# Orientação a Objetos



Daniel Wildt
http://danielwildt.blogspot.com

# Agenda



- Orientação a Objetos
- Classe x Objeto
- **Representação classe**
  - **Atributos / operações**
- **Construtores e Destrutores**
- **Liberando memória de objetos**
- **Abstração**
- **Encapsulação**

# Agenda



- **Relacionamentos**
  - **Herança**
  - **Associação**
  - **Agregação**
  - **Composição**
- **Polimorfismo**
- **Classes abstratas e métodos abstratos**
- **Visibilidade**
- Interfaces

# Orientação a Objetos

4

- Programação Estruturada
  - Uso do Goto / Estruturas para saltos dentro da programação.
  - Código monolítico
- Programação Modular
  - Divisão em módulos (procedures e funções).
  - Quebra do sistema em pedaços menores.

# Orientação a Objetos

5

- Programação Orientada a Objetos (POO)
  - Novos conceitos são inseridos na programação
    - Herança: conceitos de classe pai e classe filha
    - Abstração: abstrações do mundo real para mundo computacional
    - Encapsulação: proteger acesso a informações
    - Polimorfismo: facilidades para reuso de código, simplificações de programação.

# Orientação a Objetos

6

- Programação Orientada a Objetos (POO)
  - Histórico
    - Simula, início da aplicação de conceitos OO.
      - Voltada para processos de Simulação
      - Simula I (62): Linguagem orientada a processos
      - Simula 67: versão OO
      - Link de apoio: http://staff.um.edu.mt/jskl1/talk.html
      - Antecedeu o Smalltalk

# Orientação a Objetos

7

- Programação Orientada a Objetos (POO)
  - Histórico
    - Smalltalk
      - Linguagem OO desenvolvida na Xerox.
      - Programas são compilados para bytecodes e executados em uma máquina virtual.
      - Criação do conceito de Garbage Collector
      - No **Smalltalk-80** o código de bytecodes era compilado para a arquitetura nativa. Idéias que Java e .NET hoje implementam. Conceito de dynamic-translation.

# Orientação a Objetos

8

- Programação Orientada a Objetos (POO)
  - Histórico
    - Smalltalk
      - Tudo é um objeto (everything is a object)
      - Links interessantes:
        http://www.iam.unibe.ch/~ducasse/FreeBooks.html
        http://www.iam.unibe.ch/~ducasse/FreeBooks/SmalltalkHistoryHOPL.pdf
        http://www.whysmalltalk.com/
        http://encyclopedia.thefreedictionary.com/Smalltalk%20programming%20language

# Orientação a Objetos

9

- Programação Orientada a Objetos (POO)
  - Coesão e Acoplamento: conceitos importantes
    - Exemplo: Construir classe Calculadora com operação de somar, que deve retornar o resultado da soma

# Orientação a Objetos

10

- Programação Orientada a Objetos (POO)
  - Coesão
    - Código não coeso é: implementação do módulo Calculadora e da operação somar. Além de somar envia e-mail para o usuário com o resultado da operação e mostra mensagem na tela com o resultado.
    - Problema: manda email e mostra mensagem em tela, desnecessário para o requisitado

# Orientação a Objetos

11

- Programação Orientada a Objetos (POO)
  - Coesão
    - Perguntar sempre: Cada entidade e cada funcionalidade realizam o seu trabalho? Ou fazem mais do que devem?
    - Buscar: código mais coeso possível (mais simples possível)
    - Código sendo coeso, facilita a reutilização

# Orientação a Objetos

12

- Programação Orientada a Objetos (POO)
  - Acoplamento
    - Código acoplado é: programador implementa o módulo calculadora no mesmo arquivo da tela de entrada de dados
    - Problema: a calculadora pode ser reutilizada em outro projeto, mas acaba levando uma tela de entrada de dados desnecessária.
    - Buscar: código menos acoplado possível
    - Codigo menos acoplado facilita a reutilização

# Classe x Objeto



- Classe
  - Representação, abstração do mundo real. Exemplo: Carro, Computador, Caneta, Pessoa. É a representação de uma entidade do sistema real em um modelo computacional
- Objeto
  - Abstração que se torna realidade. Caneta -> BIC, Carro -> Fusca, Pessoa -> José Silva

# Representação de Classe

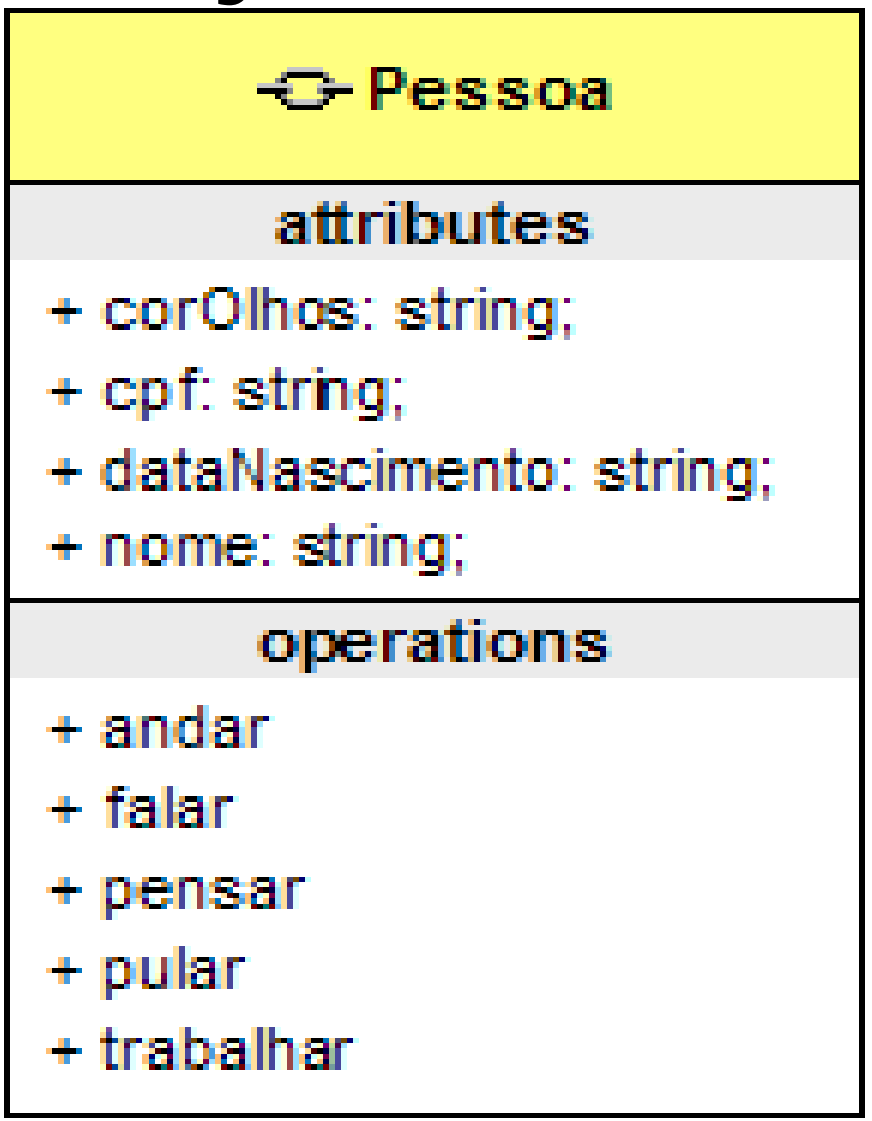


- Atributos e operações
  - Para cada representação que definimos, podemos informar atributos (variáveis) e operações (métodos ou ações) que esta representação possui e executa.
  - Exemplo (Pessoa)
    - Atributos: nome, cpf, dataNascimento, corOlhos, corCabelo. Representam o **estado** do objeto.
    - Operações: andar, falar, pular, trabalhar, pensar. Representam **comportamento**.

# Representação de Classe



- Atributos e operações



# Construtores e Destrutores



- Construtor: método de inicialização do objeto.
- Destrutor: método “chamado” no momento da liberação do objeto da memória.

# Liberando memória de Objetos



- Diferenciar processos de memória gerenciada e não gerenciada

# Abstração



- Trazer a representação de uma entidade do mundo real para o mundo computacional
- Ex: Pessoa
  - Nome
  - DataNascimento
  - Altura
  - Peso

# Encapsulação



- Exemplo: Implementação de uma calculadora
  - Pessoa1: Como funciona a função de somar?
  - Pessoa2: Você passa dois parâmetros e eu te retorno o resultado
  - Pessoa1: Mas como é implementado internamente?
  - Pessoa2: Você não precisa saber disto.

# Encapsulação



- Ocultar as especificades de funcionalidades
- Disponibilizar apenas as informações necessárias para o andamento do trabalho
- Menor complexidade de entendimento e uso.

# Relacionamentos



- Herança
  - Conceito: Classe A é base de B
  - Conceitos de generalização / especialização
  - Ver exemplos de herança disponíveis na linguagem!
  - Verificar como identificar se um objeto é de determinada classe ou se ele herda de determinada classe
  - "Compatibilidade" de uso de classes em declarações de objetos

# Relacionamentos



- Associação
  - Associação existente entre duas entidades
  - Cliente possui Pedidos. Um pedido é referente a um cliente.

# Relacionamentos



- Agregação
  - Considerar algo maior pensando na relação todo-parte.
  - Um Carro C possui um Motor M. Se o carro for removido do sistema, o motor que estava sendo desenvolvido pode ser reaproveitado em outro carro.

# Relacionamentos



- Composição
  - Também baseado na relação todo-parte.
  - Neste caso, se o objeto maior for removido, as suas partes filhas serão removidas também.
  - Imagine o caso de um Carro C que possui uma Placa P registrada. Se o carro deixa de existir, a placa não tem mais utilidade dentro do sistema.

# Relacionamentos



- Pensamentos em exemplos:
  - Herança: Pessoa física "é um tipo" de Pessoa.
  - Associação: Um pedido "usa um" cliente dentro da sua estrutura.
  - Agregação: Um carro "é composto" de um Motor, etc.

# Polimorfismo



- Um exemplo: Capacidade de objetos derivados de uma mesma base reagirem de forma diferente
- Ex:
  - Classe Animal possui operação andar
    - Classe Pessoa derivada de animal, quando andar, vai andar da sua forma
    - Classe Cachorro derivada de animal, quando andar, vai andar da sua forma

# Classes abstratas e métodos abstratos



- Classes base para outras classes
- Características:
  - Alguns métodos não possuem implementação (filhos devem implementar)
  - Ajuda bastante em processos de polimorfismo
  - Facilita o reuso de código
  - Facilita a padronização de funcionamento

# Visibilidade



- Privado, público e protegido:
  - Público (public): operação e/ou atributo pode ser acessado de outras entidades.
  - Protegido (protected): operação e/ou atributo pode ser acessado pela entidade onde foi definido e pelos filhos.
  - Privado (private): operação e/ou atributo pode acessado apenas pela entidade onde foi definido.
  - Ver visibilidades específicas de linguagem!

# Interfaces



- Contratos que classes podem assinar
- Imagine a situação:
  - Sistema possui Classes Professor, Aluno.
  - Cliente quer permitir que um professor seja aluno.
  - Como fazer isto no sistema, se muitas linguagens não possuem conceitos de herança múltipla?
  - Solução: Contratos!

# Interfaces



- Solução:
  - Criar um contrato com as operações que um aluno implementa.
  - Criar uma classe derivada de Professor que assina o contrato de aluno.
  - Desta forma se consegue usar conceitos de polimorfismo para resolver o problema.
- Mais reuso de código
- Maior padronização

# Finalizando...



- Revisão de conteúdo...
- Classe vs objeto, tradução do problema de mundo real para o ambiente computacional.
- Visibilidade de informações.
- Associações entre objetos