

# Programação Modular

## Projeto Modular

Roberto da Silva Bigonha

Mariza A. S. Bigonha

Laboratório de Linguagens de Programação

Departamento de Ciência da Computação

Universidade Federal de Minas Gerais

Agosto de 2019



*Todos os direitos reservados*

*Proibida a cópia sem autorização dos autores*



## Critérios de Projeto



# Critérios para Projeto Modular

Os seguintes critérios ou normas ajudam a avaliação dos métodos de projeto de software em relação a modularidade:

- Decomposibilidade em módulos
  - Composibilidade de módulos
  - Inteligibilidade modular
  - Continuidade do método de projeto
  - Proteção modular
- Um método de projeto de software somente pode ser chamado de **modular** se satisfizer os critérios acima.



## Critério da Decomposibilidade

- Projeto baseia-se na decomposição de problema em subproblemas
- Solução de cada subproblema desenvolvida separadamente
- Exemplo:
  - Projeto **top-down**, com desenvolvimento a partir de uma especificação
- Contra-exemplo:
  - Módulo de inicialização de todo o sistema.



# Critério da Composibilidade

- Projeto favorece a produção independente de componentes de software que podem ser combinados entre si para produzir **novos** sistemas.
- Relacionado com o desenvolvimento de componentes para construção de novos sistemas e **não** a partir de uma especificação.
- **Reusabilidade é chave neste critério**
- Exemplos: Biblioteca de sub-rotinas  
Pipeline do Unix
- Contra-exemplo: Preprocessadores



# Critério da Inteligibilidade

- Projeto produz módulos que podem ser estudados e entendidos isoladamente.
- Contra-exemplo:
  - Conjunto de módulos que somente funcionam se ativados em certa ordem.



# Critério da Continuidade

- O método de projeto deve ser **contínuo** no sentido em que **pequenas** alterações na especificação afetam **poucos** módulos.

método-de-projeto: Especificação  $\rightarrow$  Sistema

- Relacionado com **extensibilidade**.
- Propagação de Mudanças
- Exemplos:
  - Constantes Simbólicas
  - Referência uniforme
$$x := t.a$$
- Contra-exemplos:
  - Uso de informação sobre a representação física dos dados
  - Arranjos estáticos de PASCAL



# Critério da Proteção

- Módulos são construídos de forma a confinar os efeitos de ocorrências anormais dentro de cada módulo, ou então propagá-las a um número mínimo de outros módulos.
- Propagação de Erros.
- Exemplo:
  - Módulo que lê e valida entrada de dados.
- Contra-exemplo:
  - Tratamento de exceções em PL/I e ADA.



# Noção de Projeto Orientado por Objeto

@Roberto S. Bigonha e Mariza A. S. Bigonha

Projeto Modular

Critérios de Projeto



## Idéia Básica de POO

*Projeto orientado por objetos é o método que produz arquiteturas de software baseadas nos objetos que o sistema manipula.*

- A propriedade básica do método:
  - Evite perguntar O que o sistema faz?
  - Melhor perguntar Sobre o que o sistema faz o que?
- **Olhe primeiro para o dado** é a regra para favorece a **reusabilidade**.
- Questões importantes:
  1. Como encontrar os objetos.
  2. Como descrever os objetos.
  3. Como descrever as relações entre objetos

@Roberto S. Bigonha e Mariza A. S. Bigonha

Projeto Modular

Critérios de Projeto



# Como Encontrar OBJETOS

- Sistema de software deve:
  1. Fornecer respostas a questões do mundo exterior.  
Exemplo: Computação para resolver um problema
  2. Interagir com o mundo exterior.  
Exemplo: Sistema de controle de processo
  3. Criar novas entidades no mundo exterior.  
Exemplo: Editor de texto
- Sistema de software é um **Modelo Operacional**, baseado em *interpretação* do mundo.
- Os objetos que compõem o software devem ser representação dos objetos relevantes que constituem o mundo exterior.

**CONCLUSÃO: Os objetos estão aí; é só pegá-los.**



# Descrição de OBJETOS

- Descrição de objetos deve ser:
  - completa
  - precisa
  - não ambígua
  - independente de representação física
- Sistema de software é um **Modelo Operacional**, baseado em *interpretação* do mundo.
- Os objetos que compõem o software devem ser representações dos objetos relevantes que constituem o mundo exterior.
- Uma solução é: **A Teoria de Tipos Abstrato de Dados**
- O comportamento dos objetos é definido pelo seu **tipo abstrato**.
- Descrição de um ADT deve compreender: **interface do ADT**  
**comportamento do ADT**



# Tipo Abstrato de Dados

- Um tipo abstrato de dados é uma **classe** de estruturas de dados descritas por uma visão externa:
  1. lista de serviços disponíveis
  2. propriedades destes serviços
- A representação das estruturas de dados do ADT ficam completamente encapsulada
- A representação das estruturas de dados do ADT não faz parte de sua definição.
- O adjetivo **abstrato** de um **tipo abstrato de dados** enfatiza o fato de as estruturas de dados que representam o tipo **NÃO** fazem parte da definição, isto é, da interface, do tipo.



## Um Tipo Abstrato de Dados: STACK

**TYPES** STACK[X]

**FUNCTIONS**

- empty: STACK[X]  $\rightarrow$  BOOLEAN
- new:  $\rightarrow$  STACK[X]
- push: X  $\times$  STACK[X]  $\rightarrow$  STACK[X]
- pop: STACK[X]  $\rightarrow$  STACK[X]
- top: STACK[X]  $\rightarrow$  X

**PRECONDITIONS**

- pre** pop(s : STACK[X]) = (**not**) empty(s)
- pre** top(s : STACK[X]) = (**not**) empty(s)

**AXIOMS** for all x:X, s:STACK[X]:

- empty(new())
- not** empty(push(x,s))
- top(push(x,s)) = x
- pop(push(x,s)) = s



# Programação Orientada por Objetos

- **Programação Orientada por Objetos:**
    - é a construção de sistemas de software como uma coleção estruturada de implementações de tipos abstratos de dados.
  - Tipos abstratos de dados:
    - Módulos são construídos com base em abstrações de dados (classes).
  - Coleção:
    - Metodologia baseada na montagem **bottom-up** de classes existentes.
- Estruturada:
- Relação Cliente-servidor e de hierarquia entre classes.



# Ciclo de Vida do Desenvolvimento

- Análise
- Projeto tem por objetivo construir a ponte sobre o **gap semântico**
- Implementação







# Gap Semântico

- Gap semântico é a distância entre os conceitos que seres humanos conhecem e entendem e os conceitos que os computadores manipulam.
- **Exemplo de Abordagem Convencional:**
  - O objetivo principal da **análise** de um sistema de radar é identificar sua função principal.
  - O resultado é coleta de dados e sua exibição na tela do radar.



## ... Gap Semântico

- **Exemplo de Abordagem Orientada a Objetos:**
  - O objetivo principal da **análise** de um sistema de radar é identificar os principais objetos que compõem o sistema.
  - O resultado é uma coleção de classes que descrevem os principais objetos, como por exemplo, aviões, telas, radar, etc.
  - O programa principal tem a função de criar os principais objetos e enviar-lhes mensagens para iniciar a execução.



# Gap Semântico: Exemplo

**SOLUÇÃO I:** Linguagem com apenas booleanos e sem arranjos

```
program Gap;
```

```
var x0,x1,x2,x3,x4,x5,x6,x7,x8,x9 ,x10,x11,x12,x13,x14,x15:Boolean;
```

```
    y0,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10, y11, y12, y13, y14, y15:Boolean;
```

```
    t0,t1,t2,t3,t4,t5,t6,t7,t8,t9 ,t10, t11, t12, t13, t14, t15 : Boolean;
```

```
    c : Boolean;
```

```
begin
```

```
    read(x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15);
```

```
    read(y0,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10, y11,y12,y13,y14,y15);
```

```
    c := false;
```



## .. Gap Semântico: Exemplo

**SOLUÇÃO I (continuação):**

```
t15 := (not x15) and (not y15) and (c) or (not x15) and (y15) and (not c) or  
      (x15) and (not y15) and (not c) or (x15) and (y15) and (c);
```

```
c := (x15 and y15) or (x15 and c) or (y15 and c);
```

```
t14 := (not x14) and (not y14) and (c) or (not x14) and (y14) and (not c)  
      or (x14) and (not y14) and (not c) or (x14) and (y14) and (c);
```

```
c := (x14 and y14) or (x14 and c) or (y14 and c);
```

```
t13 := (not x13) and (not y13) and (c) or (not x13) and (y13) and (not c)  
      or (x13) and (not y13) and (not c) or (x13) and (y13) and (c);
```

```
c := (x13 and y13) or (x13 and c) or (y13 and c); .....
```



## ... Gap Semântico: Exemplo

**SOLUÇÃO I** (continuação):

```
t1 := (not x1) and (not y1) and (c) or not x1) and (y1) and (not c) or  
      (x1) and (not y1) and (not c) or (x1) and (y1) and (c);
```

```
c := (x1 and y1) or (x1 and c) or (y1 and c);
```

```
t0 := (not x0) and (not y0) and (c) or (not x0) and (y0) and (not c)  
      or (x0) and (not y0) and (not c) or (x0) and (y0) and (c);
```

```
println(t0,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15);
```

```
end.
```



## ...Gap Semântico: Exemplo

**SOLUÇÃO II:** Suponha que exista em sua linguagem o tipo integer com a operação +:

```
program Gap;  
  var x : integer;  
      y : integer;  
      t : integer;  
begin  
  read(x);  
  read(y);  
  t := x + y;  
  println(t)  
end.
```

**Por que a Solução II é melhor que Solução I?**



# Enunciado do Exemplo Resolvido

## ❑ **Problema:**

Implementar um programa para ler, somar dois valores inteiros no intervalo  $[-32.768, 32.767]$  e imprimir o resultado.

## ❑ **Análise:**

- ❑ Os valores a serem operados podem ser representados por 16 bits em forma de complemento de 2.
- ❑ Objetos que devem ser modelados pelo software são três valores **inteiros**.
- ❑ **Na solução I**, supõe-se que sua linguagem de programação possua somente o tipo de dados **Boolean**, com as operações **and**, **or** e **not**.
- ❑ **Na solução II**, supõe-se disponível o tipo integer



## Sistemas OO

- As classes formam o núcleo de um programa OO.
- Os objetos provêm o comportamento, devendo ser criados apropriadamente.
- Cada objeto implementa uma pequena parte do comportamento geral da aplicação:
  - Objetos (*transientes*) de Computação
  - Objetos (*persistentes*) de Banco de Dados
  - Objetos de interface
- O programa principal fica reduzido a uma função de criar e inicializar objetos principais e iniciar a computação.

# Objetos e Persistência

- A existência de um objeto transcende o tempo e/ou o espaço de uma execução do programa
- O estado do objeto e o de seus componentes estarão a salvo ao longo do tempo ou espaço.