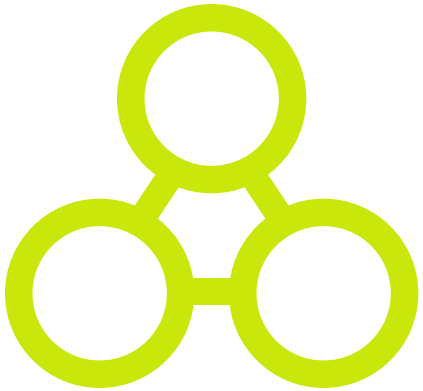
Desenvolvimento OO com Java

3 – Estruturas de Controle e Programação Básica



nemo

ontology & conceptual modeling research group

Vítor E. Silva Souza

(vitor.souza@ufes.br)

http://www.inf.ufes.br/~vitorsouza

Departamento de Informática

Centro Tecnológico

Universidade Federal do Espírito Santo

Objetivos da parte 3

- Apresentar as estruturas de controle de fluxo;
- Ensinar comandos de entrada e saída de dados básicos (teclado / tela);
- Desta forma:
  - Capacitar os alunos a escrever programas simples, utilizando os recursos da programação imperativa;
  - Concluir a parte básica (procedural) para partir para conceitos novos (orientação a objetos).

Controle de fluxo

- Um programa de computador (algoritmo) é como uma receita: uma sequência de instruções:
  - Leia um valor X;
  - Calcule Y como função de X: Y = X * 2;
  - Imprima o resultado.

- Característico da programação imperativa;
- Outros paradigmas são diferentes:
  - Ex.: em LPs funcionais, um programa é um conjunto de funções matemáticas.

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>”.

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`.

Outros mecanismos

- Mecanismos de modularização:
  - Divisão de um programa em funções e procedimentos (ou métodos);
  - O fluxo é desviado de uma função para outra.
- Tratamento de exceções:
  - Quando um erro ocorre, desvia o fluxo para o código de tratamento do erro (exceção);
- Ambos serão discutidos em maior profundidade posteriormente.

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;
...
```

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;
  ...
}
...
```

Estruturas de desvio de fluxo

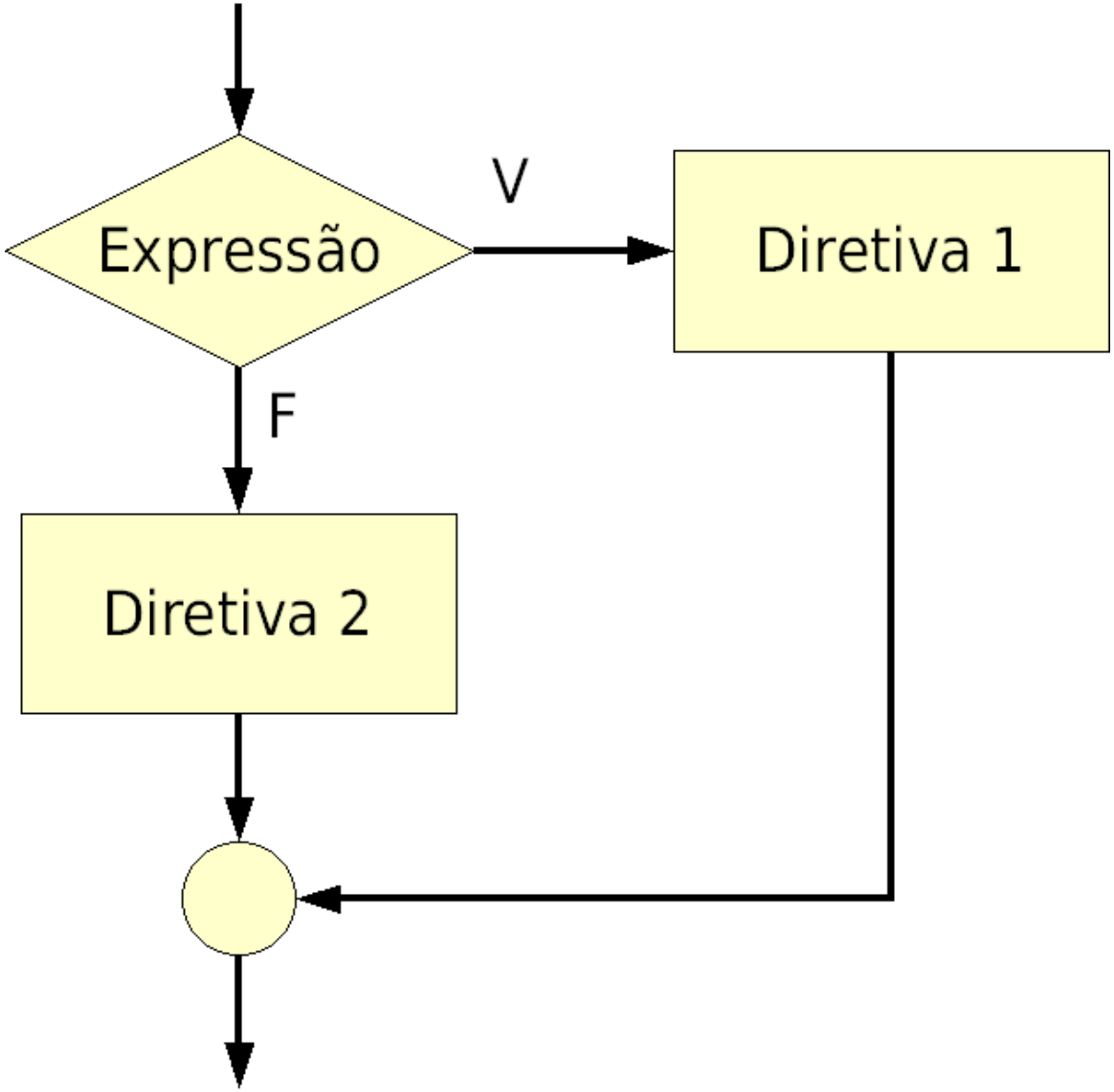
- Desviam o código para um outro trecho, ao invés de prosseguir para a linha seguinte;
- Há dois tipos de desvio de fluxo:
  - Desvio condicional (`if`, `switch`);
  - Desvio incondicional (`goto`).
- Java não possui `goto`. Possui dois casos específicos de desvio incondicional com `break` e `continue`.

Desvio condicional com `if`

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

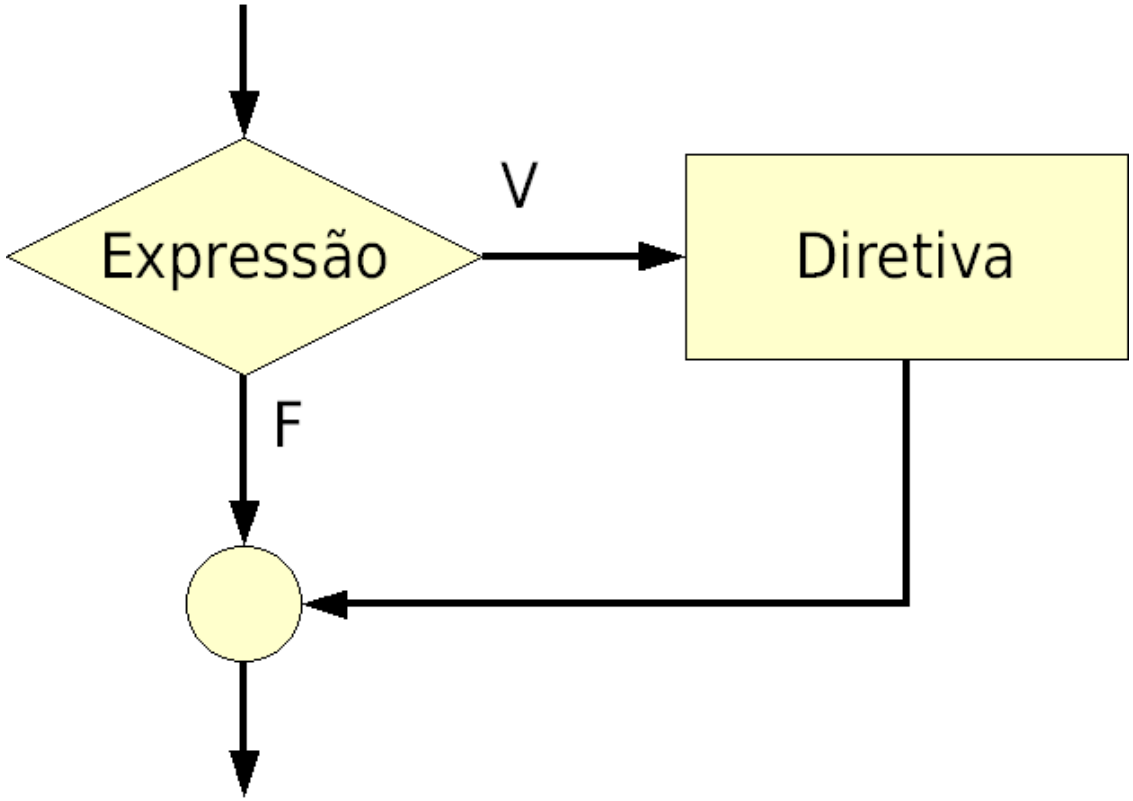
- [expressão]: expressão lógica avaliada (deve retornar valor do tipo `boolean`);
- [diretiva 1]: diretiva ou bloco de diretivas executadas se a condição retornar `true`;
- [diretiva 2]: diretiva ou bloco de diretivas executadas se a condição retornar `false`.

Funcionamento do if



A parte do else é opcional

```
if ([expressão])
  [diretiva]
```



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]
```

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!");
```

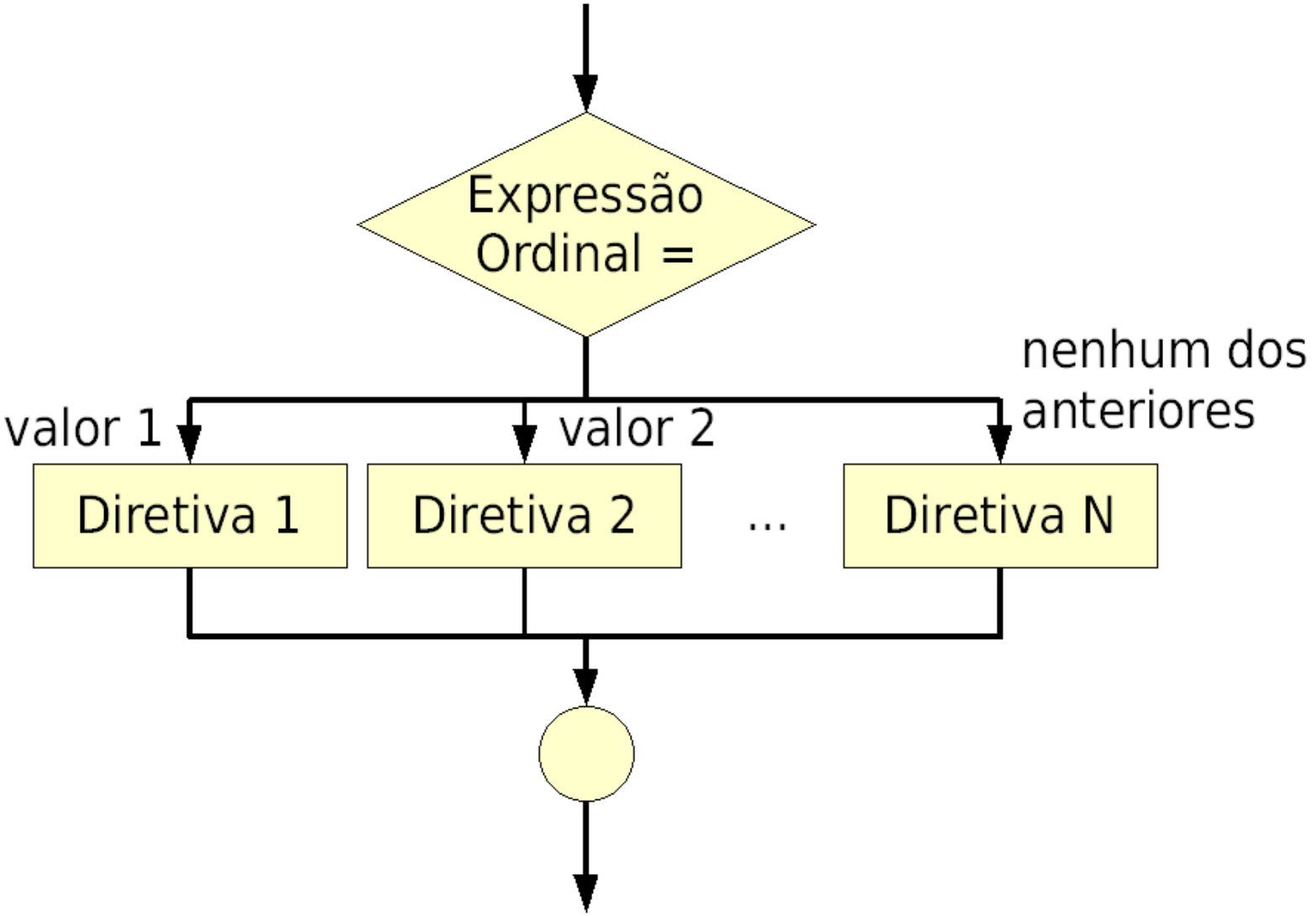
Desvio condicional com switch

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

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



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.

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");
}
```

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 (sintaxe também herdada de C):

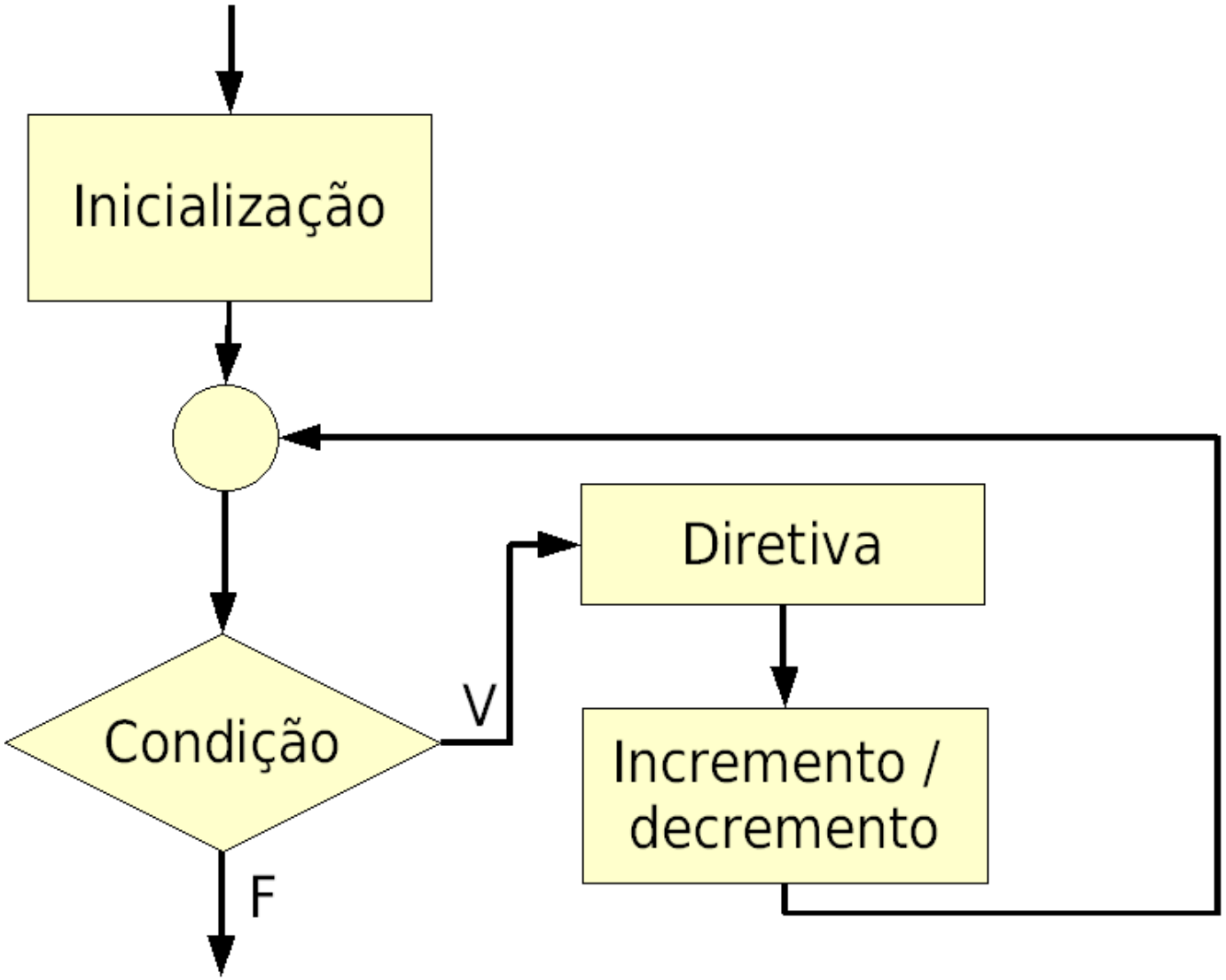
```
// Contar de 1 a 10:
for (int i = 1; i <= 10; i++)
  System.out.println(i);
```

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.

Funcionamento do for



Os campos do for são opcionais

```java
// 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 (;;);
```

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++) {
  /* ... */
}
```

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++) {
   /* ... */
}
```

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++):

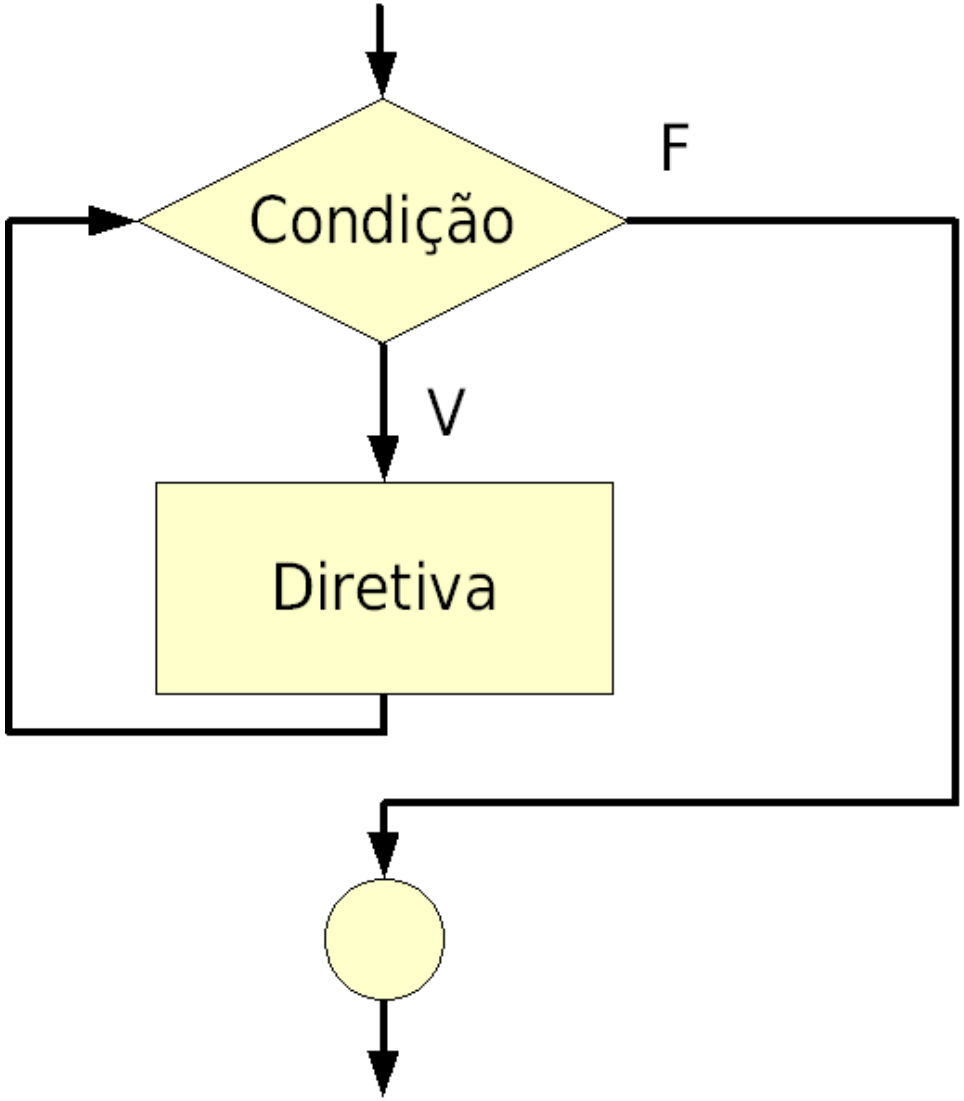
```
// Contar de 1 a 10:
int i = 1;
while (i <= 10) System.out.println(i++);

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

Estrutura de while e do while

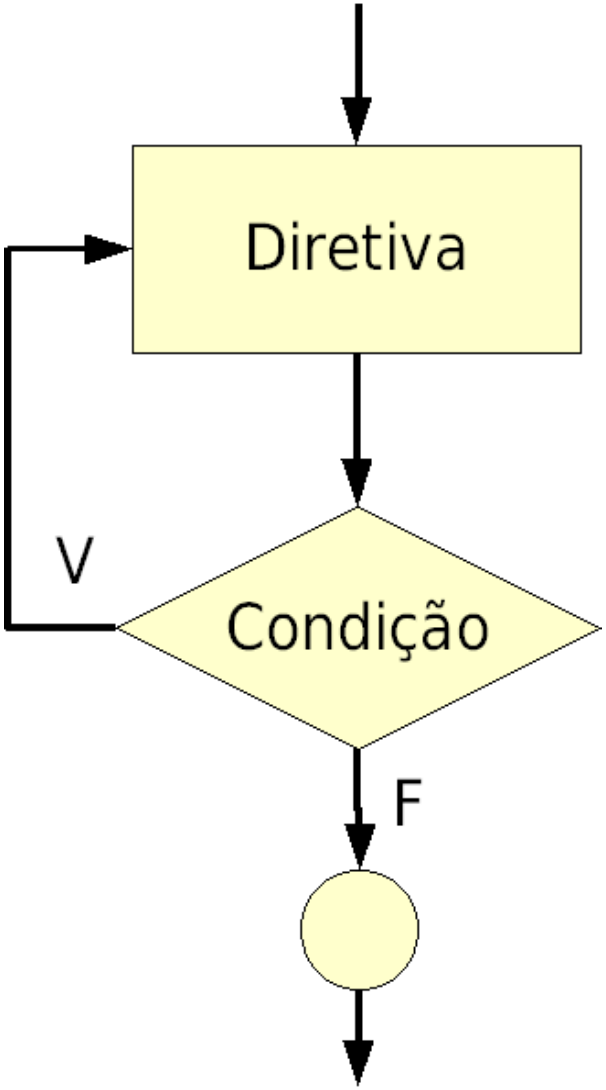
```
while ([condição]) [diretiva]

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



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.

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);
}
```

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).

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);
}
```

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 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.*;*
- Aprenderemos agora a forma de interação básica, pelo console:
  - O “shell” ou “console” do Linux/Mac;
  - O “prompt de comando” do Windows.

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)`.

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
```

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`.

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).

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.

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!
```

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`.

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).

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();
```

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();
```

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.

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();

  }
}
```

http://nemo.inf.ufes.br/