Estruturas de Controle em Java

Prof. Omero Francisco Bertol

(omero@utfpr.edu.br)

Estruturas de Controle

Um programa de computador é uma sequência de instruções organizadas de forma a produzir a solução de um determinado problema; o que representa uma das habilidades básicas da programação, que é a sequenciação.

Naturalmente, as instruções de um programa são executadas em sequência, o que se denomina fluxo sequencial de execução. Mas, em inúmeras circunstâncias, é necessário executar as instruções de um programa em uma ordem diferente da estritamente sequencial.

Tais situações são caracterizadas pela necessidade da repetição de instruções individuais ou conjuntos de instruções, e também pelo desvio do fluxo de execução, tarefas que podem ser realizadas por meio das “estruturas de controle” da linguagem.

Definições: Diretivas e Blocos (1/2)

Formalmente, as “instruções”, ou comandos de um programa são chamadas diretivas (*statements*).

Em Java, tal como nas linguagens C/C++ e Pascal, as diretivas são separadas umas das outras pelo símbolo de pontuação “;” (ponto-e-vírgula), sendo possível existir várias diretivas numa mesma linha, desde que separadas por um ponto-e-vírgula.

Veja os exemplos hipotéticos colocando “N” diretivas sequencialmente:

```
diretiva1;
diretiva2;
...
diretivaN;
```

```
diretiva1; diretiva2; diretiva3;
...
diretivaN;
```

Definições: Diretivas e Blocos (2/2)

As diretivas podem ser tratadas individualmente ou como um conjunto, que é denominado bloco. Um bloco em Java é um grupo de diretivas delimitadas por chaves: { diretivas }.

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

Assim, as estruturas de controle podem tanto operar sobre uma diretiva individual como sobre um bloco de diretivas. Outra consequência da criação de blocos é que as estruturas de controle podem ser aninhadas de forma coletiva, isto é, dentro de um bloco podem existir várias estruturas de controle contendo, por sua vez, seus próprios blocos de diretivas.

Classe Scanner (1/4)

Quem programa em linguagens estruturadas, como C e Pascal, e está aprendendo Java, depara-se com a seguinte situação: como atribuir valores para uma variável usando o teclado?

Em **C**, por exemplo, tem-se a função **scanf()** e em **Pascal**, o procedimento **readln()**.

No Java, a partir do Java 1.5 ou J2SE 5, que recebeu o codinome "Tiger", está disponível a classe **Scanner** do pacote **java.util**. Essa classe implementa as operações de entrada de dados pelo teclado no console.

Classe Scanner (2/4)

Para utilizar a classe Scanner em uma classe Java deve-se proceder da seguinte maneira:

1. importar o pacote java.util:

```
import java.util.Scanner;
```

2. Instanciar e criar um objeto Scanner:

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

3. Lendo valores através do teclado:

   3.1 Lendo um valor inteiro: `int n = ler.nextInt();`

   3.2 Lendo um valor real: `float preco = ler.nextFloat();`

   3.3 Lendo um valor real: `double salario = ler.nextDouble();`

   3.4 Lendo uma String: `String palavra = ler.next();`

   usado na leitura de palavras simples, ou seja, que não usam o caractere de espaço (ou barra de espaço)

   3.5 Lendo uma String: `String palavra = ler.nextLine();`

   usado na leitura de palavras compostas, por exemplo: Pato Branco

Classe Scanner (3/4)

```
import java.util.Scanner;

public class Tabuada {

  public static void main(String args[]) {
    Scanner ler = new Scanner(System.in);

    while (true) {
      System.out.println("Informe o nro, (-1) para encerrar:");
      int n = ler.nextInt();

      if (n == -1)
         break;

      System.out.println();
      for (int i=1; i<=10; i++)
        System.out.printf("%2d * %d = %3d\n", i, n, (i*n));

      System.out.println();
    }
  }

}
```

1

2

3.1

Classe Scanner (4/4)

Na leitura consecutiva de valores numéricos e String deve-se esvaziar o buffer do teclado antes da leitura do valor String, por exemplo:

```
System.out.println("Nro Inteiro:");
int n = ler.nextInt();
```

3.1

```
// esvazia o buffer do teclado
ler.nextLine();
```

```
System.out.println("String:");
String palavra = ler.nextLine();
```

3.5

Lendo um Caractere

Para ler um caractere deve-se utilizar o método “read()” do pacote de classes **System.in**:

```
public class Exemplo {

    public static void main(String args[])
      throws Exception {

      char sexo;

      System.out.println("Informe o Sexo (M/F: ");

      sexo = (char)System.in.read();

      if ((sexo == 'M') || (sexo == 'm'))
         System.out.println("(masculino).");
      else System.out.println("(feminino).");
    }
}
```

Método print() e println()

Funções e impressão na saída principal (System.out)

- System.out.print(“Texto a ser impresso”);
  - Imprime e deixa o cursor na mesma linha.

- System.out.println(“Texto a ser impresso”);
  - Imprime e passa o cursor para a próxima linha.

```
public class Main {
	public static void main(String[] args) {
		System.out.println("Olá mundo ");
	}
}
```

Método printf()

Outra novidade no Java 1.5 foi a incorporação da função "printf()" do C como um método do pacote de classes **System.out**.

Código em Java usando o método printf()	Resultado
• Tabuada de "n" = 7: `for (int i=1; i<=10; i++)` `  System.out.printf("%2d * %d = %3d\n", i, n, (i*n));` ***obs:*** ***%2d*** *– será substituido por um valor decimal usando 2 posições da tela.* ***\n*** *– pula uma linha*	1 * 7 = 7 2 * 7 = 14 3 * 7 = 21 4 * 7 = 28 5 * 7 = 35 6 * 7 = 42 7 * 7 = 49 8 * 7 = 56 9 * 7 = 63 10 * 7 = 70
• Imprimindo o nome dos meses do ano: `String nomeMes[] = {"Janeiro", "Fevereiro", "Março",` `  "Abril", "Maio", "Junho", "Julho", "Agosto",` `  "Setembro", "Outubro", "Novembro", "Dezembro"};` `System.out.println("Mês- Nome do Mês");` `System.out.println("================");` `for (int i=0; i<12; i++)` `  System.out.printf(" %0,2d- %s\n", (i+1), nomeMes[i]);` ***obs.*** ***0,2*** *– coloca zeros a esquerda, por exemplo: 01, 02, 03, 04, ..., 09, 10*	Mês- Nome do Mês ================ 01- Janeiro 02- Fevereiro 03- Março 04- Abril 05- Maio 06- Junho 07- Julho 08- Agosto 09- Setembro 10- Outubro 11- Novembro 12- Dezembro
• Imprimindo os elementos de um vetor de tamanho 10: `for (int i=0; i<n; i++)` `  System.out.printf("v[%d]= %2d.\n", i, v[i]);`	v[0]= 84. v[1]= 60. v[2]= 38. v[3]= 32. v[4]= 69. v[5]= 9. v[6]= 37. v[7]= 91. v[8]= 91. v[9]= 94.
• Relação dos códigos de controle (%): %d – será substituído por um valor decimal (valores inteiros) %i – será substituído por um valor inteiro %ld – será substituído por um valor inteiro longo (**long**) %f – será substituído por um valor real (%8.2f reservando 8 posições da tela das quais 2 serão usadas para as casas decimais) %c – será substituído por um caractere %s – será substituído por uma cadeia de caracteres	

Para justificar (ou alinhar) o resultado à direita no comando de saída:

%3d	%6.2f	%15s
1	1.00	Brasil
2	2.00	Pato Branco
...	...	Curitiba
9	9.00	UTFPR
10	10.00	Mater Dei
11	11.00	Programar é Bom
...	...	
99	99.00	
100	100.00	
101	101.00	
...	...	
999	999.99	

Formatando Números (1/2)

Outra alternativa para a formatação de números em Java é a classe "java.text.DecimalFormat".

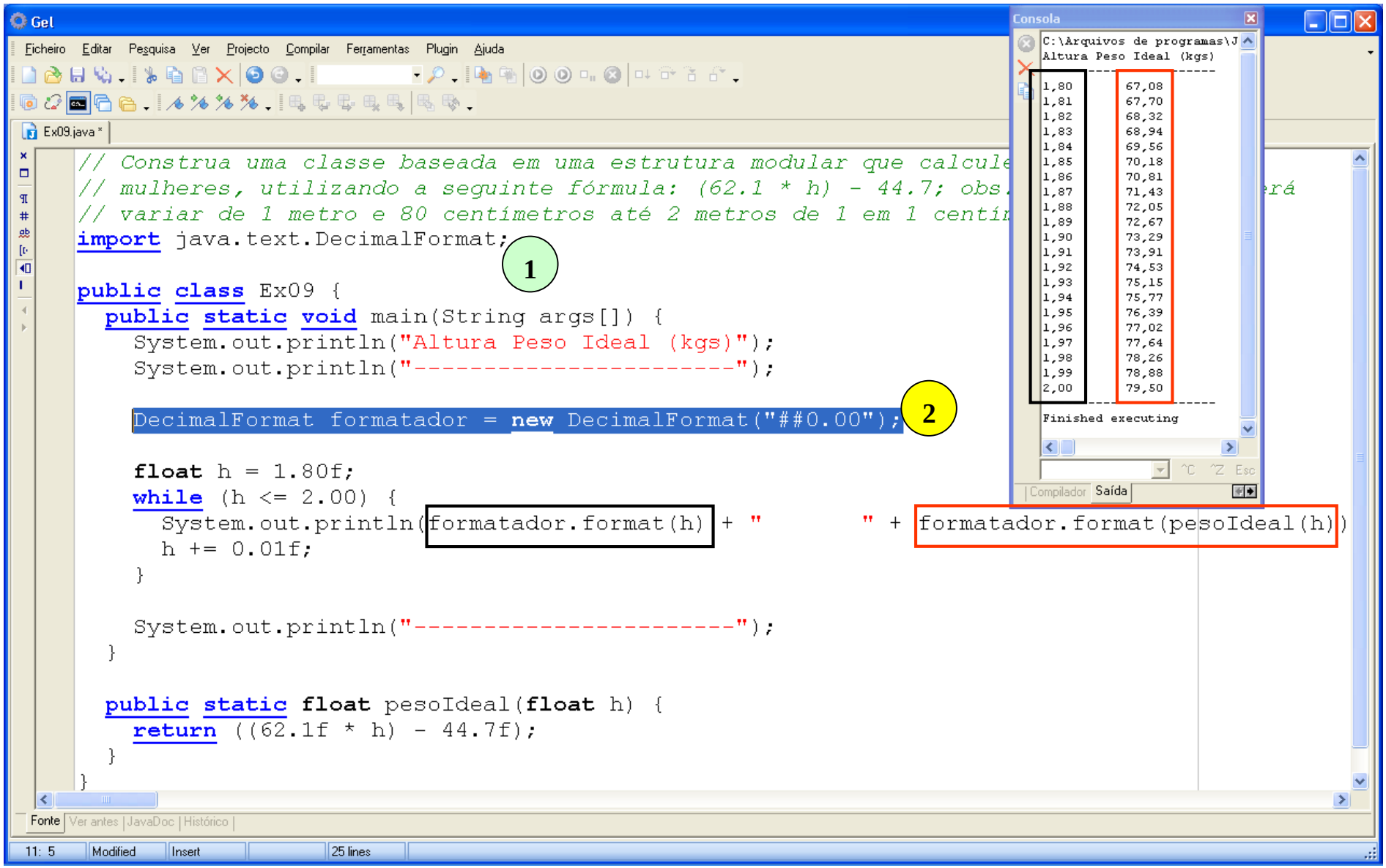
Tipos de formatação:

0	Imprime dígitos ou zero se vazio
#	Imprime dígitos ou espaço em braco
.	Separador Decimal
-	Sinal para números negativos

Utilizando a classe DecimalFormat:

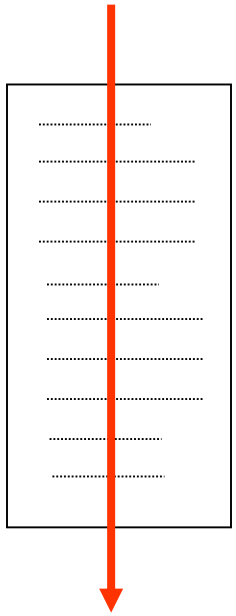
1. importar o pacote java.text:
   import java.text.**DecimalFormat**;
2. Instanciar e criar um objeto DecimalFormat:
   DecimalFormat formatador = **new** **DecimalFormat(**"##0.00");

Formatando Números (2/2)



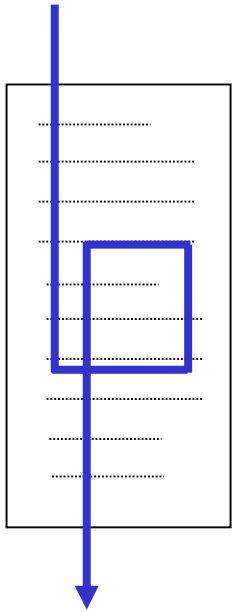
Tipos de Estruturas de Controle

fluxo sequencial de execução



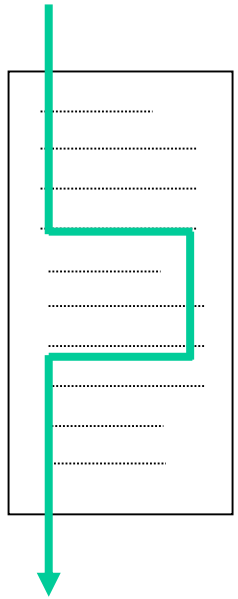
as diretivas são executadas uma após a outra sequencialmente do início ({) até o final (})

fluxo repetitivo de execução



as diretivas são executadas, de forma repetida, um determinado número de vezes:
for, while e do while.

fluxo de execução com desvio



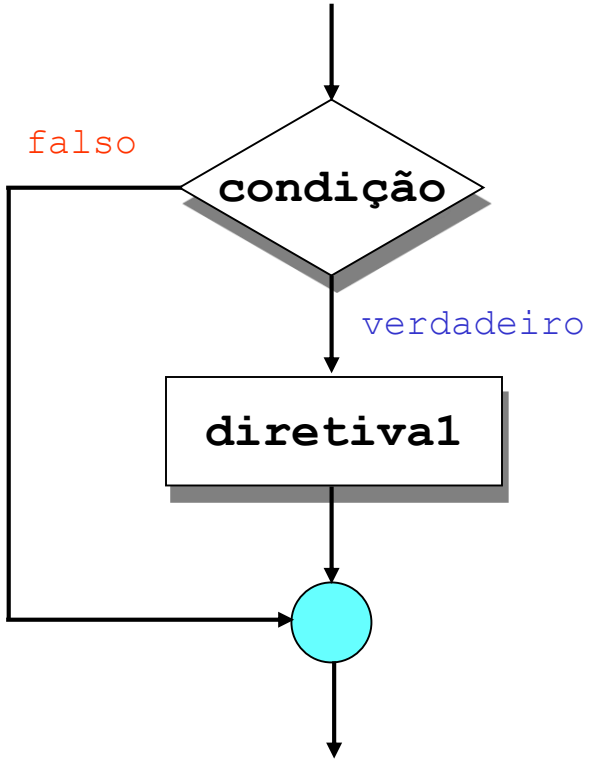
as diretivas são executadas dependendo do valor de uma condição, ou expressão lógica:
if e switch.

Estruturas de Desvio de Fluxo (1/3)

O **if** é uma diretiva de desvio "condicional" simples do fluxo de execução, isto é, capaz de selecionar um entre dois caminhos distintos para execução, dependendo do resultado, falso ou verdadeiro, resultante da expressão lógica associada.

```
if (expressão lógica, ou codição)
    diretiva1;
```

Esta primeira forma (simples) de desvio de fluxo permite executar ou não uma certa diretiva, conforme o resultado de uma condição.

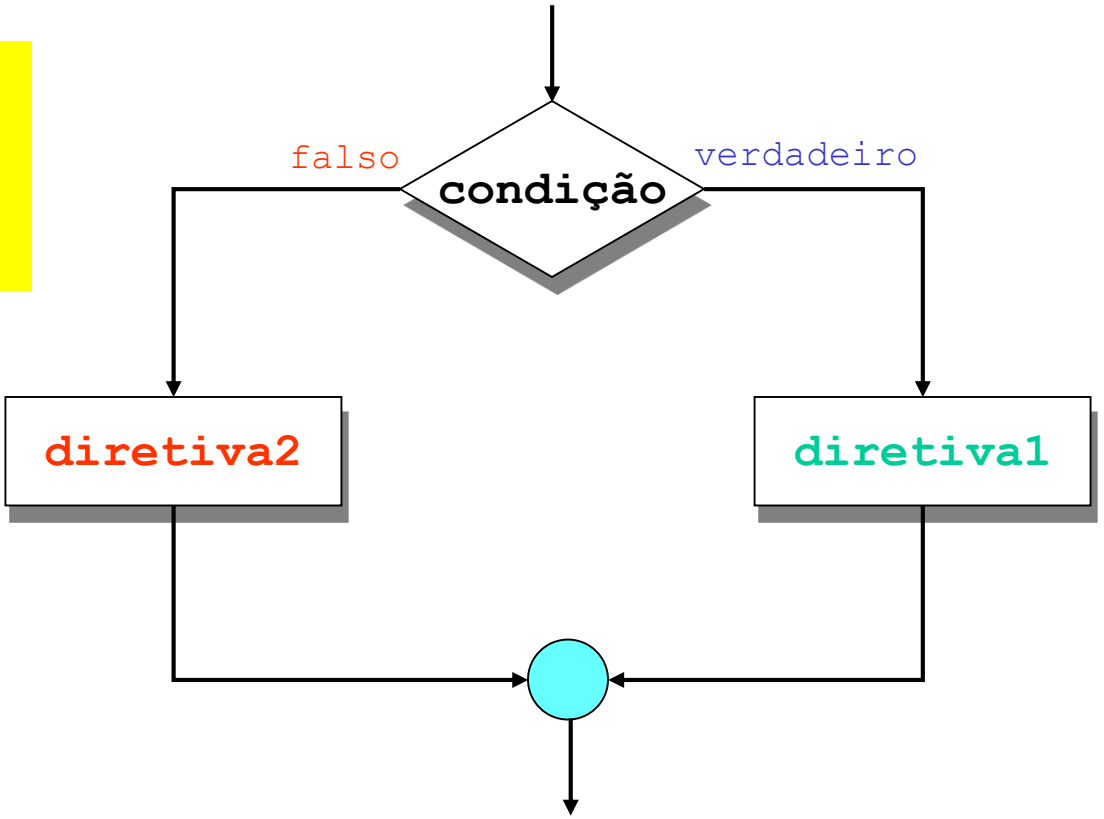


```java
// Efetuar a leitura de um valor inteiro positivo ou negativo e
// apresentar o número lido como sendo um valor positivo, ou seja, o
// programa deverá apresentar o módulo de um número fornecido.
// Lembre-se de verificar se o número fornecido é menor que zero,
// sendo multiplique-o por -1.

import java.util.Scanner;

public class _01 {

    public static void main(String[] args) {
      Scanner ler = new Scanner(System.in);

      int numero;

      System.out.printf("Informe um nro inteiro: ");
      numero = ler.nextInt();

      if (numero < 0)
         numero = numero * (-1);

      System.out.printf("\nO módulo do nro informado eh = %d\n",
        numero);
    }

}
```

Estruturas de Desvio de Fluxo (2/3)

Esta segunda forma (composta) de desvio de fluxo permite que seja executada a diretiva1, caso o resultado da expressão lógica for verdadeiro e no caso do resultado da expressão lógica for "falso" a "diretiva2" e que será executada.

```
if (expressão lógica)
    diretiva1;
else
    diretiva2;
```



```java
// Calcula a média de um acadêmico e define sua situação:
// aprovado, reprovado ou em exame.
import java.util.Scanner;
public class SituacaoAcademico {

  public static void main(String args[]) {
    Scanner ler = new Scanner(System.in);
    System.out.println("Forneça a nota do 1o. Bimestre: ");
    double nota1 = ler.nextDouble();
    System.out.println("Forneça a nota do 2o. Bimestre: ");
    double nota2 = ler.nextDouble();
    double media = (nota1 + nota2) / 2;

    System.out.println();

    if (media >= 7.0)
       System.out.println(media + ", aprovado.");
    else
      if (media < 4.0)
         System.out.println(media + ", reprovado.");
      else
         System.out.println(media + ", em exame.");

  }

}
```

Uso "Aninhado" de if else

```
if (condição1)
   if (condição2)
       diretiva1;
   else
       diretiva2;
```

Neste caso a diretiva1 é executada quando a condição1 e a condição2 forem verdadeiras, e a "diretiva2" é executada quando a "condição1 for verdadeira **e** a condição2 for falsa".

Isso porque a regra básica é que uma cláusula else sempre pertence à diretiva if imediatamente anterior. O uso de chaves permite modificar esta regra.

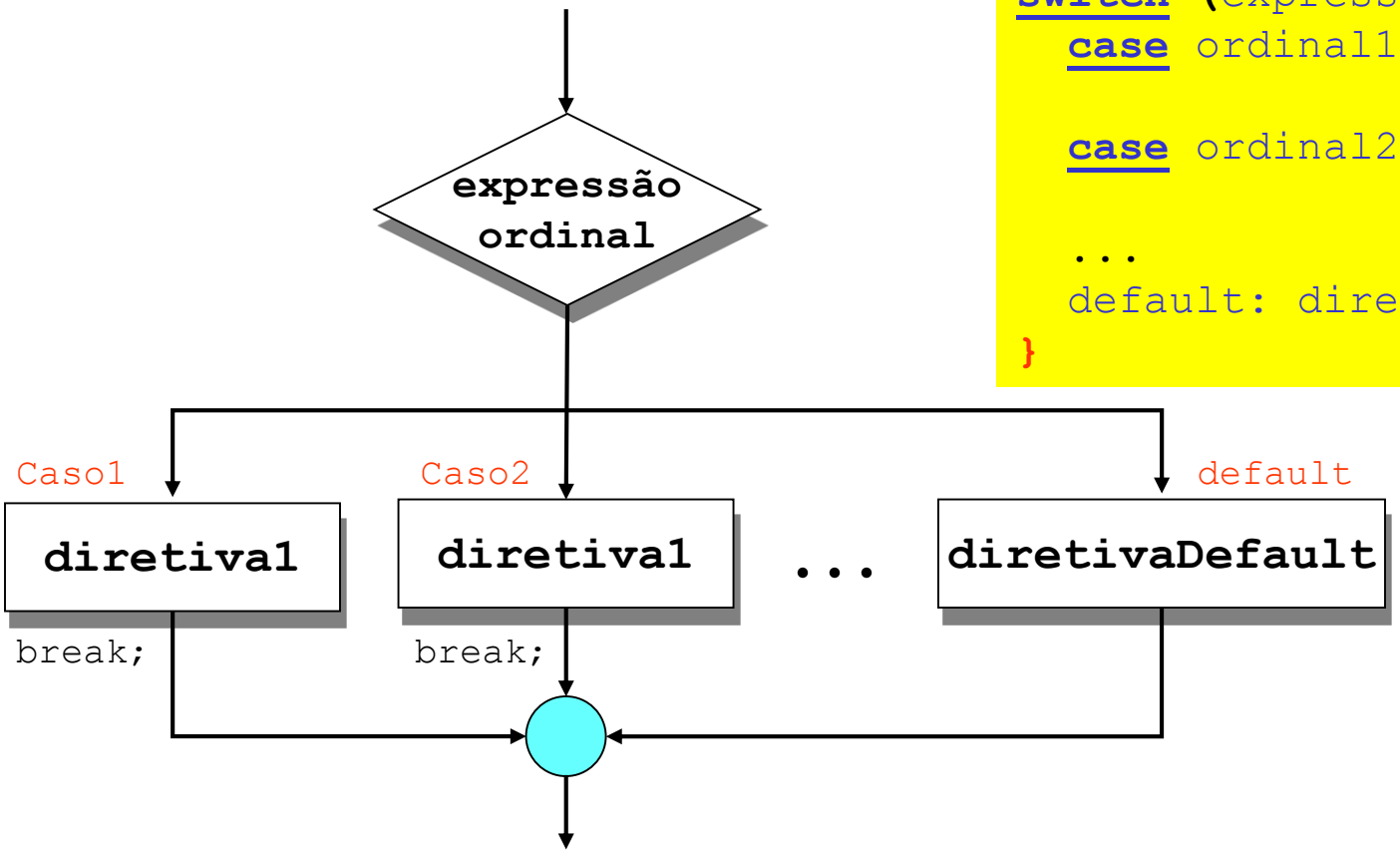
```
if (condição1) {
   if (condição2)
       diretiva1;
} else
    diretiva2;
```

Agora a diretiva1 continua sendo executada quando a condição1 e a condição2 forem verdadeiras, entretanto, a "diretiva2" é executada, simplesmentes, quando a "condição1 for falsa".

Estruturas de Desvio de Fluxo (3/3)

O **switch** é uma diretiva de desvio "múltiplo" de fluxo, isto é, com base na avaliação de uma expressão ordinal é escolhido um caminho de execução "dentre muitos".

```
switch (expressão ordinal) {
  case ordinal1: diretiva1;
                 break;
  case ordinal2: diretiva2;
                 break;
  ...
  default: diretivaDefault;
}
```



Semântica da Diretiva switch

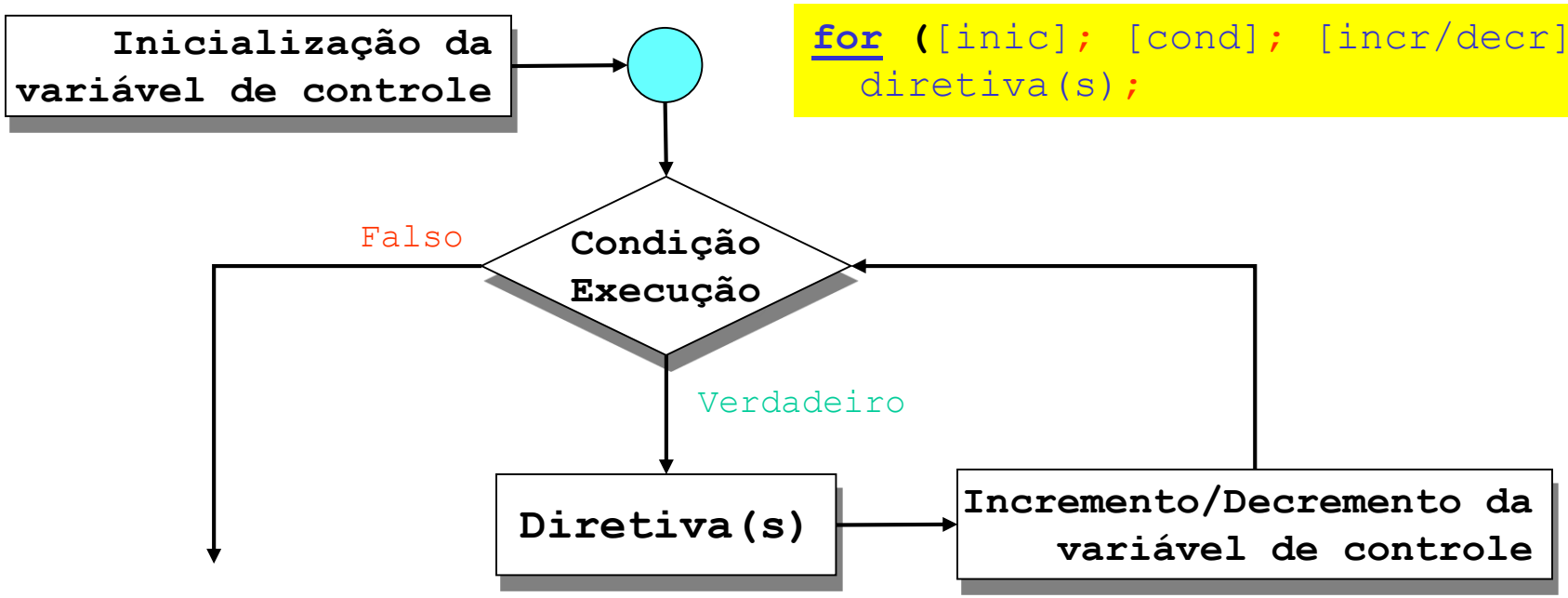
O corpo da diretiva switch tem um ou mais casos (cláusulas case) rotulados por valores ordinais. Conforme o resultado da avaliação da expressão ordinal associada ao switch, é selecionado o caso correspondente, se existir. Se um caso for selecionado, serão executadas as diretivas encontradas a partir deste até o final do corpo do switch (mesmo que isso signifique "invadir" outros casos) ou até a primeira diretiva break encontrada, que encerra o switch. Pode ser adicionado um caso especial, identificado pela cláusula default, cujas diretivas são executadas se nenhum caso for selecionado por meio do resultado da avaliação da expressão ordinal. A cláusula default e suas diretivas sempre são posicionadas no final do corpo da diretivas switch.

```
char opcao; // obter um valor para a variável "opcao".
switch (opcao) {
  case 'a':
  case 'A': System.out.println("Alteração");
            break;
  case 'i':
  case 'I': System.out.println("Inclusão");
            break;
  case 'e':
  case 'E': System.out.println("Exclusão");
            break;
  default: System.out.println("Opção inválida !");
}
```

```java
// Conta e exibe a quantidade de vogais existentes em uma cadeia de caracteres (string).
import java.util.Scanner;

public class ContaVogais {

  public static void main(String args[]) {
    Scanner ler = new Scanner(System.in);
    System.out.println("Informe uma cadeia de caracteres (string):");
    String msg = ler.nextLine();
    int ctA = 0, ctE = 0, ctI = 0, ctO = 0, ctU = 0;
    for (int i=0; i<msg.length(); i++) {
      switch (msg.charAt(i)) {
        case 'a':
        case 'A': ctA += 1;
                  break;
        case 'e':
        case 'E': ctE += 1;
                  break;
        case 'i':
        case 'I': ctI += 1;
                  break;
        case 'o':
        case 'O': ctO += 1;
                  break;
        case 'u':
        case 'U': ctU += 1;
      }
    }
    System.out.println("Na cadeia de caracteres: \"" + msg + "\".");
    System.out.println("Existem " + ctA + " vogais A.");
    System.out.println("Existem " + ctE + " vogais E.");
    System.out.println("Existem " + ctI + " vogais I.");
    System.out.println("Existem " + ctO + " vogais O.");
    System.out.println("Existem " + ctU + " vogais U.");
  }
}
```

Estruturas de Repetição Simples

A repetição é uma das tarefas mais comuns da programação, sendo utilizada para realizar contagens, totalizações, a obtenção de múltiplos dados, impressão etc. A repetição simples é a execução consecutiva de uma diretiva ou bloco de um número conhecido e fixo de vezes, o que muitas vezes se denomina laço automático. Cada repetição efetuada é o que se chama "iteração".




```
for ([inic]; [cond]; [incr/decr])
  diretiva(s);
```

Sintaxe e Semântica da Diretiva for

O for tem três seções distintas de controle, todas opcionais, delimitadas por um par de parênteses, nos quais cada seção é separada de outra por um ponto-e-vírgula. Após as seções de controle, é declarada a diretiva ou o bloco de diretivas que será repetido.

A primeira seção é utilizada para declarar e inicializar uma "variável de controle" (geralmente, um contador de repetições). A segunda seção determina a condição de execução da diretiva associada ao for e, geralmente, é uma expressão lógica que utiliza a variável de controle e outros valores. A última seção determina como a variável de controle será modificada a cada iteração.

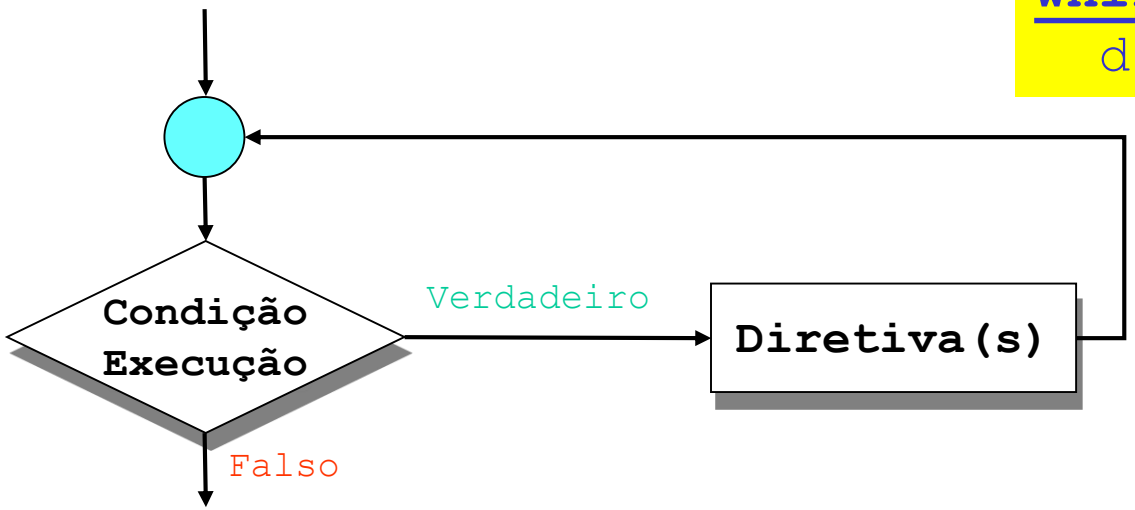
O for funciona como ilustrado no slide anterior, ou seja, após a execução da seção de inicialização, ocorre a avaliação da expressão lógica. Se a mesma for avaliada como verdadeira, a diretiva associada será executada uma vez (uma iteração); caso contrário, o comando for é encerrado e a execução do programa prossegue com a próxima diretiva após o for. A cada iteração, executa-se uma vez a seção de incremento ou decremento, que usualmente modifica o valor da variável de controle, e novamente é feita a verificação da condição para determinar se uma nova iteração, ou passo, deverá ser realizada ou não.

```java
// Apresenta a tabuada de um número inteiro qualquer.
import java.util.Scanner;

public class Tabuada {

  public static void main(String args[]) {
    Scanner = ler new Scanner(System.in);

    System.out.println("Informe um numero inteiro:");
    int n = ler.nextInt();

    for (int i=1; i<=10; i++) {
      System.out.println(i + " * " + n + " = " + (i*n));
    }
  }
}

// 1ª seção = a variável de controle "i" é inicializada com 1
// (int i=1; i<=10; i++)
// 2ª seção = determina o valor final para a variável de controle "i"
// (int i=1; i<=10; i++)
// 3ª seção = determina que o incremento na variável de controle "i"
// será realizado de um em um a cada iteração
// (int i=1; i<=10; i++)
// Em síntese: a variável de controle inicia com o valor 1 vai até 10
// (inclusive) de um em um, ou seja, executando 10 vezes a diretiva:
// System.out.println(i + " * " + n + " = " + (i*n));
```

Estruturas de Repetição Condicionais (1/2)

As estruturas de repetição condicionais são as diretivas cujo controle da repetição de execução é feito por meio da avaliação de uma expressão lógica associada à diretiva, sendo então chamados de laços condicionais. Essas estruturas são adequadas para permitir a execução repetida de uma ou mais diretivas por um número indeterminado de vezes, isto é, um número que não é conhecido durante a programação, mas que se torna conhecido durante a execução do programa, tal como um valor fornecido pelo usuário, obtido de um arquivo, ou ainda de cálculos realizados com outros dados. No Java, tais estruturas são o while e do while.

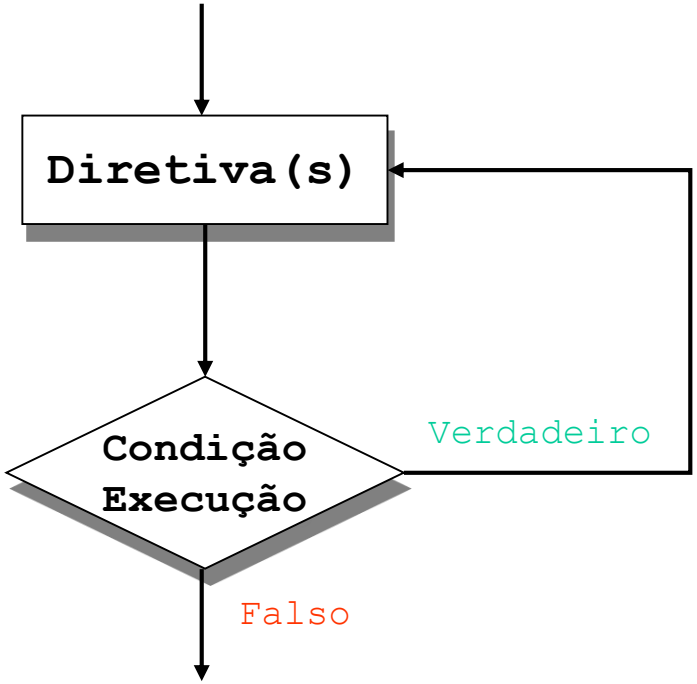
```
while (condição)
  diretiva(s);
```



```java
// Verifica se um número inteiro é primo.
import java.util.Scanner;
public class NumeroPrimo {
  public static void main(String args[]) {
    Scanner = ler new Scanner(System.in);
    System.out.println("Informe um numero inteiro:");
    int n = ler.nextInt();
    boolean ehPrimo = true;
    int i = 2;
    while ((ehPrimo == true) && (i <= (n / 2))) {
      if ((n % i) == 0)
         ehPrimo = false;   // encontrou um divisor (não eh primo)
      else i++;             // próximo divisor usando autoincremento
    }
    if (ehPrimo == true)
       System.out.println(n + " \"eh\" um numero primo.");
    else
       System.out.println(n + " \"não eh\" um numero primo.");
  }
}
// na diretiva while, um conjunto de instruções é repetido, enquanto
// o resultado de uma expressão lógica (a condição) é avaliado como
// verdadeiro.
// É importante observar que o while avalia o resultado da expressão
// lógica antes (a priori) de executar a diretiva associada; assim, é
// possível que esta diretiva nunca seja executada, caso a condição
// seja inicialmente falsa.
```

Estruturas de Repetição Condicionais (2/2)

A diretiva do while é outro laço condicional, no qual um conjunto de instruções também é repetido, enquanto o resultado da condição é avaliado como verdadeiro, mas, diferentemente do while, a diretiva associada é executada "uma primeira vez antes" da avaliação da expressão lógica e da eventual continuação da repetição.




```
do {
   diretiva(s);
} while (condição);
```

Controles de Estruturas de Repetição

As diretivas a seguir podem ser utilizadas para controlar melhor as estruturas de repetição:

label: diretiva; // onde diretiva deve ser um for, while ou do while

break [label]; // quebra, ou interrompe, um bloco repetitivo

continue [label]; // continua na próxima iteração

```
// "loop" e "test" são declarações de labels, ou rótulos
1.  loop: while (...) {
2.    for (...){
3.     // pula para fora da diretiva while, para a linha 7
4.     break loop;
5.    } // finaliza o for linha 2
6.  } // finaliza o while linha 1

7.  test: for (...) {
8.    while (...){
9.      if (...)                  // pula para a próxima iteração da
10.        continue test;        // diretiva for (linha 7)
11.   } // finaliza o while linha 8
12. } // finaliza o for linha 7
```

Referência

- Introdução ao Java.
  - Peter Jandl Junior.
  - São Paulo: Berkeley - 2002.
  - Capítulo 2: Java e sua Sintaxe, pág. 36..51.