Centro Tecnológico
Departamento de Informática

Prof. Vítor E. Silva Souza
http://www.inf.ufes.br/~vitorsouza

[Desenvolvimento OO com Java]

Variáveis primitivas e controle de fluxo

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conteúdo do curso

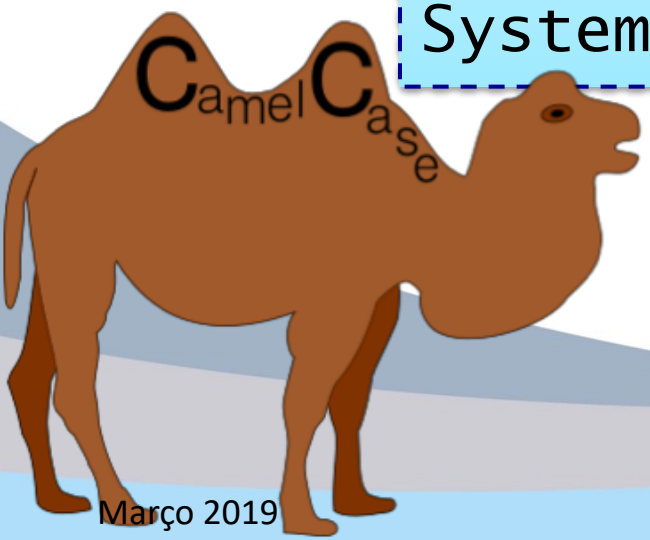
- O que é Java;
- ➡ Variáveis primitivas e controle de fluxo;
- Orientação a objetos básica;
- Um pouco de vetores;
- Modificadores de acesso e atributos de classe;
- Herança, reescrita e polimorfismo;
- Classes abstratas e interfaces;
- Exceções e controle de erros;
- Organizando suas classes;
- Utilitários da API Java.

Estes slides foram baseados na apostila do curso FJ-11: Java e Orientação a Objetos da Caelum e na apostila Programação Orientada a Objetos em Java do prof. Flávio Miguel Varejão.

Declaração e uso de variáveis

- Java funciona como C, com tipagem estática:

```
// Define a variável e já atribui um valor:
int idade = 35;
System.out.println(idade); // 35

// Define a variável, depois atribui um valor:
int idadeAnoQueVem;
idadeAnoQueVem = idade + 1;
System.out.println(idadeAnoQueVem); // 36
```



Para executar, o código acima deve estar dentro de um método main(), como explicado na aula anterior.

Convenção de código: variáveis começam com letra minúscula e usam *CamelCase*.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos primitivos

- Existem 8 tipos de dados básicos para valores inteiros, reais, caracteres e lógicos;
- São chamados primitivos porque:
  - *Não são objetos;*
  - *São armazenados na pilha.*
- Java não é OO pura por causa deles;
- Existem por motivos de eficiência;
- São completamente definidos na especificação Java (nome, tamanho, etc.).

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos primitivos inteiros

Tipo	Tamanho	Alcance
byte	1 byte	-128 a 127
short	2 bytes	-32.768 a 32.767
int	4 bytes	-2.147.483.648 a 2.147.483.647
long	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

```
/* Cuidados com o alcance do tipo: */
byte b = 127;
System.out.println("b = " + b); // b = 127

b++;
System.out.println("b = " + b); // b = -128
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Representação de valores literais

```
// Especificando o tipo do literal.
int i = 10;
long l = 10l;
float f = 10.0f;
double d = 10.0;

// Números longos (Java 7).
int bilhao = 1_000_000_000;

// Diferentes bases (binário - Java 7).
int binario = 0b0001_0000;
int octal = 020;
int decimal = 16;
int hexa = 0x10;
System.out.println(binario + ", " + octal + ", " +
decimal + ", " + hexa); // 16, 16, 16, 16
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipos primitivos reais

- Também conhecido como “ponto flutuante”.

Tipo	Tamanho	Alcance
`float`	4 bytes	aprox. ±3.402823 E+38F
`double`	8 bytes	aprox. ±1.7976931348623 1 E+308

```
// A separação decimal é feita com "."
float f = 1.0f / 3.0f;
double d = 0.1e1 / 3.; // 0.1e1 = 0,1 x 10¹

// Note a diferença na precisão.
System.out.println(f); // 0.33333334
System.out.println(d); // 0.3333333333333333
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Precisão de ponto flutuante

- A precisão se refere ao número de casas decimais, não ao tamanho (ex.: 3.4 x 10^{38});
- Quando a precisão é fundamental, devemos usar a classe `BigDecimal` e não os tipos primitivos.

```
double d = 1.230000875E+2;
System.out.println(d); // 123.0000875

// Converte de double para float.
float f = (float)d;
System.out.println(f); // 123.000084
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipo primitivo caractere

- Para caracteres, Java usa o tipo `char`:
  - *Segue o padrão Unicode;*
  - *Ocupa 2 bytes;*
  - *Representa 32.768 caracteres diferentes;*
  - *Permite a construção de software internacionalizado;*
  - *Depende também do suporte do SO.*
- Representação entre aspas simples:
  - *'a', '5', '\t', '\u0061', etc.*

http://www.joelonsoftware.com/articles/Unicode.html

Um caractere é um inteiro

- Por sua representação interna ser um número inteiro (código Unicode), um `char` pode funcionar como um inteiro:

```
char c = 'a';
System.out.println("c: " + c); // c: a

c++;
System.out.println("c: " + c); // c: b

c = (char)(c * 100 / 80);
System.out.println("c: " + c); // c: z

int i = c;
System.out.println("i: " + i); // i: 122
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Caracteres especiais

Código	Significado
`\n`	Quebra de linha (*newline* ou *linefeed*)
`\r`	Retorno de carro (*carriage return*)
`\b`	Retrocesso (*backspace*)
`\t`	Tabulação (*horizontal tabulation*)
`\f`	Nova página (*form feed*)
`\'`	Aspas simples (apóstrofo)
`\"`	Aspas
`\\`	Barra invertida (“\”)
`\u233d`	Caractere Unicode de código 0x233d (hexadecimal)

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Cadeias de caractere

- Algumas linguagens definem um tipo primitivo *string* para cadeias de caracteres;
- Java não possui um tipo primitivo *string*;
- Em Java, *strings* são tipos compostos (objetos);
- Veremos mais adiante...

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tipo primitivo lógico (booleano)

- Valores verdadeiro (`true`) ou falso (`false`);
- Não existe equivalência entre valores lógicos e valores inteiros (como em C);

```
boolean b = true;
if (b) System.out.println("OK!"); // OK!

int i = (int)b; // Erro de compilação!
i = 1;
if (i) System.out.println("??"); // Idem!
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conversões entre tipos numéricos

- Tipos numéricos podem se misturar em operações, seguindo as seguintes regras:
  - *Se um dos operandos for `double` e o outro não, será convertido para `double`;*
  - *Senão, se um dos operandos for `float` e o outro não, será convertido para `float`;*
  - *Senão, se um dos operandos for `long` e o outro não, será convertido para `long`;*
  - *Nos demais casos, os operandos serão sempre convertidos para `int`, caso já não o sejam.*

Conversões entre tipos numéricos

```
byte b = 2; short s = 4; int i = 8;
long l = 16; float f = 1.1f; double d = 9.9;

d = s + l + d;
```

long

double

`float` é o único que precisa especificar o tipo do valor literal.

```
// i + b resulta em int, convertido pra long;
l = i + b;

// Erro: b + s resulta em int!
s = b + s;
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conversões entre tipos numéricos

- Conversões de um tipo menor para um tipo maior ocorrem automaticamente;
- Podemos forçar conversões no sentido contrário, sabendo das possíveis perdas;
- Utilizamos o operador de coerção (*cast*):

```
double x = 9.997;
int nx = (int)x;
System.out.println(nx); // 9

nx = (int)Math.round(x);
System.out.println(nx); // 10
```

Cast: to give a shape to (a substance) by pouring in liquid or plastic form into a mold and letting harden without pressure.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Operadores aritméticos

Símbolo	Significado	Exemplo
+	Adição	a + b
-	Subtração	a - b
*	Multiplicação	a * b
/	Divisão	a / b
%	Resto da divisão inteira	a % b
-	(Unário) inversão de sinal	-a
+	(Unário) manutenção de sinal	+a
++	(Unário) Incremento	++a ou a++
--	(Unário) Decremento	--a ou a--

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Operadores relacionais

Símbolo	Significado	Exemplo
==	Igual	a == b
!=	Diferente	a != b
>	Maior que	a > b
>=	Maior que ou igual a	a >= b
<	Menor que	a < b
<=	Menor que ou igual a	a <= b

- Observações:
  - *Os parâmetros devem ser de tipos compatíveis;*
  - *Não confunda = (atribuição) com == (igualdade).*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Operadores lógicos

Símbolo	Significado	Exemplo
&&	AND (E)	a && b
&	AND sem curto-circuito	a & b
||	OR (OU)	a || b
|	OR sem curto-circuito	a | b
^	XOR (OU exclusivo)	a ^ b
!	NOT (NÃO, inversão de valor, unário)	! a

- Observações:
  - *Só operam sobre valores lógicos;*
  - *Podem ser combinados em expressões grandes.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Curto-circuito

```
int x = 10, y = 15, z = 20;

// (z > y) não é avaliado.
if ((x > y) && (z > y)) { /* ... */ }

// (z == y) não é avaliado.
if ((x == 10) || (z == y)) { /* ... */ }

// Ambos são avaliados.
if ((x > y) & (z > y)) { /* ... */ }

// Ambos são avaliados.
if ((x == 10) | (z == y)) { /* ... */ }
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Operador de atribuição

- Usado para alterar o valor de uma variável:
  - ✓ `x = 10 * y + 4;`
- Várias atribuições podem ser feitas em conjunto:
  - ✓ `x = y = z = 0;`
- O lado direito da atribuição é sempre calculado primeiro, seu resultado é armazenado na variável do lado esquerdo:
  - ✓ `int x = 5;`
  - ✓ `x = x + 2;`

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Operadores bit a bit

- Operam em variáveis inteiras, bit a bit:

Símbolo	Significado	Exemplo
&	AND (E)	a & b
|	OR (OU)	a | b
^	XOR (OU exclusivo)	a ^ b
~	NOT (NÃO, unário)	~ a
>>	Deslocamento de bits para a direita	a >> b
<<	Deslocamento de bits para a esquerda	a << b

Atribuição composta

- Une-se um operador binário com o sinal de atribuição:

Expresão	Equivale a
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Expresão	Equivale a
x &= y	x = x & y
x |= y	x = x | y
x ^= y	x = x ^ y
x >>= y	x = x >> y
x <<= y	x = x << y

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Incremento e decremento

- Somar / subtrair 1 de uma variável inteira é tão comum que ganhou um operador só para isso:
  - ✓ *++x e x++ equivalem a x = x + 1;*
  - ✓ *--x e x-- equivalem a x = x – 1.*
- Quando parte de uma expressão maior, a forma prefixada é diferente da pós-fixada:

```
int x = 7;
int y = x++; // y = 7, x = 8.

x = 7;
y = ++x;     // y = 8, x = 8.
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Operador ternário

- Forma simplificada de uma estrutura `if – else` (que veremos mais adiante);
- Produz um valor de acordo com uma expressão:
  - ✓ *<expressão> ? <valor 1> : <valor 2>*
  - ✓ *Se <expressão> for `true`, o resultado é <valor 1>, do contrário o resultado é <valor 2>.*

```
int x = 7;

int y = (x < 10) ? x * 2 : x / 2;
System.out.println("y = " + y); // y = 14
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Precedência de operadores

- Podemos combinar expressões:
  - ✓ `x = a + b * 5 - c / 3 % 10;`
- Atenção à precedência de operadores!
  1) `b * 5;`
  2) `c / 3;`
  3) (resultado de `c / 3`) `% 10;`
  4) Da esquerda para a direita.
- Podemos usar parênteses para modificá-la:
  - ✓ `x = (a + b) * (5 - (c / 3)) % 10.`

Na dúvida, utilize parênteses.

Precedência de operadores

Operator Precedence

Operators	Precedence
postfix	*expr++ expr--*
unary	*++expr --expr +expr -expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Escopo de variável

- Variáveis podem ser declaradas em qualquer ponto do programa;
- O escopo define onde a variável é visível (onde podemos ler/atribuir seu valor);
- O escopo de uma variável vai do "{" anterior à sua declaração até o próximo "}".

```
int i = 10;
if (i > 0) {
  int j = 15;
  System.out.println(i + j); // 25
}
j = i + j; // Erro: variável fora de escopo!
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Nomes de variáveis

- Deve ser iniciado por uma letra, _ ou $;
- Seguido de letras, números, _ ou $;
- Podem ter qualquer tamanho;
- Não podem ser igual a uma palavra reservada;
- Lembre-se sempre: Java é *case sensitive*.

Nomes válidos	Nomes inválidos
a1	1a
total	total geral
$_$$	numero-minimo
_especial	tico&teco
Preço	void

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Palavras reservadas

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Constantes

- Para declarar constantes, basta usar a palavra-chave final:

```
final double CM_POR_POLEGADA = 2.54;
CM_POR_POLEGADA = 2.55; // Erro!

double larguraPol = 13.94;
double larguraCm = larguraPol * CM_POR_POLEGADA;

System.out.println(larguraCm);
```

Convenção de código: constantes são escritas em CAIXA_ALTA.

Controle de Fluxo

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Controle de fluxo

- Somente programas muito simples são estritamente sequenciais:
  - *Leia três notas de um aluno: N1, N2 e N3;*
  - *Calcule a média do aluno pela função M = (N1 + N2 + N3) / 3;*
  - *Se a média M for maior ou igual a 7:*
  - *Imprima "Aluno aprovado";*
  - *Se a média não for maior ou igual a 7:*
  - *Calcule a nota que o aluno precisa para passar pela função PF = (2 * F) – M;*
  - *Imprima "Aluno em prova final, nota mínima: <PF>".*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Controle de fluxo

- LPs imperativas geralmente possuem as seguintes estruturas de controle de fluxo:
  - *Estruturas de desvio de fluxo: desvia o fluxo e quebra a estrutura sequencial. Pode ser condicional ou incondicional. Em Java temos* `if` *e* `switch`*;*
  - *Estruturas de repetição simples: repete um ou mais comandos em laços ou loops um número fixo de vezes. Em Java, temos a diretiva* `for`*;*
  - *Estruturas de repetição condicional: semelhante à repetição simples, mas um número indefinido de vezes, associada a uma condição. Em Java temos* `while` *e* `do while`*.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Estruturas de desvio de fluxo

- Desviam o código para um outro trecho, ao invés de prosseguir para a linha seguinte;
- Há dois tipos de desvio de fluxo:
  - *Desvio condicional (`if`, `switch`);*
  - *Desvio incondicional (`goto`).*
- Java não possui `goto`. Possui dois casos específicos de desvio incondicional com `break` e `continue`.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conceitos básicos: diretivas

- Diretivas (*statements*) são as instruções que uma LP fornece para a construção de programas;
- Deve haver alguma forma de separar uma diretiva da outra:
  - *Cada uma em uma linha;*
  - *Separadas por um caractere (ex.: “.”, “;”, etc.).*
- Java herdou de C/C++ a separação com “;”:

```
diretiva1;
diretiva2; diretiva3;
diretiva4;
...
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conceitos básicos: blocos

- Diretivas podem ser dispostas em blocos;
- Um bloco recebe o mesmo tratamento de uma diretiva individual;
- Java também herdou de C/C++ a definição de blocos com "{" e "}":

```
{
  diretiva1;
  diretiva2; diretiva3;
  diretiva4;
  ...
}
...
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Estruturas de desvio de fluxo

- Desviam o código para um outro trecho, ao invés de prosseguir para a linha seguinte;
- Há dois tipos de desvio de fluxo:
  - *Desvio condicional (`if`, `switch`);*
  - *Desvio incondicional (`goto`).*
- Java não possui `goto`. Possui dois casos específicos de desvio incondicional com `break` e `continue`.

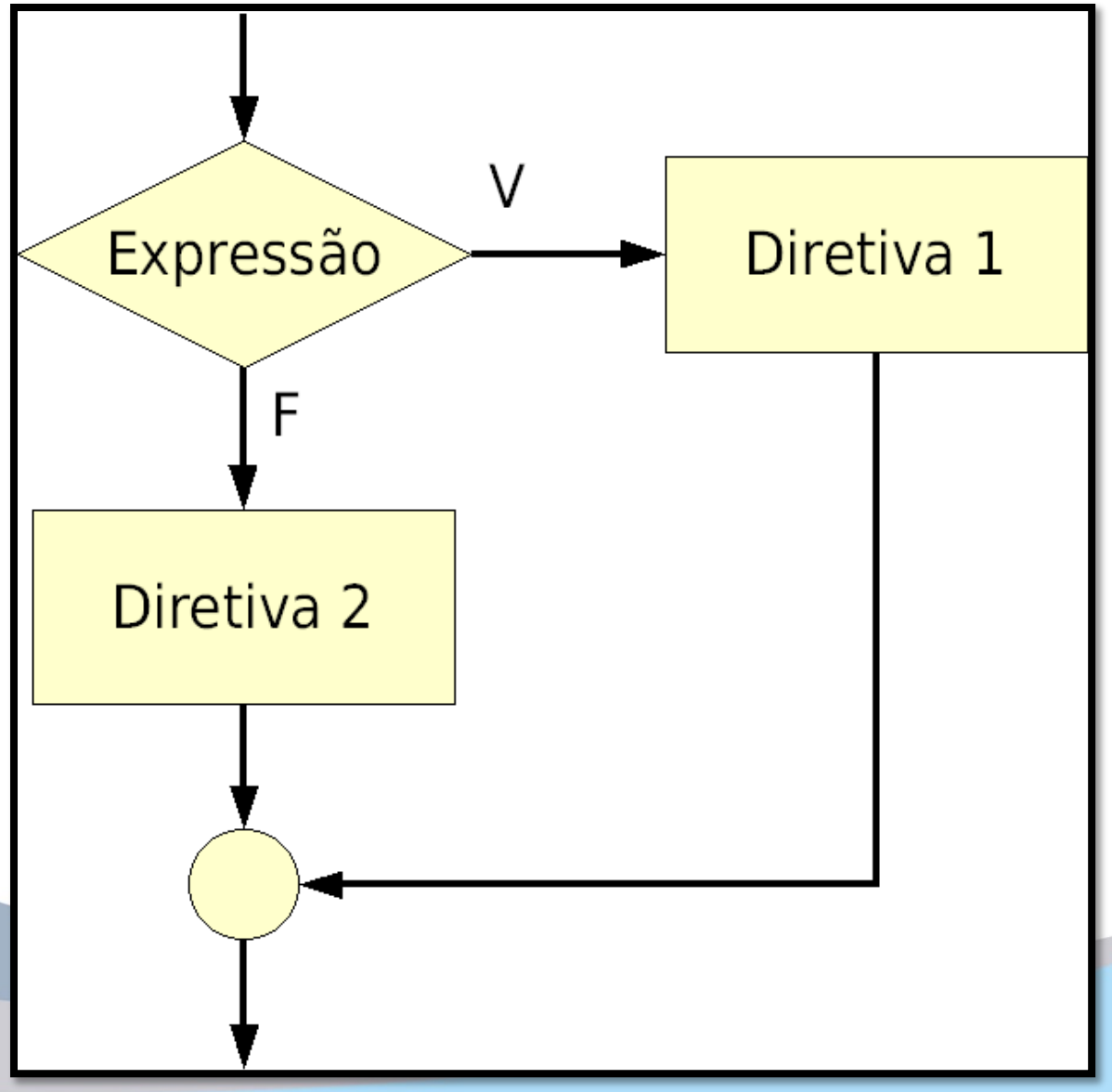
UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Desvio condicional com `if`

```
if ([expressão])
  [diretiva 1]
else
  [diretiva 2]
```

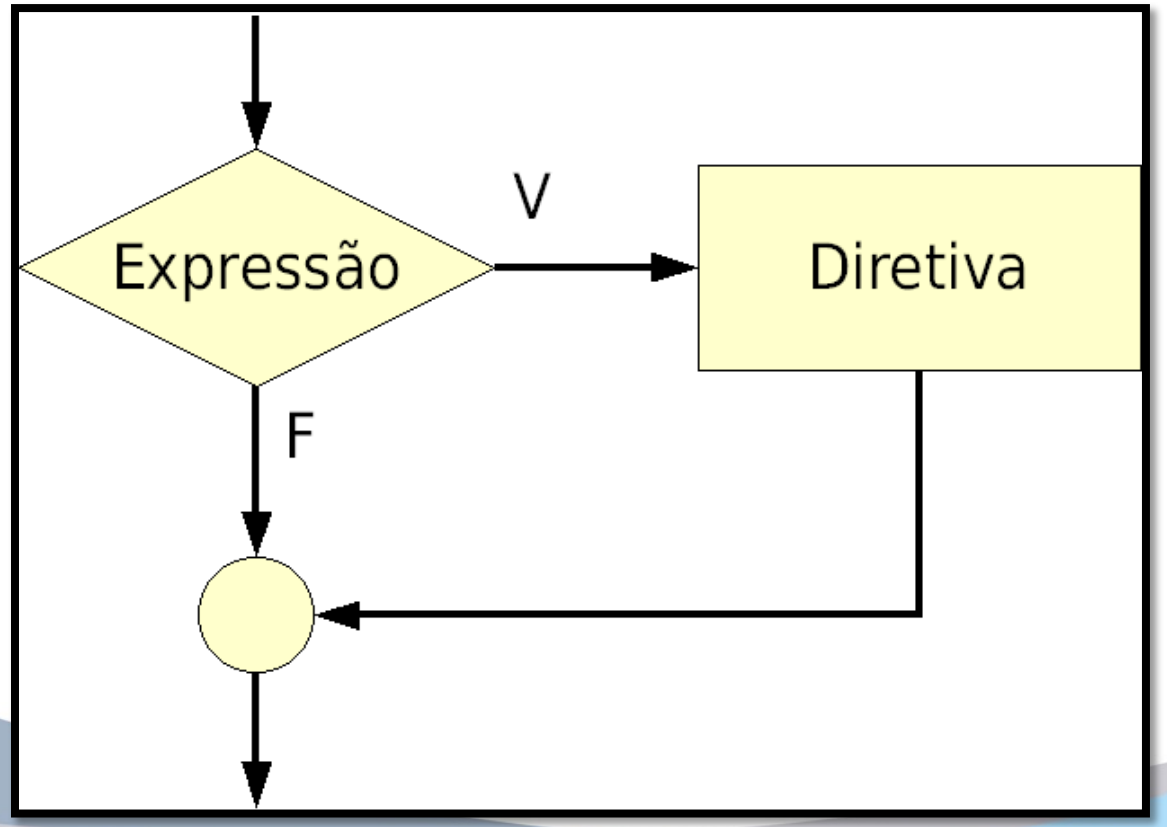
- *[expressão]: expressão lógica avaliada (deve retornar valor do tipo `boolean`);*
- *[diretiva 1]: diretiva ou bloco de diretivas executadas se a condição retornar `true`;*
- *[diretiva 2]: diretiva ou bloco de diretivas executadas se a condição retornar `false`.*

Funcionamento do if



A parte do else é opcional

```
if ([expressão])
  [diretiva]
```



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Encadeamento de ifs

- O `if` é uma diretiva como qualquer outra;
- Podemos colocá-lo como [diretiva 2], logo após o `else` (executada quando expressão é `false`):

```
if ([expressão])
  [diretiva 1]
else if ([expressão 2])
  [diretiva 2]
else if ([expressão 3])
  [diretiva 3]
...
else
  [diretiva N]
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exemplos

```
int x = 10, y = 15, z = 20;
boolean imprimir = true;

if ((x == 10) || (z > y)) {
  if (imprimir) System.out.println(x);
}
else if (x == 20) {
  z += x + y;
  if (imprimir) System.out.println(z);
}
else System.out.println("Não sei!");
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Desvio condicional com switch

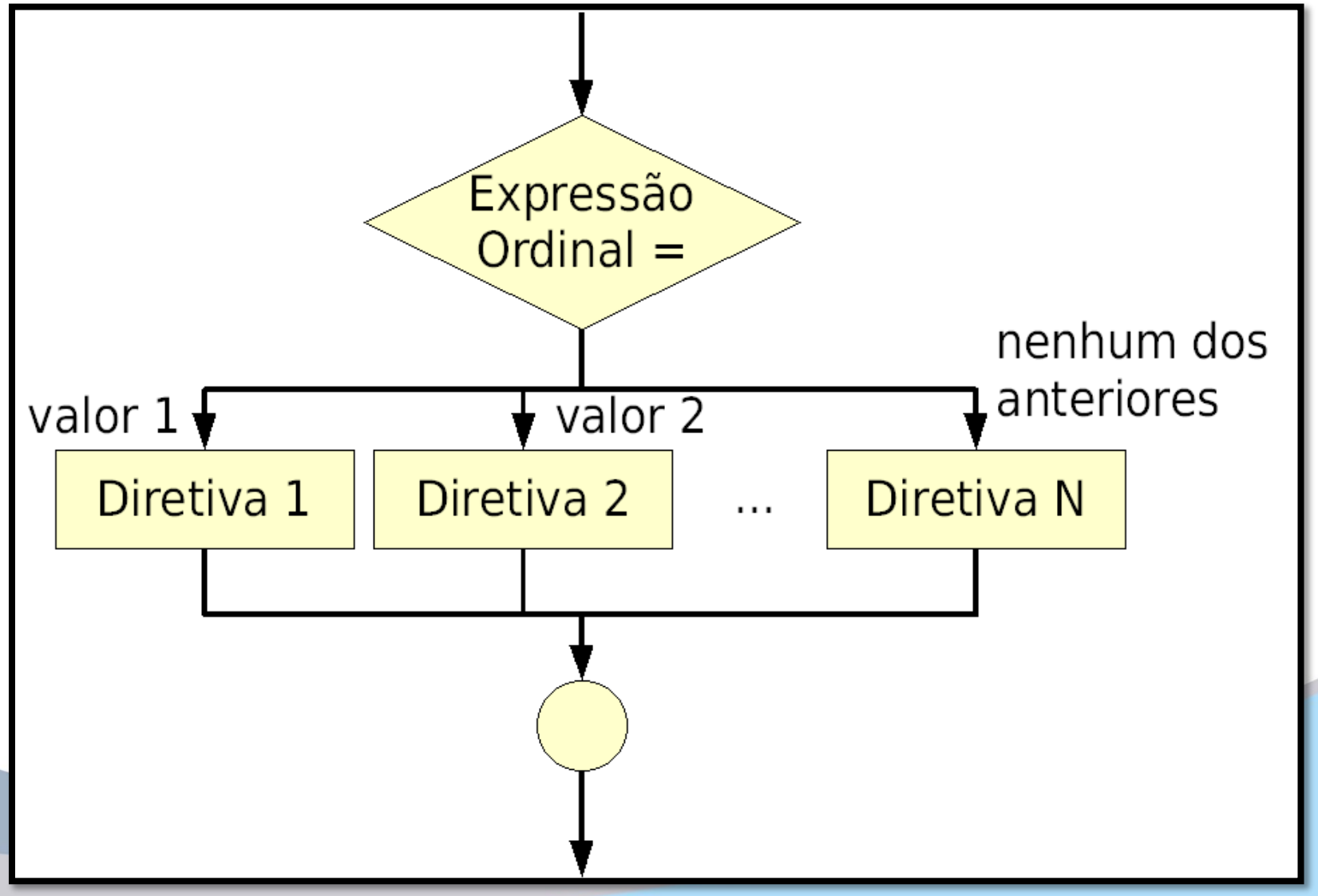
```
switch ([expressão ordinal ou string]) {
case [valor ordinal 1]:
  [diretiva 1]
  break;
case [valor ordinal 2]:
  [diretiva 2]
  break;
...
default:
  [diretiva N]
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Desvio condicional com switch

- [expressão ordinal ou string]: expressão que retorna um valor de algum tipo discreto (`int`, `char`, etc.) ou *string* (a partir do Java 7);
- [valor ordinal X]: um dos possíveis valores que a expressão ordinal pode assumir (deve ser do mesmo tipo);
- [diretiva X]: diretiva ou conjunto de diretivas (não é necessário abrir um bloco) executado se a expressão ordinal for igual ao [valor ordinal X].

Funcionamento do switch



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Características

- É possível construir um `if` equivalente ao `switch`, mas este último tem desempenho melhor;
- Ter uma opção `default` é opcional;
- O fluxo é desviado para o *case* apropriado e continua dali até encontrar um `break` ou o fim do `switch`.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exemplo

```
switch (letra) {   // letra é do tipo char
case 'a':
case 'A': System.out.println("Vogal A");
  break;
case 'e': case 'E':
  System.out.println("Vogal E");
  break;
/* ... */
case 'u': case 'U':
  System.out.println("Vogal U");
  break;
default:
  System.out.println("Não é uma vogal");
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Estruturas de repetição simples

- Repetição de um trecho de código;
- Número fixo de repetições (sabe-se de antemão quantas vezes o trecho será repetido);
- Java dispõe da diretiva `for` (também herdada de C):

```
// Contar de 1 a 10:
for (int i = 1; i <= 10; i++)
  System.out.println(i);
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

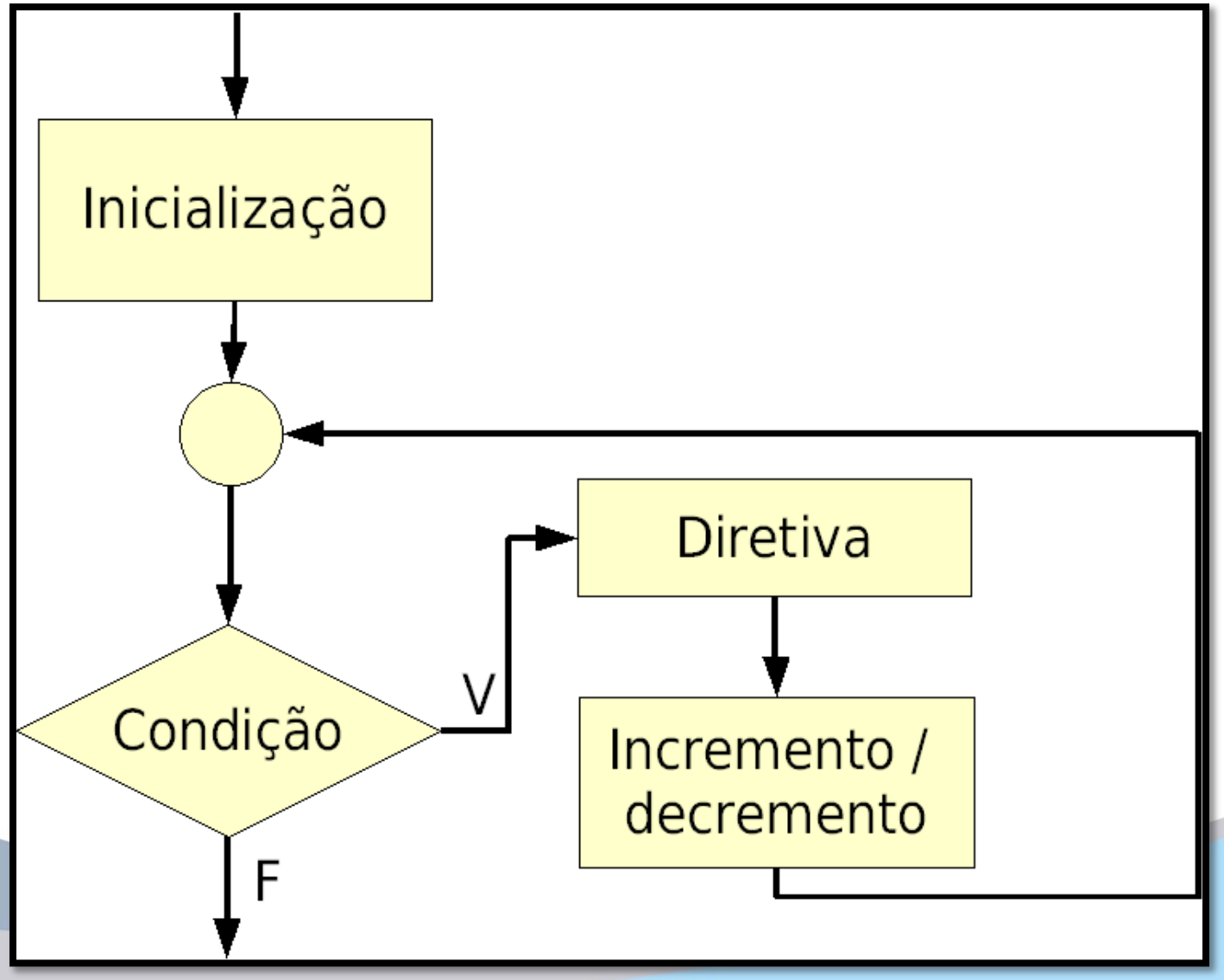
Estrutura do for

```
for ([início]; [condição]; [inc/dec])
  [diretiva]
```

- *[início]: diretiva executada antes do laço começar (geralmente, atribuir o valor inicial do contador);*
- *[condição]: expressão de condição de parada do laço (geralmente, comparação com o valor final);*
- *[inc/dec]: diretiva executada após cada iteração do laço (geralmente usada para incrementar ou decrementar o contador);*
- *[diretiva]: diretiva ou bloco de diretivas executadas em cada iteração do laço.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Funcionamento do for



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Os campos do for são opcionais

```
// Conta até 10.
int i = 1;
for (; i < 10;) {
  System.out.println(i++);
}

// Preenche um vetor.
int[] v = new int[5];
for (int i = 0; i < 5; v[i] = i++);

// Loop infinito.
for (;;);
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Repetição condicional com o for

- Em várias linguagens de programação, o for (ou similar) serve somente para repetição simples:

```
para i de 1 até 10 faça
  Escreva i
fim_para
```

- Em Java pode-se usar para fazer repetição condicional:

```
boolean achou = false;
for (int i = 0; (! achou); i++) {
  /* ... */
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Inicialização / incremento múltiplo

- Podemos efetuar múltiplas diretivas na inicialização e no incremento, se necessário, separando com vírgulas:

```
for (int i = 1, j = i + 10; i < 5; i++, j = i * 2) {
  System.out.println("i= " + i + " j= " + j);
}
```

- Claro que também podemos ter condicionais grandes (usando operadores lógicos):

```
for (int i = 0; (i < 5) && (! achou); i++) {
   /* ... */
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Estruturas de repetição condicional

- Repetição de um trecho de código;
- Número indeterminado de repetições, depende de uma condição (expressão lógica);
- Java dispõe da diretiva `while` e `do while` (sintaxe também herdada de C/C++):

```
// Contar de 1 a 10:
int i = 1;
while (i <= 10) System.out.println(i++);

i = 1;
do System.out.println(i++); while (i <= 10);
```

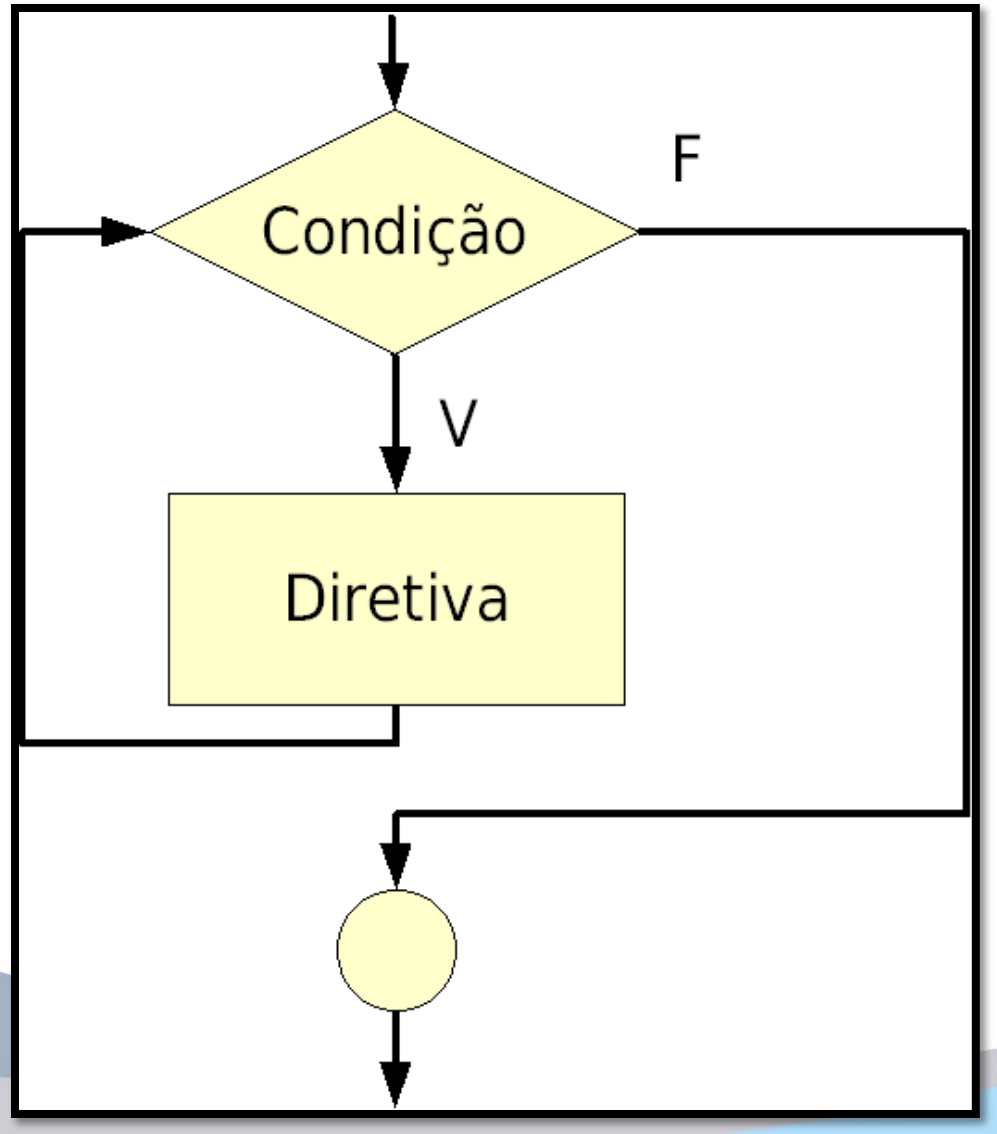
UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Estrutura de while e do while

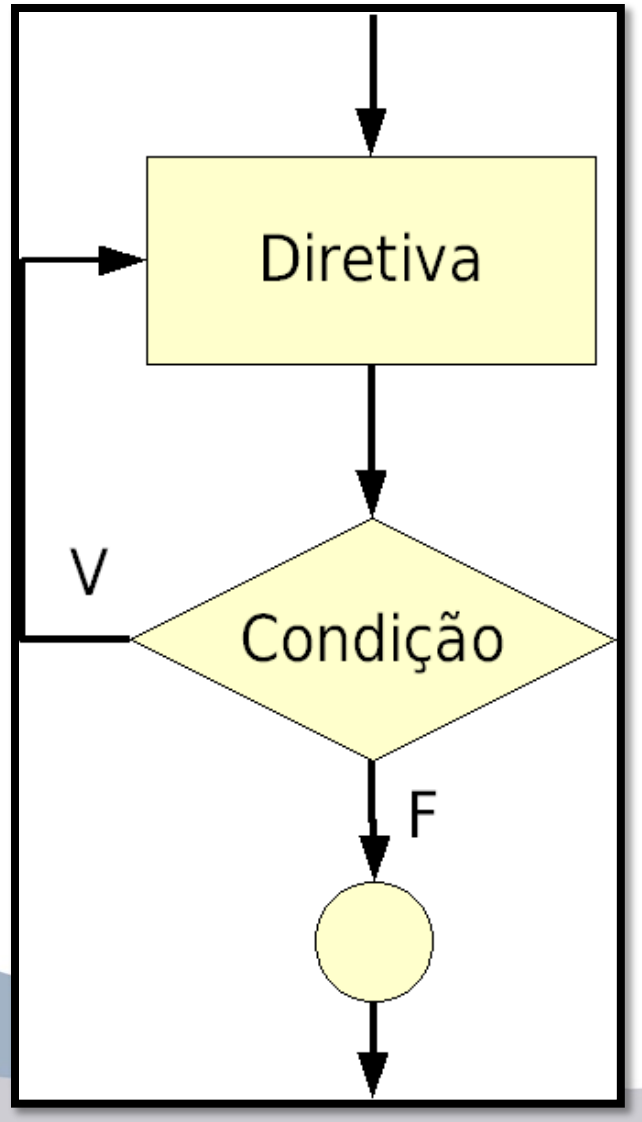
```
while ([condição]) [diretiva]

do [diretiva] while ([condição])
```

- *[condição]: expressão de condição de parada do laço (expressão lógica);*
- *[diretiva]: diretiva ou bloco de diretivas executadas em cada iteração do laço.*

Funcionamento do while



Funcionamento do do while



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Características

- `while` avalia a condição antes da diretiva, podendo nunca executá-la;
- `do while` só avalia depois, certamente executando a diretiva ao menos uma vez;
- O programador deve garantir que a condição se torna falsa em algum momento na diretiva, do contrário poderá gerar um *loop* infinito.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exemplos

```
int i = 0, j = 10;
while (i <= j) {
  System.out.println(i + " - " + j); i++; j--;
}

// Executará ao menos 1 vez!
do {
  System.out.println(i + " - " + j);
  i++; j--;
} while (i <= j);

// Podemos fazer um for equivalente!
for (i = 0, j = 10; i <= j; i++, j--) {
  System.out.println(i + " - " + j);
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Desvios incondicionais

- O uso de desvios incondicionais com `goto` levam a programas de baixa legibilidade;
- Java só tem dois casos específicos de desvios incondicionais: `break` e `continue`;
- Podem ser usados dentro de laços ou dentro da estrutura `switch` (como já vimos):
  - *`break` sai da estrutura (laço ou `switch`);*
  - *`continue` vai para a próxima iteração (somente laços).*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exemplos

```
while (ano < 2001) {
  saldo = (saldo + salario) * (1 + juros);
  if (saldo >= saldoLimite) break;
  ano++;
}

for (int i = 0; i < 100; i++) {
  if (i % 9 == 0) continue;
  System.out.println(i);
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

break e continue com rótulos (labels)

- Rótulos podem indicar de qual laço queremos sair ou continuar a próxima iteração;
- Podem ser usados apenas em laços e só fazem sentido em laços aninhados.

```
externo:
for (int i = 1; i < 100; i++) {
  for (j = 5; j < 10; j++) {
    int x = i * j - 1;
    if (x % 9 != 0) break;
    if (x % 15 == 0) break externo;
    System.out.println(x);
  }
}
```

Entrada e Saída Básica

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Entrada e saída de dados básica

- Toda linguagem de programação deve prover um meio de interação com o usuário;
- O meio mais básico é o uso do console, com entrada de dados pelo teclado e saída em texto;
- Outros meios são: interface gráfica (janelas), pela *Web*, comandos de voz, etc.*;*
- Veremos a forma de interação básica, pelo console:
  - *O "shell" ou "console" do Linux/Mac;*
  - *O "prompt de comando" do Windows.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Saída de dados pelo console

- Java usa o conceito de *stream*: um duto capaz de transportar dados de um lugar a outro;
- A classe `java.lang.System` oferece um *stream* padrão de saída chamado `out`;
  - *É um objeto da classe `java.io.PrintStream`, aberto e mantido automaticamente pela JVM;*
  - *Oferece vários métodos para impressão de dados: `print()`, `println()` e `printf()`.*
- Podemos trocar o dispositivo padrão de saída: `System.setOut(novaStream)`.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exemplos

```
// 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
for (i = 1; i < 10; i++) {
  System.out.print(i + ", ");
}
System.out.println(10);

String s = "Olá, Java!";
float valor = 45.67;
boolean teste = true;

System.out.println(s);          // Olá, Java!
System.out.print(valor);        // 45.67 (sem quebra)
System.out.println();           // Quebra de linha
System.out.println(teste);      // true
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Saída formatada

- A partir do Java 5, a função `printf()` do C foi colocada na classe `PrintWriter`;
  - *Facilita a migração de código C para Java;*
  - *É uma forma mais poderosa de formatar a saída;*
  - *O trabalho de formatação é, na verdade, feito pela classe `java.util.Formatter`.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Como funciona o `printf`

- Argumentos:
  - *Uma string de formatação, com códigos especiais;*
  - *Uma lista de argumentos a serem impressos.*
- Exemplos:

```
System.out.printf("Olá, Java!%n"); // Olá, Java!

// [   12] e [12345678]
System.out.printf("[%5d]%n", 12);
System.out.printf("[%5d]%n", 12345678);

double PI = 3.141592;
System.out.printf("[%6.6f]%n", PI); // [3.141592]
System.out.printf("[%6.4f]%n", PI); // [  3.1416]
```

Códigos de formatação

- Possuem a seguinte sintaxe:

```
%[i$][flags][tam][.prec]conversão
```

  - *i: índice do argumento (opcional);*
  - *flags: modificam o formato de saída (opcional);*
  - *tam: tamanho da saída em caracteres (opcional);*
  - *prec: precisão das casas decimais (opcional);*
  - *conversão: código de conversão (indica se é um texto, inteiro, real, booleano, etc. – obrigatório).*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Códigos de conversão

Código	Tipo do arg.	Descrição
'b' ou 'B'	Booleano	Imprime true ou false.
's' ou 'S'	Geral	Imprime como string (texto).
'c' ou 'C'	Caractere	O resultado é um caractere Unicode.
'd'	Inteiro	O resultado é formatado como número inteiro na base decimal.
'e' ou 'E'	Real (PF)	O resultado é formatado como número decimal em notação científica.
'f'	Real (PF)	O resultado é formatado como número decimal.
'g' ou 'G'	Real (PF)	Uma das duas opções acima (depende do valor).
'%'	-	O resultado é o caractere %.
'n'	-	O resultado é uma quebra de linha.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Mais exemplos

```
// 5, 00005
System.out.printf("%1$d, %1$05d%n", 5);

// Agora: 12 de Maio de 2006 - 04:29:42 PM.
System.out.printf("Agora: %te de %<tB de %<tY -
%<tr.%n", new Date());

// PI = 3.141592654
System.out.printf("PI = %.9f%n", Math.PI);

// PI = 3,141592654
Locale br = new Locale("pt", "BR");
System.out.printf(br, "PI = %.9f%n", Math.PI);

// Veja outros formatos na documentação!
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Entrada de dados pelo console

- A classe `java.lang.System` oferece um *stream* padrão de entrada chamado `in`;
  - *É um objeto da classe `java.io.InputStream`, aberto e mantido automaticamente pela JVM;*
  - *Seus métodos de leitura são muito primitivos, se comparados com os métodos de escrita que vimos;*
  - *Precisamos de outras classes que auxiliem na leitura.*
- Além da leitura por `System.in`, podemos ler também os argumentos passados na chamada do programa.
  - *Já vimos isso na classe `Eco.java`.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Leitura de argumentos

- O método `main()` possui a seguinte assinatura:

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

- O vetor `args` contém os dados passados como argumentos para o programa.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Eco.java novamente

```
public class Eco {

  // Método principal.
  public static void main(String[] args) {
    // i vai de 0 até o nº de argumentos.
    for (int i = 0; i < args.length; i++) {
      // Imprime o argumento na tela.
      System.out.print(args[i] + " ");
    }
    // Quebra de linha ao final do programa.
    System.out.println();

  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

A classe `java.util.Scanner`

- Poderoso meio de ler dados de qualquer *stream* de entrada, existente a partir do Java 5;
- Funcionamento:
  - *Quebra a informação em tokens de acordo com um separador (que pode ser configurado);*
  - *Lê uma informação de cada vez;*
  - *Converte para o tipo de dados adequado (quando possível).*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exemplos

```
// Lê do console.
Scanner scanner = new Scanner(System.in);

// Lê linha por linha e imprime o "eco".
while (scanner.hasNextLine()) {
  String s = scanner.nextLine();
  System.out.println("Eco: " + s);
}
// Quebra palavra por palavra.
while (scanner.hasNext()) {
  String s = scanner.next();
  System.out.println("Eco: " + s);
}
// Depois de usado deve ser fechado.
scanner.close();
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Mais exemplos

```
// Lê do console.
Scanner scanner = new Scanner(System.in);
// Lê números e imprime o dobro.
while (scanner.hasNextDouble()) {
  double d = scanner.nextDouble();
  System.out.println("Dobro: " + (d * 2));
}
// Separa os tokens usando vírgulas.
scanner.useDelimiter(",");
while (scanner.hasNext()) {
  String s = scanner.next();
  System.out.println("Eco: " + s);
}


scanner.close();
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exercitar é fundamental

- Apostila FJ-11 da Caelum:
  - *Seção 3.13, página 30 (exercícios simples);*
  - *Seção 3.14, página 31 (Fibonacci).*

Faremos exercícios em laboratório em breve...