Aula Prática

Introdução à Programação - IF669

Monitoria IP/CC 2010.1

Centro de Informática
Universidade Federal de Pernambuco

31 de março de 2010

Definindo uma classe

```
public class PrimeiraClasse {

    // corpo da classe

    public static void main(String[] args) {

        // corpo do método (main)

    }

}
```

Tipos Primitivos

- Inteiros: **byte** (8 bits), **short** (16), **int** (32) e **long** (64)
- Números reais: **float** e **double**
- Caracteres: **char**
- Valores booleanos: **boolean**

Tipos Primitivos

- Inteiros: **byte** (8 bits), **short** (16), **int** (32) e **long** (64)
- Números reais: **float** e **double**
- Caracteres: **char**
- Valores booleanos: **boolean**

```
int idade = 10;
float peso = 65.6f;
double quantia = 11.4;
```

```
char letra = 'a';
boolean flag = false;
```

Classe String

- É utilizada em Java para representar uma cadeia de caracteres;
- Apesar de não ser um tipo primitivo, pode ser inicializado de maneira semelhante;
- Tem alguns métodos muito úteis. (ver na documentação)

Classe String

- É utilizada em Java para representar uma cadeia de caracteres;
- Apesar de não ser um tipo primitivo, pode ser inicializado de maneira semelhante;
- Tem alguns métodos muito úteis. (ver na documentação)

```
String titulo = "Aula Prática - IP/CC";
String titulo2 = new String("Aula Prática - IP/CC");
```

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 operador unário.

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 igualidade (==) 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`.

Saída de dados

- É a maneira que temos de mostrar dados ao usuário durante execução do programa;
- Utilizaremos inicialmente o console para nos comunicar com o usuário;
- Para imprimir mensagens no console utilizaremos o objeto `System.out`.

Saída de dados

- É a maneira que temos de mostrar dados ao usuário durante execução do programa;
- Utilizaremos inicialmente o console para nos comunicar com o usuário;
- Para imprimir mensagens no console utilizaremos o objeto `System.out`.

```
String titulo = "Aula Prática - IP/CC";
float pi = 3.1415f;
System.out.println(titulo);
System.out.println("Alguma coisa.");
System.out.printf("Valor de PI: %f", pi);
```

Entrada de dados

- É a maneira que temos de receber dados do usuário durante execução do programa;
- Para receber dados através do console utilizaremos o objeto `Scanner`. (ver documentação)

Entrada de dados

- É a maneira que temos de receber dados do usuário durante execução do programa;
- Para receber dados através do console utilizaremos o objeto `Scanner`. (ver documentação)

```
Scanner input = new Scanner(System.in);
System.out.println("Digite seu nome:");
String nome = input.next();
System.out.println("Digite sua idade:");
int idade = input.nextInt();
System.out.println("Digite seu peso:");
double peso = input.nextDouble();
```

Algumas palavras...

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

Algumas palavras...

- A menos que o programador especifique o contrário, um método é executado de 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 de 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 de 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(condicao){
    instrucao1;
    instrucao2;
    // ...
}
```

- `condicao` deve ser uma expressão booleana;
- Se `condicao` for **true**, `instrucao1` e o que estiver depois dela é executado;
- Se `condicao` for **false**, o que estiver dentro do bloco do `if` é pulado.

if-else

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

- Se `condicao` for **true**, `instrucao1` será executada;
- Se `condicao` for **false**, `instrucao2` será executada;
- Uma ou outra é executada, mas não as duas.

Indentação

- Consiste em deixar o código no interior de um bloco com mais espaçamento do que o código que está fora desse bloco;
- 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;
- Grande parte dos editores atuais ajudam a manter o código identado.

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;

```
// comentário de uma única linha

/*  comentário de
            várias linhas   */

/** comentário javadoc
    (pode ter múltiplas linhas)

    @author Luís Gabriel (lgnfl)
*/
```

Dúvidas?

Go go go!

- http://cin.ufpe.br/~if669/
- Atividades
- Aulas práticas
- Aula 1