Computação II – Orientação a Objetos

Fabio Mascarenhas - 2016.2

http://www.dcc.ufrj.br/~fabiom/java

Introdução

- Esse não é um curso de Java!
- O objetivo é aprender os conceitos da programação orientada a objetos, e sua aplicação em programas reais
- A linguagem Java é só um veículo para isso
- Cuidado: programar em Java não é necessariamente programar de maneira OO

Java não-OO

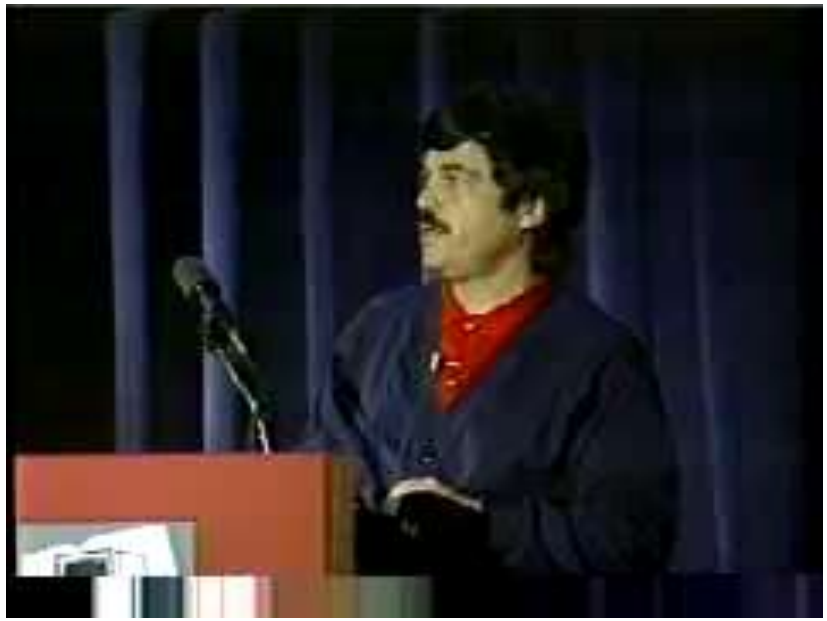
- Vamos nos concentrar nas partes OO de Java, mas ela também tem tudo que vocês já estão acostumados com C:
  - Variáveis locais e globais, funções, números, entrada e saída no console...
- Até a sintaxe é muito parecida
- Programas C simples podem ser reescritos em Java com pequenas alterações, mas **não são exemplos de programação OO**!

De C para “Java com sabor de C”

```c
#include <stdio.h>

void geraConjunto(double vetor[], int tamanho, double inicial) {
    for (int i = 0 ; i < tamanho; i++) {
        vetor[i] = inicial;
        inicial *= 2;
    }
}

int main(int argc, char *argv[])
{
    double vetor[5], num;
    int i;

    puts("Este programa gera um vetor de numeros inteiros.\n");
    puts("Entre com o numero inicial do conjunto. ");
    scanf("%lf", &num);

    /* Geracao do conjunto */
    geraConjunto(vetor, 5, num);

    /* Impressao do conjunto */
    for (i = 0; i < 5; i++)
        printf("Elemento %d = %lf\n", i, vetor[i]);

    return 0;
}
```

História da OO

- Sketchpad (1963)
  - Editor gráfico pioneiro, introduz o conceito de *objeto*
  - Desenhos e diagramas podiam ser agrupados e clonados: um desenho *mestre* podia ter várias *instâncias*, e mudanças no mestre eram refletidas nas instâncias

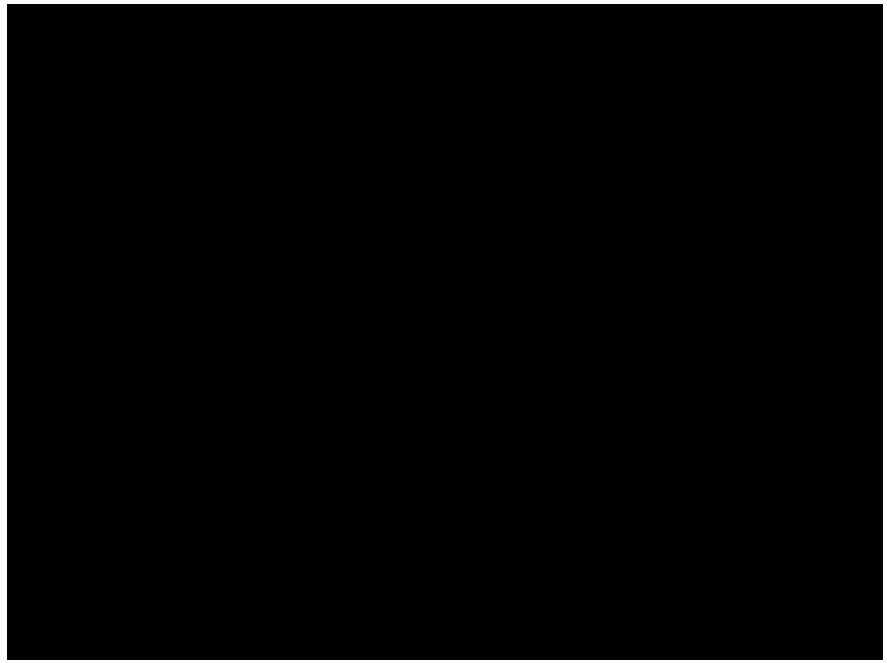


História da OO

- Simula (1962-68)
  - Família de linguagens de programação para simulação de eventos discretos
  - Simula 67 introduz os conceitos de *classes*, *métodos virtuais*, *herança* e *subtipagem*
  - O modelo orientado a objetos é bastante natural para simulações, onde pode-se modelar cada elemento da simulação como um objeto responsável pelo seu próprio comportamento
  - Jogos também são simulações!

História da OO

- Smalltalk (1972-80)
  - Primeira linguagem OO “pura”
  - Qualquer coisa em Smalltalk é um objeto: uma *instância* de uma *classe* que responde a *mensagens* (*métodos virtuais*)
  - Inclusive coisas como as próprias classes, o contexto onde ficam armazenadas as variáveis, as partes do ambiente de programação...



História da OO

- Java (1995-?)
  - Criada originalmente para ser embutida em receptores de TV a cabo
  - Depois embutida em páginas web, como *applets*, mas ganhou maior uso como linguagem usada em sistemas web no lado do servidor, e em algumas aplicações desktop
  - Sintaxe inspirada em C, mas o modelo de execução está mais próximo de Simula e Smalltalk

Dados primitivos

- Objetos concretos são compostos de *estado* e *comportamento*
- O estado pode ser composto por outros objetos, que por sua vez têm seu próprio estado, composto por outros objetos, que têm seu próprio estado, que...
- Alguma hora batemos nos *átomos* da linguagem, os *tipos primitivos*
- Manipulamos os tipos primitivos usando os operadores pré-definidos de Java; eles não possuem métodos

Inteiros

- Sinalizados com o tipo `int`
  - Possuem sinal e 32 bits de precisão, então vão de ~-2.000.000.000 a ~2.000.000.000
  - Operações como em C: +, -, *, /, &, |
  - Exemplos: 2, 5, 42, 1000000, -1

“Reais”

- Sinalizados pelo tipo `double`
  - Ponto flutuante com 64 bits, o que dá 52 bits de precisão (a mantissa)
  - Também possui as mesmas operações que seus similares em C, e os mesmos problemas com erros e incapacidade de representar exatamente diversos números
  - Exemplos: 1.23, 2.56e7, 3e-2, 0.0

Booleanos

- Sinalizados pelo tipo `boolean`
  - Dois valores: verdadeiro (true) e falso (false)
  - Operações como os “booleanos” de C: && (e), || (ou), ! (negação)
  - Java também tem o operador ternário ?:

exp. boolean ? e.p.v : exp.f

Cadeias (strings)

- Sinalizados pelo tipo `String`
  - Não são exatamente tão primitivos como os anteriores, pois já são objetos que também possuem métodos
  - Mas têm uma operação pré-definida: concatenação com +
  - Concatenação cria uma nova string, não muda seus operandos
  - Mesmos códigos de escape que em C mas, ao contrário de C, strings não são vetores de caracteres, e não podemos acessar (ou modificar) seus caracteres com `[]`

Vetores

- Pode-se construir um vetor a partir de qualquer tipo (inclusive vetores de objetos complexos)
  - `int[]`, `double[]`, `int[][]`, `String[]`, ...
  - Um vetor é criado com `new`: `new int[5]`
  - Acessados com `[]` como em C
  - Não existe `free`: se um vetor não é mais alcançável ele alguma hora é liberado