Revisão para 1ª Miniprova (2009.2)

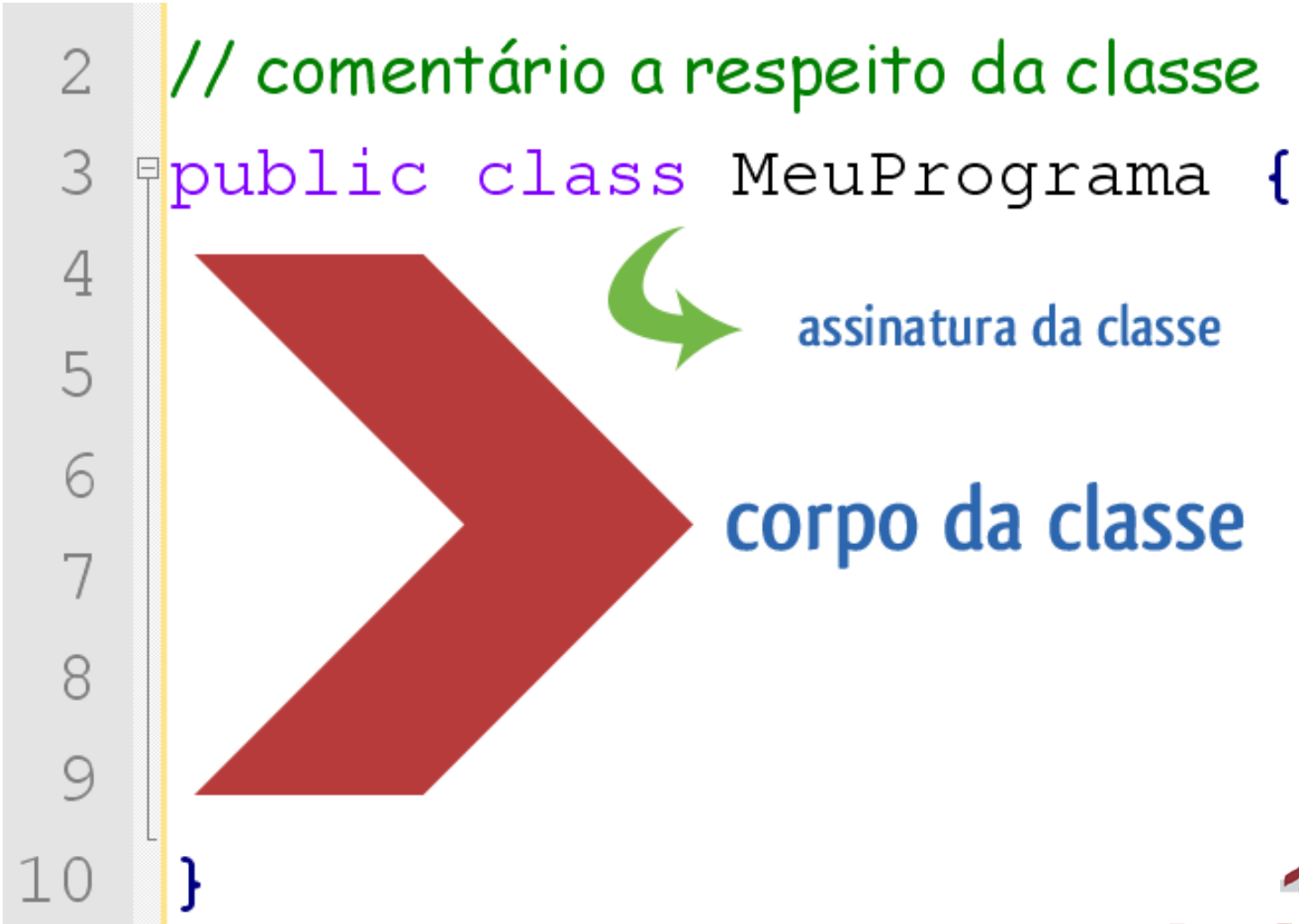
Introdução à Programação - IF669

Luís Gabriel Lima (lgnfl)

Centro de Informática
Universidade Federal de Pernambuco

4 de setembro de 2009

Definindo uma classe



... agora com o método main

```
2  // comentário a respeito da classe
3  public class MeuPrograma {
4
5    // comentários a respeito do método
6    public static void main(String[] args) {
7
8
9
10   }
11
12 }
```

assinatura do método

corpo do método

Inteiros

Existem 4 tipos primitivos para se representar números inteiros em Java:

- **byte** (8 bits);
- **short** (16 bits);
- **int** (32 bits);
- **long** (64 bits).

Pontos flutuantes

Existem 2 tipos primitivos para se representar números reais em Java:

- **float**;
- **double**.

Pontos flutuantes

Existem 2 tipos primitivos para se representar números reais em Java:

- **float**;
- **double**.

Atenção!

No código, para atribuir um valor para uma variável de ponto flutuante se a variável for do tipo **float** deve-se usar um **f** após o número. Exemplo: **float** *variavel* = `2.0f`;

Caracters e valores booleanos

- Para representar caracteres utiliza-se o tipo **char**. Os caracteres em Java utilizam a codificação **Unicode** (16 bits);
- Para representar valores booleanos utiliza-se o tipo **boolean**. Esse tipo pode assumir os valores **true** ou **false**.

Operadores matemáticos

- Adição (+);
- Subtração (-);
- Multiplicação (*);
- Divisão (/);
- Resto da divisão inteira (%).

Precedência

divisão, multiplicação e resto > adição e subtração

Exemplos de precedência

- a + b + c + d + e

Exemplos de precedência

- a + b + c + d + e
- a + b * c - d / e

Exemplos de precedência

- a + b + c + d + e
- a + b * c - d / e
- a / (b + c) - d % e

Exemplos de precedência

- a + b + c + d + e
- a + b * c - d / e
- a / (b + c) - d % e
- a / (b * (c + (d - e)))

Operadores lógicos

- Negação (NÃO lógico) (`!`);
- E lógico (`&&`);
- OU lógico (`||`).

Operadores lógicos

- Negação (NÃO lógico) (`!`);
- E lógico (`&&`);
- OU lógico (`||`).

Atenção!

- Todos esses operadores usam operandos booleanos e produzem resultados booleanos;
- O NÃO lógico é um operado unário;
- O E lógico e o OU lógico são operadores binários.

Expressões booleanas

Expressões booleanas geralmente usam os operadores de igualdade ou os operadores relacionais de Java, que retornam resultados booleanos:

- == igual a
- != diferente de
- < menor que
- > maior que
- <= menor ou igual que
- >= maior ou igual que

Expressões booleanas

Expressões booleanas geralmente usam os operadores de igualdade ou os operadores relacionais de Java, que retornam resultados booleanos:

- == igual a
- != diferente de
- < menor que
- > maior que
- <= menor ou igual que
- >= maior ou igual que

Esse tipo de expressão é muito utilizada para representar condições.

Expressões booleanas

Expressões booleanas geralmente usam os operadores de igualdade ou os operadores relacionais de Java, que retornam resultados booleanos:

- == igual a
- != diferente de
- < menor que
- > maior que
- <= menor ou igual que
- >= maior ou igual que

Esse tipo de expressão é muito utilizada para representar condições.

Atenção!

Note a diferença entre o operador de igualdade (==) e o operador de atribuição (=).

Precedência

Operadores aritméticos têm maior precedência do que operadores de igualdade e relacionais.

Precedência

Operadores aritméticos têm maior precedência do que operadores de igualdade e relacionais.

Exemplo:

```
total != temp + 132
```

Precedência

Operadores aritméticos têm maior precedência do que operadores de igualdade e relacionais.

Exemplo:

```
total != temp + 132
```

Nesse caso primeiro é calculado `temp + 132`, depois esse valor será comparado com `total`.

Algumas palavras...

- A menos que o programador especifique o contrário, um método é executado forma linear, uma instrução após a outra;

Algumas palavras...

- A menos que o programador especifique o contrário, um método é executado forma linear, uma instrução após a outra;
- Algumas estruturas de programação nos permitem decidir se uma instrução vai ou não ser executada;

Algumas palavras...

- A menos que o programador especifique o contrário, um método é executado forma linear, uma instrução após a outra;
- Algumas estruturas de programação nos permitem decidir se uma instrução vai ou não ser executada;
- Também existem estruturas de programação nos permitem executar várias vezes uma mesma instrução;

Algumas palavras...

- A menos que o programador especifique o contrário, um método é executado forma linear, uma instrução após a outra;
- Algumas estruturas de programação nos permitem decidir se uma instrução vai ou não ser executada;
- Também existem estruturas de programação nos permitem executar várias vezes uma mesma instrução;
- Essas decisões são tomadas de acordo com o resultado (**true** ou **false**) de expressões booleanas.

Estruturas condicionais

Estruturas condicionais nos permitem escolher qual instrução (ou conjunto de instruções) vai ser executado. As estruturas condicionais de Java são:

- `if`
- `if-else`
- `switch`

if

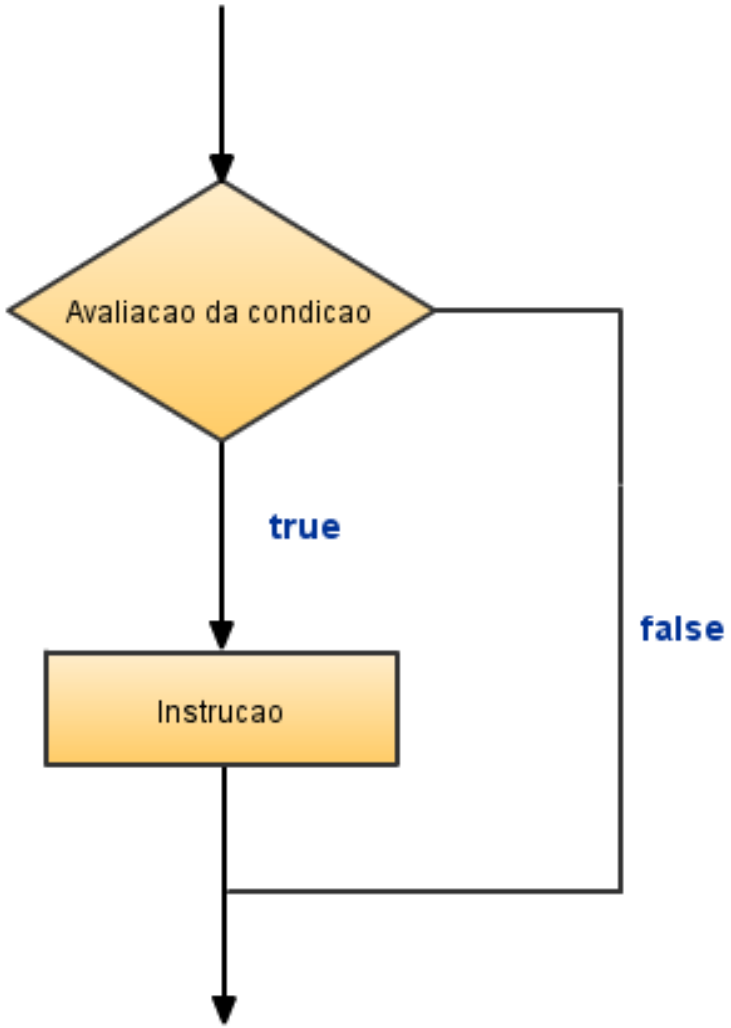
`if` é palavra reservada de Java

A *condicao* deve ser uma expressão booleana.

```
if ( condicao ) {
    instrucao;
}
```

Se a *condicao* for true, a *instrucao* é executada;
Se ela for false, a *instrucao* é pulada.

Lógica de um if



if-else

Adicionamos um **else** a estrutura **if** para fazer o if-else:

Sintaxe

if (`condicao`) { instrucao1; }
else { instrucao2; }

if-else

Adicionamos um **else** a estrutura **if** para fazer o if-else:

Sintaxe

if (`condicao`) { instrucao1; }
else { instrucao2; }

- Se *condicao* for **true**, *instrucao1* é executada, caso contrário (for **false**) a *instrucao2* é executada.

if-else

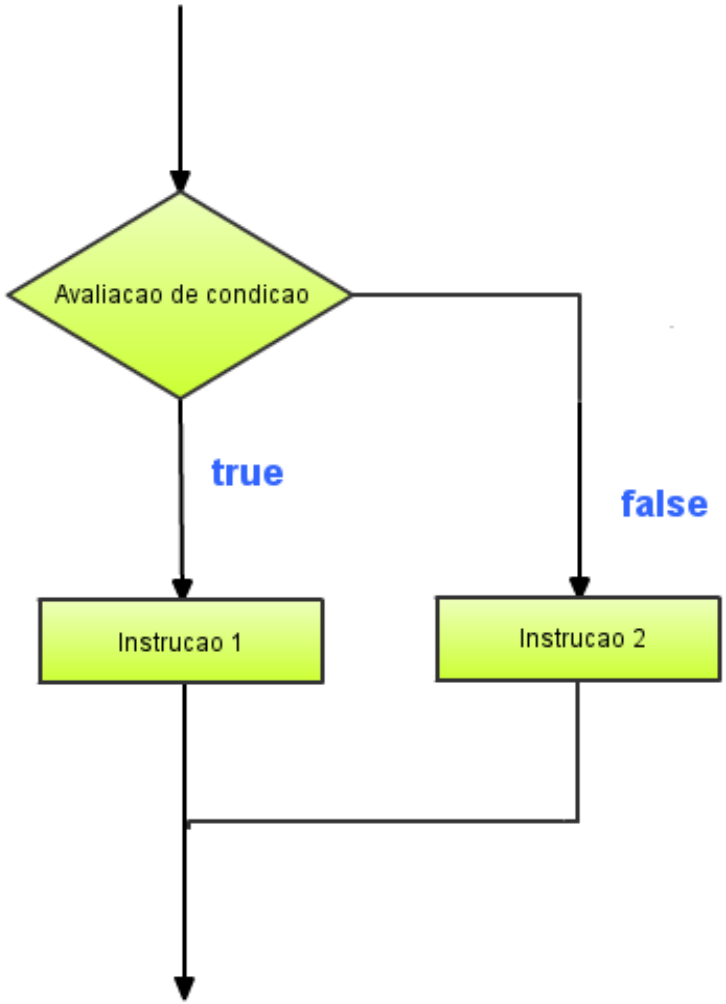
Adicionamos um **else** a estrutura **if** para fazer o if-else:

Sintaxe

if (`condicao`) { instrucao1; }
else { instrucao2; }

- Se *condicao* for **true**, *instrucao1* é executada, caso contrário (for **false**) a *instrucao2* é executada.
- Uma ou outra é executada, mas não as duas.

Lógica de um if-else



Indentação

- O uso de uma indentação consistente faz um programa ser mais fácil de ser lido e entendido;

Indentação

- O uso de uma indentação consistente faz um programa ser mais fácil de ser lido e entendido;
- Mesmo não fazendo diferença para o compilador, uma indentação apropriada é muito importante.

Comentários

- São incluídos no código para explicar os propósitos do programa e para descrever o seu processamento passo-a-passo;

Comentários

- São incluídos no código para explicar os propósitos do programa e para descrever o seu processamento passo-a-passo;
- Eles não afetam o modo como o programa é executado;

Comentários

- São incluídos no código para explicar os propósitos do programa e para descrever o seu processamento passo-a-passo;
- Eles não afetam o modo como o programa é executado;

Java possui três tipos de comentários:

Comentários

- São incluídos no código para explicar os propósitos do programa e para descrever o seu processamento passo-a-passo;
- Eles não afetam o modo como o programa é executado;

Java possui três tipos de comentários:

```
// comentário de uma única linha
```

Comentários

- São incluídos no código para explicar os propósitos do programa e para descrever o seu processamento passo-a-passo;
- Eles não afetam o modo como o programa é executado;

Java possui três tipos de comentários:

```
// comentário de uma única linha
/* comentário de
várias linhas */
```

Comentários

- São incluídos no código para explicar os propósitos do programa e para descrever o seu processamento passo-a-passo;
- Eles não afetam o modo como o programa é executado;

Java possui três tipos de comentários:

```
// comentário de uma única linha
/* comentário de
várias linhas */
/** comentário javadoc
(pode ter múltiplas linhas) */
```

Game Over! ;]