

Paradigmas



Paradigmas



Tradicional Fortran, Algol, Algol-68, Pascal, C, Cobol, PL/I

Paradigmas



Tradicional OO Fortran, Algol, Algol-68, Pascal, C, Cobol, PL/I
Simula-67, Smalltalk, C++, Eiffel, Object Pascal, Java, C#

Paradigmas





Tradicional	Fortran, Algol, Algol-68, Pascal, C, Cobol, PL/I
OO	Simula-67, Smalltalk, C++, Eiffel, Object Pascal, Java, C#
Funcional	Lisp, ML, Scheme, Miranda, Haskell

Paradigmas



Tradicional	Fortran, Algol, Algol-68, Pascal, C, Cobol, PL/I
OO	Simula-67, Smalltalk, C++, Eiffel, Object Pascal, Java, C#
Funcional	Lisp, ML, Scheme, Miranda, Haskell
Lógico	Prolog, Mercury



Paradigma Imperativo

visão global / conceituação

- Variável e atribuição
- Comandos
 - ★ Composição seqüencial
 - ★ Seleção
 - ★ Repetição
- Funções e procedimentos

Modificação do valor de variáveis: base da programação imperativa

- **Variável:** *lugar* (posição na memória) que contém um certo *valor* (difere do usual em matemática!)
- Valor armazenado em uma variável pode ser modificado por meio de um **comando de atribuição**
- Execução baseada em **comandos**, que modificam / controlam a modificação de valores de variáveis



Declaração de variáveis

```
boolean x; int y = 10;
```


Declaração de variáveis

```
boolean x; int y = 10;
```

- Em Java (e LPs em geral), toda variável deve ser *declarada*.

Declaração de variáveis

```
boolean x; int y = 10;
```

- Em Java (e LPs em geral), toda variável deve ser *declarada*.
- Declaração especifica nome e *tipo*

Declaração de variáveis

```
boolean x; int y = 10;
```

- Em Java (e LPs em geral), toda variável deve ser *declarada*.
- Declaração especifica nome e *tipo*
- **Tipo determina conjunto de valores que podem ser armazenados na variável**

Declaração de variáveis

```
boolean x; int y = 10;
```

- Em Java (e LPs em geral), toda variável deve ser *declarada*.
- Declaração especifica nome e *tipo*
- **Tipo determina conjunto de valores que podem ser armazenados na variável**
- Declaração pode especificar valor inicial (valor armazenado no instante da criação)



Atribuição

$$v = e;$$

Atribuição

$$v = e;$$

- Execução: expressão e é avaliada e valor resultante atribuído à variável v

Atribuição

$$v = e;$$

- Execução: expressão e é avaliada e valor resultante atribuído à variável v
- Após atribuição, valor anterior de v “é perdido”
(não pode ser mais obtido usando v , a não ser que nova atribuição seja feita)



Atribuição

$$a = b = b + 1;$$

Atribuição

```
a = b = b + 1;
```

- Comando de atribuição é expressão em Java

Atribuição

```
a = b = b + 1;
```

- Comando de atribuição é expressão em Java
- Não confundir: $a = b$ com $a == b$

Comandos

Composição de comandos estabelece ordem de execução
(determina ordem de modificação do valor de variáveis)

- Composição seqüencial

$$c_1; c_2;$$

Execução de c_1 e, em seguida, c_2

```
 $a = 10; b = \text{true}; c = 2*a;$ 
```

Comandos

- Seleção (comando if)

<code>if (<i>b</i>) <i>c</i>₁; else <i>c</i>₂;</code>

Se a avaliação de *b* retornar `true`, *c*₁ é executado;
se `false`, *c*₂ é executado.

Cláusula `else` opcional: ausência \Rightarrow
nenhum comando é executado se avaliação de *b* retornar `false`.

```
if ( a > 10 ) { a = a + 10; b = b + 1; }  
    else { b = 0; if ( c > 1 ) a = a + 5; }
```

Comandos

- Repetição

```
while ( b ) c;
```

Expressão *b* é avaliada; se resultado for true, *c* é executado, e o processo se repete; se false, execução termina

```
soma = 0; i = 1;  
while ( i <= n )  
{ soma = soma + i; i = i + 1; }
```



Funções e Procedimentos

Mecanismos de **abstração**

- **Funções:** fornecem um resultado, de acordo com argumentos
Ex: + fornece resultado da adição, de dois argumentos
- **Procedimentos:** modificam valores de variáveis, de acordo com argumentos
- Em Java (e LOOs em geral), funções e procedimentos são casos especiais de **métodos**

 Paradigma Imperativo