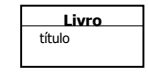
PROGRAMAÇÃO DE COMPUTADORES

Fases na 'fabricação' de um software

- Análise ou especificação
  - Requisitos, conceito do domínio
- Projeto
  - Solução lógica, representação da análise
- Programação, construção ou desenvolvimento
  - Código, representação em uma linguagem de programação;
  - Delphi, C++, Java, Visual Basic, Cobol, etc
- Testes




```
class Livro {
  public:
    void imprimir();
  private
    String titulo;
}
```

Estruturada x Orientação a Objetos

- Com a orientação a objetos procura-se eliminar as diferenças entre as etapas de análise, projeto e implementação, reabilitando a difamada tarefa de implementação
- O segredo é fazer com que os conceitos de programação, e as notações para programação, sejam suficientemente de alto-nível para que possam servir apropriadamente como ferramentas de modelagem.

Metodologia 'estruturada'

- Criada nos anos 60/70
  - –Grandes Projetos
  - –Ambientes de Grande Porte
  - –Pouca maturidade em Programação
- Funciona!
  - –Quando é bem utilizada…
  - –Quando os projetos tem início, meio e fim…

Metodologia 'estruturada'

- Tradicionalmente, a literatura tratava análise, projeto e implementação como atividades diferentes e, até certo ponto, desconexas pois tinham:
  - diferentes métodos;
  - diferentes notações e
  - diferentes objetivos.
- Na necessidade de especificar o que está prestes a ser implementado tratavam a análise e o projeto como as únicas coisas que realmente interessam.
  - A implementação seria apenas algo inevitável.

Paradigma Estruturado vs. Orientação a Objetos

- Estruturada
  - Os sistemas são divididos em subprogramas;
  - Fixa a atenção muito mais nos procedimentos que nos dados.
- Orientação a Objetos
  - Dados e Procedimentos possuem a mesma importância;

Orientação a objetos

- Altera a forma pela qual dados e procedimentos intercomunicam-se.
  - Reutilização de Software (Reusibilidade)
    - reaproveitamento de código
  - Manutenção de Software
    - facilidade de manutenção de sistemas
  - Tamanho de Código Gerado
    - menor código

Orientação a objetos

- Crise do software e necessidade de mudar
  - Dificuldade manutenção, prazos, custos, etc...
- Comparação da evolução entre hardware e software
  - Hardware:
    - Computadores imensos à microprocessadores
  - Software
    - Linguagem de máquina à programação estruturada
- Por que tal diferença?
  - Componentes
    - Funções específicas
    - Trabalho simplificado pelo trabalho de antecessores
    - Criadores de software geralmente partem da areia...

Analogia de David Chappell, http://pt.kioskea.net/contents/poo/poointro.php3

Orientação a objetos

- orientação a objetos:
  - representa melhor o mundo real
  - percepção e o raciocínio do ser humano estão relacionados diretamente com o conceito de objetos.
- Objetivo principal da utilização do paradigma da Orientação a Objetos, na construção de software :
  - Rápido
    - não perder muito tempo no desenvolvimento
  - Barato
    - linhas de montagem e reutilização de código
  - Flexível
    - fácil modificar ou extender

Programação Procedural

- Paradigma: Decida quais procedimentos você quer; utilize os melhores algoritmos que você encontrar
- Enfoque no processamento, o algoritmo necessário para executar uma computação desejada
- Linguagens oferecem facilidades para passar parâmetros para funções e retornar valores
- Funções são utilizadas para criar ordem em um algoritmo complicado

Programação Procedural

```
double sqrt(double arg) {
  // RAIZ QUADRADA
}

void RAIZ() {
  double root2 = sqrt(2);
}
```

Programação Modular

- Mudança de enfoque: projeto de procedimentos -> organização de dados
- Paradigma: um conjunto de procedimentos, juntamente com os dados que eles manipulam, é dito um módulo
- Este paradigma utiliza o princípio de encapsulamento de dados

Programação Modular

- “stack.h” – interface externa

```
// declaration of the interface for
// module stack of characters

void push(char);
char pop();
const int stack_size = 100;
```

Programação Modular

- “stack.c” -- implementação

```
#include “stack.h”
static char v[stack_size];
static char *p = v;

void push(char c) { … }
void pop() { … }
```

Programação Modular

- “some_file.cc”

```
#include “stack.h”

void some_function() {
push(‘c’);
char c = pop();
if (c!=‘c’) error(‘impossible’);
```

Abstração de Dados

- “Tipos” criados por meio de módulos e diferentes “tipos” embutidos na linguagem
- “tipos” embutidos permitem maior verificação e são mais simples

Abstração de Dados

- Paradigma:

  Decida quais os tipos; implemente um conjunto de operações para cada tipo
- “complex.h”

```
class complex {
      double re, im;
public:
      complex(double r, double i)
            { re=r; im=i; }
      complex(double r) // float->complex
            { re=r; im=0; }
```

Abstração de Dados

```
    friend complex operator+(complex,
      complex);
    friend complex operator-(complex,
      complex);
    friend complex operator-(complex);
    friend complex operator*(complex,
      complex);
    /* …. */
}
```

Abstração de Dados

- "complex.cc"

```
complex operator+(complex a1, complex a2) {
    return complex(a1.re+a2.re,a1.im+a2.im);
}
```

- "f.cc"

```
void f() {
    complex a = 2.3;
    complex b = 1/a;
    complex c = a+b*complex(1,2.3);
    // …
    c = -(a/b)+2; }
```

Problemas com Abstração de Dados

```
class point { … };
class color { … };
enum kind { circle, triangle,
    square };
class shape {
    point center;
    color col;
    kind k;
public:
    point where() { return center; }
    void move(point to) {
        center = to; draw(); }
    void draw();
    void rotate();
}
```

```
void shape::draw() {
    switch(k) {
    case circle:
        // draw a circle
        break;
    case triangle:
        // draw a triangle
        break;
    case square:
        // draw a square
        break;
    }
}
```

Problemas com Abstração de Dados

- Funções como "draw" devem saber tudo sobre todos os tipos de objetos gráficos
- O código da função cresce toda a vez que um novo objeto gráfico é adicionado
  - Todas as operações devem ser verificadas
  - Possibilidade de introdução de "bugs"

Programação Orientada a Objetos

- A abordagem anterior não oferece uma distinção entre as propriedades gerais e as específicas dos objetos gráficos
- Programação orientada a objetos permite expressar tais distinções/semelhanças por meio de herança

Programação Orientada a Objetos

```
class shape {
    point center;
    color col;
public:
    point where()
        { return center; }
    void move(point to)
      {center=to; draw();}
    virtual void draw();
    virtual void rotate(int);
}
```

```
class circle : public shape {
    int radius;
public:
    void draw() { … }
    void rotate(int) { …}
}
```

Programação Orientada a Objetos

- Paradigma:
  - decida quais classes você deseja;
  - implemente as operações suportadas por cada classe; e
  - torne explícitas as semelhanças por meio de herança

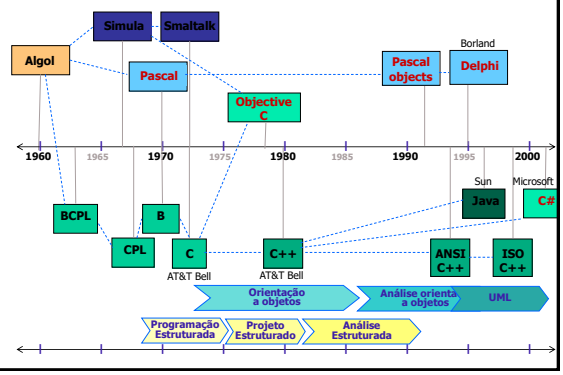
Procedural vs. Orientado a objetos

Procedural	Orientado a objetos
Tipos de dados	Classes
variáveis	Instâncias/objetos
Função/procedimento	métodos
Chamada de funções	mensagens

Evolução da programação através das décadas

- 40 - alteração física de circuitos eletrônicos;
- 50 - linguagens montadoras (assembler), visão lógica;
- 60 - surgem as primeiras linguagens de alto nível;
- 70 – programação estruturada;
- 80 – Orientação a objetos

Evolução



Poderes do C e C++

- C++ é superconjunto do C,
- Fiel à característica do C de linguagem de pequeno tamanho,
  - o C++ incluiu apenas algumas novas palavras reservadas às já existentes no C.
  - Linguagem versátil, concisa e relativamente de baixo nível;
  - Pequeno tamanho
  - Poucos Comandos
  - nível de Bits
  - Eficiência na Manipulação de Memória
  - Portabilidade

Por que o C++

- Linguagem padronizada
  - Faz em um SO e compila em qualquer outro
    - DOS, Windows, UNIX, OS/2, VAX,MacOS, etc
      - Java é interpretado, necessita de uma máquina virtual
  - Uso em SO com pouco recurso memória
    - PalmOS, Tecnologia WAP
- Diversidade de aplicações
  - De aplicações simples a Implementação de SO
- Várias Bibliotecas disponibilizadas
- Utilizada em diversas ferramentas
  - uma das linguagens mais utilizadas na construção de sistemas comerciais