

# Computação II – Orientação a Objetos

---

Fabio Mascarenhas - 2016.1

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

# 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

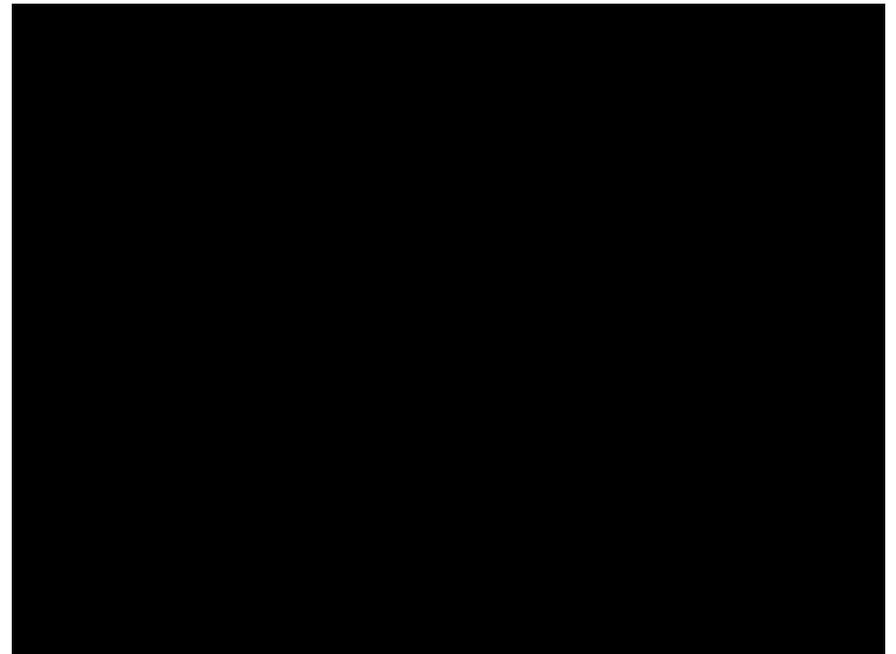
---

- 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  $\sim -2.000.000.000$  a  $\sim 2.000.000.000$
- Operações como em C: +, -, \*, /, &, | , <, ==, >=  
*aritméticas* , *relacionais*
- Exemplos: 2, 5, 42, 1000000, -1

# “Reais”

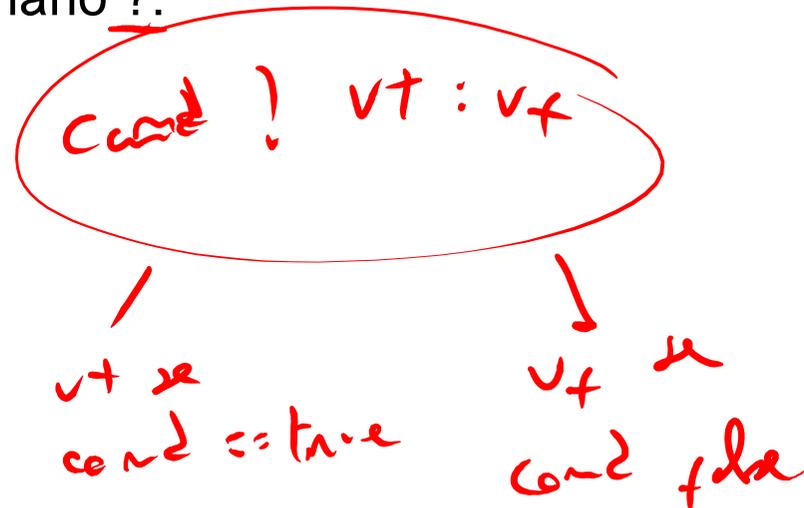
---

- 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:  $1.23$ ,  $2.56e7$ ,  $3e-2$ ,  $0.0$   
 $2.56 \times 10^7$

# 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 ?:



# Cadeias (strings)

---

- Sinalizados pelo tipo String *~ classe*
  - 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 [ ]

*int(10)[5] vs. int\*\**

# Vetores

- Pode-se construir um vetor a partir de qualquer tipo (inclusive vetores de objetos complexos)

*vetor de vetores de int*

- int[], double[], int[][3], String[], ...

- Um vetor é criado com new: new int[5]

~~*new int[10][5]*~~

- Acessados com [] como em C

*v[i] = x  
x = v[i]*

*new int(5)(5)  
vv[i][j]*

- Não existe free: se um vetor não é mais alcançável ele alguma hora é liberado

# Classes

---

- Uma das unidades básicas de um programa Java
- No nível mais simples é como uma *struct* de C, agrupando diversos valores em uma mesma entidade, e possui *campos*
- Campos de uma classe podem ter qualquer tipo, inclusive outras classes
- *Instâncias* de uma classe também são criadas com `new`

# Construtor

---

- Podemos inicializar os campos de uma nova instância de modo parecido do que fazemos em C, mas esse não é o estilo OO apropriado
- A inicialização dos campos de um objeto é tarefa do seu construtor
- A declaração de um construtor se parece com a declaração de uma função em C, mas um construtor sempre tem o mesmo nome da classe
- Podemos passar argumentos para a classe quando usamos new, e esses argumentos são os parâmetros do construtor
- Dentro do construtor, a variável especial this aponta para o objeto recém-criado que deve ser inicializado