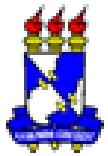
Encapsulamento e Modularização

Prof. Alberto Costa Neto
alberto@ufs.br

Linguagens de Programação



Departamento de Computação
Universidade Federal de Sergipe

Conteúdo

- Encapsulamento
- Modularização
- Tipos Abstratos de Dados (TADs)
- Objetos
- Classes

Encapsulamento

- Programação em grande escala
- A chave da modularidade é abstração
  - Qual é o propósito de um módulo? (**O quê** ele faz?)
  - Como se chega no propósito?
- Abstração exige ocultamento (controle de visibilidade)
  - Interfaces mínimas para outros módulos
- Com o encapsulamento, o usuário de um módulo preocupa-se apenas com **o quê** ele faz, não **como faz**.

Encapsulamento

- Conceitos
  - Módulo: grupo de declarações + encapsulamento
  - TAD: tipo definido indiretamente por suas características e operações exportadas
  - Objeto: variável oculta + operações exportadas
  - Classe: (descrição de) um tipo de objetos
  - *Generics*: declarações com tipos parametrizados

Vantagens do Encapsulamento

- Empacotar representação e operações em unidades lógicas que podem ser compiladas separadamente
- **Flexibilidade:** como os clientes não dependem da representação, ela pode ser modificada sem exigir mudanças nos clientes
- **Confiabilidade:** clientes não podem quebrar a integridade dos dados, seja intencionalmente ou acidentalmente
- **Documentação reduzida:** O cliente só precisa conhecer a parte pública

Módulos (pacotes)

- Módulo (pacote):
  - um grupo de componentes declaradas
  - um conjunto de vínculos encapsulados
- Exemplos
  - Pacotes em Ada e estruturas em ML agrupam qualquer vinculável

```
package P  is
    D
 end;
```

```
structure P =
 struct
    D
 end
```

  - Pacotes em Java agrupam apenas classes
  - Classes em Java, C++, Eiffel, ... agrupam (e encapsulam) variáveis e métodos.

Módulos com encapsulamento

- Ada e Modula-2 dividem o módulo em duas partes separadas: interface e corpo

```
package trig is
    function sin(x:Float)
            return Float;
    function cos(x:Float)
            return Float;
end trig;
```

```
package body trig is
   pi : constant Float := 3.1416;
   function sin(x:Float)
                  return Float is
      ...;
   function cos(x:Float)
                  return Float is
      ...;
end;
```

- Vantagens:
  - Compilação separada da interface e do corpo. Para quê?
  - O cliente/usuário do pacote só conhece a interface

Outras alternativas

- Módulos em ML
  - possui *block declarations*:

    local D_1

    in D_2 **end**
  - combinado com **struct** permite criar módulos com encapsulamento

- Haskell: explicitando listas de exportação e importação
  module P export list_id import list_id from M is D
  - limitação na compilação separada (não há interfaces)

Outras alternativas

- Java, C++, Object Pascal: mecanismos de encapsulamento usando modificadores de visibilidade.

```
class C {
    private int a;
    public void f( ) {...}...;
    protected int g( ) {...}
}
```

```
package P;
import Q;
public class A {...}
```

```
package P;
class B {...}
```

Tipos Abstratos de Dados

- TAD: um tipo definido por um grupo de operações
  - representação dos dados é oculta
- Ex. Suponha que precisamos trabalhar com Racionais

  type Rational = (Int, Int)
  - Rational pode ser usado em contextos indesejados (onde se espera um ponto por exemplo)
  - Podem ser construídos valores sem sentido, ex: (2,0)
  - Valores iguais podem ser considerados diferentes, ex: (4,2) e (2,1)
- Solução: esconder a representação ⇒ um TAD
  - todas as vantagens do encapsulamento
  - resolvem-se os três problemas anteriores

Um TAD em ADA

```
package directory_type is
   type Directory is limited private;
   procedure insert (dir : in out Directory;
                     newname: in Name; newnumber: in Number);
   procedure lookup (dir : in Directory;
                     oldname: in Name; oldnumber: out Number
                     found : out Boolean);
private
   type DirNode;
   type Directory is access DirNode;
   type DirNode is record .... end record;
end directory_type;
```

Implementação do TAD

```
package body directory_type is
   procedure insert (dir : in out Directory;
                     newname: in Name; newnumber: in Number) is
         ...; -- aqui a implementação
   procedure lookup (dir : in Directory;
                     oldname: in Name; oldnumber: out Number
                     found : out Boolean) is
         ...; -- aqui a implementação
end directory_type;
```

Usando o TAD

```
use directory_type; -- permite abreviar a notação ponto
homedir : Directory;
workdir : Directory;
...
insert(workdir, me, 6041);
insert(homedir, me, 8715);
lookup(workdir, me, mynumber, ok);
```

TADs em outras linguagens

- Algumas linguagens permitem definir TADs
  - Com construções próprias, por exemplo ML e Haskell 96
  - Usando módulos -- algumas linguagens permitem outras não (Units do turbo pascal?).

Objetos

- Objeto: uma variável escondida com um grupo de operações visíveis que trabalham sobre esta variável
- Instâncias de um TAD (classe)
- Todas as vantagens dos TADs
- Introduz um novo paradigma
- Geralmente pacotes/módulos suportam objetos (Units de Pascal?)

Um Objeto em ADA

```
package directory_object is
   procedure insert (newname: in Name
                     newnumber: in Number);
   procedure lookup (oldname: in Name;
                     oldnumber: out Number
                     found : out Boolean);
end directory_object;
```

Nota: ao contrário das definições anteriores, as operações não recebem mais como parâmetro um objeto **directory_object**

Declarando um Objeto em Ada

```
package body directory_object is
      type DirNode;
      type DirPtr is access DirNode;
      type DirNode is record .... end record; -- uma árvore
     root : DirPtr;   - - objeto
     procedure insert (newname: in Name
                                newnumber: in Number) is
          ...; -- adiciona na árvore cuja raiz é root

     procedure lookup (oldname: in Name;
                                oldnumber: out Number
                                found : out Boolean) is
          ...; -- busca na árvore cuja raiz é root
end directory_object;
```

Usando um Objeto em Ada

```
directory_object.insert(me, 6041);
directory_object.insert(myMother, 8715);
...
directory_object.lookup(me, mynumber, ok);
```

Classes de Objetos

- Classe: um "tipo" de objetos (uma forma de definir TADs)
- Em Ada: usando pacotes genéricos
- Módulos de outras linguagens em geral não suportam classes
- Em Linguagens OO (Java, C++, Smalltalk) há uma construção linguística específica para definir classes

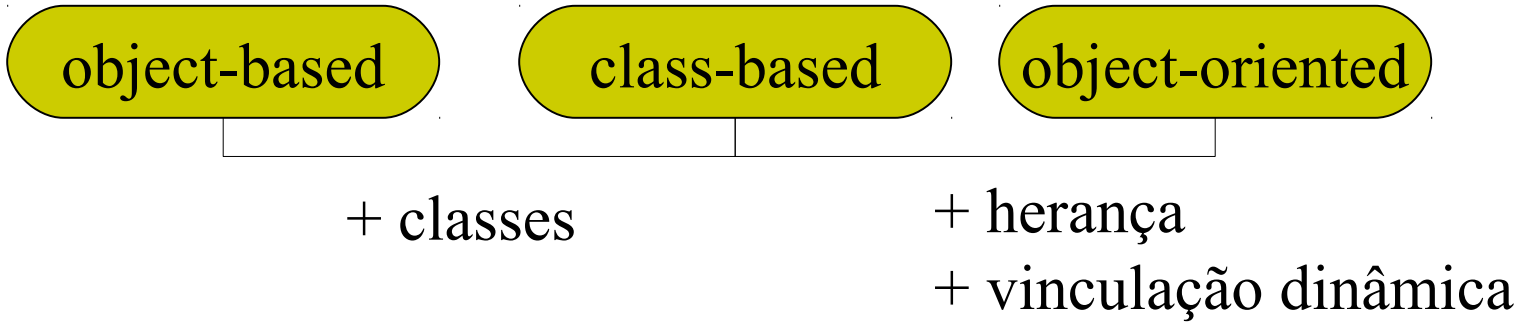
Uma classe em ADA

```
generic package directory_class is
    -- idem interface de directory_object
end directory_class;
package body directory_class is
    -- idem corpo de directory_object
end directory_class;

package homedir is new directory_class;
package workdir is new directory_class;

workdir.insert(me,6041);
homedir.insert(me, 8715);
workdir.lookup(me,mynumber,ok);
```

Baseadas ou Orientadas em/a Objetos?



- Linguagens Baseadas em Objetos: facilmente pode-se definir objetos (Modula2)
- Baseado em Classes: baseado em Objetos e também pode-se definir classes (ADA)
- Orientado a Objetos: Baseado em Classes + herança + vinculação dinâmica (C++, Java)

Sugestões de Leitura

- Concepts of Programming Languages (Robert Sebesta)
  - Seções 11.1 a 11.4
- Programming Language Concepts and Paradigms (David Watt)
  - Seções 6.1, 6.2 e Capítulo 7
- Linguagens de Programação (Flávio Varejão)
  - Capítulo 6