

Sintaxe:

do{
	<lista de comandos>
} while(<condições a testar>)

Exemplo:

```
do{
	x *= i++; //multiplica o valor corrente de x por i e guarda o resultado em x. Após essa operação, incrementa o valor de i.
}while(x < 1000);
```

No caso acima, quando o programa chegar no ponto de execução do **while**, acontecerá:

1. Multiplica o valor corrente de x por i e guarda o resultado em x (x*= i). Após essa operação, incrementa o valor de i (i++);
2. Depois testa se x é menor do que 1000 (x < 1000). Se for, volta para o passo 1. Se não for, aborta o loop.

OBS.: Todas as estruturas de repetição podem ser abortadas prematuramente, sem que se aguarde o alcance da condição de parada. Para isso utilizamos a palavra reservada **break**. Vejamos um exemplo.

```
while( i < vet.length){
	if ( vet[i] == param ){
		break; //aborta o loop
	}

	i++;
}
```

Vetores:

A declaração de vetores pode ser de duas maneiras:

<Tipo de dado> <nome da variável>[ ]

ou

<Tipo de dado>[ ] <nome da variável>

Exemplo:

```
String nomes[]; //declara um vetor de strings
int[] vet; //Declara um vetor de inteiros
```

TODOS OS VETORES EM JAVA TÊM COMO ÍNDICE INICIAL O NÚMERO ZERO (0)

Para inicializar/definir o tamanho do vetor é necessário usar a palavra-reservada **new**, que é responsável por alocar memória para seu armazenamento. Veja a sintaxe:

```
String nomes[] = new String[10]; //declara/aloca um vetor de strings com dez posições (0..9)
int[] vet = new int[5]; //Declara/aloca um vetor de inteiros com 5 posições (0..4)
```

É possível declarar um vetor já atribuindo valores a suas células:

```
String nomes[] = {“João”, “Maria”, “José”}; //declara/aloca um vetor de strings com três posições, onde “João” é armazenado na posição 0, “Maria” na posição 1 e “José” na posição 2
```

Para vetores multidimensionais (matrizes) as regras são as mesmas:

```
int[][] vet = new int[4][4]; //Declara uma matriz de inteiros com quatro linhas e quatro colunas
int matriz[][] = {{1, 2, 1, 3}, {1, 2, 3, 5}}; //Declara/aloca uma matriz de inteiros com duas linhas e quatro colunas já inicializando valores
```

Em situações como

```
String nomes[ ] = new String[10];
```

o Java **não inicializa o valor de cada uma das células do vetor**, apenas a alocação de memória para ele é feita. Se você desejar inicializar o valor de cada uma das posições do vetor, por exemplo, com a String “”, você deve escolher uma das duas alternativas abaixo:

```
String nomes[] = {“”, “”, “”, “”, “”, “”,“”, “”, “”, “”};
```

Ou

```
String nomes[ ] = new String[10];

for( int i = 0; i < 10; i++ ){
	nomes[i] = “”;
}
```

Um programa Java:

Todo programa Java deve possuir algumas características para que possa ser passível de ser carregado e executado. Vejamos:

1. A classe principal do programa deve possuir uma função chamada **main**, que possui um cabeçalho que deve ser respeitado obrigatoriamente;
2. O nome da classe deve ser o mesmo nome do arquivo, inclusive respeitando maiúsculas e minúsculas;
3. A classe principal deve ser pública.

Exemplo:

```
public class OlaMundo{
	public static void main( String args[ ] ){
		System.out.println( “Meu primeiro programa Java!” );
```

```
        }
}
```

Explicando:

1. A classe deve ser pública (**public** class OlaMundo);
2. Deve existir uma função **main** com um cabeçalho que dever ser exatamente assim (public static void main(String args[ ]));
3. O nome da classe deve ser igual ao nome do arquivo (o arquivo desse programa seria, obrigatoriamente, OlaMundo.Java).

A função main:

Expliquemos o cabeçalho da função main:

public – como é o ponto de entrada de um programa Java, essa função deve ser pública. Digamos que a função main é o "programa principal" em uma aplicação Java.
static – É o especificador da linguagem que indica que um método ou atributo é pertencente à classe e não precisa de instância para ser chamado. Como a função **main** fica em uma classe que será chamada sem que a máquina virtual precise criar um objeto que a referencie, esse método deve ser declarado como static.
void – A função **main** não retorna valor algum após sua execução.
main – O nome da função. Obrigatoriamente deve ser todo em minúsculo.
`String args[ ]` – `É um vetor com parâmetros passados pela linha de comando.`