7- Encapsulamento

- Sistemas de programação de grande porte são constituídos de módulos
- Um módulo é qualquer unidade de programa que possua um nome e que possa ser implementada como uma entidade independente
- Um módulo bem projetado tem um único objetivo e possui uma interface pequena com outros módulos $\Rightarrow$ é reutilizável
- A chave para modularidade é a abstração
- Diz-se que um módulo encapsula seus componentes. Esses componentes podem ser tipos, constantes, variáveis, procedimentos, funções, etc
- *Componentes exportáveis* de um módulo são os componentes que são visíveis externamente ao módulo. Os componentes escondidos (ocultos) são usados somente para auxiliar a implementação dos componentes exportáveis

7- Encapsulamento...

- Pacotes

  Pacotes simples

  - Um pacote é especificado através de uma lista de informações declarativas. Em geral, essas informações podem ser qualquer denotável da LP, tais como tipos, constantes, variáveis, procedimentos, funções e outros pacotes
  - Um pacote pode ser visto como um conjunto encapsulado de amarrações
  - Exemplo: Ada

```
package I is D end I;

package conversao_metrica is
  pol_ cm:        constant Float := 2.54;
  pe_ cm:         constant Float := 30.48;
  jarda_ cm:      constant Float := 91.44;
  milha_ km:      constant Float := 1.609;
end conversao_metrica;
```

7- Encapsulamento...

Ocultamento de informação (*information hiding*)

- Em geral, um pacote contém declarações de componentes exportáveis e escondidos. A distinção, em Ada, é feita através da divisão de um pacote em duas partes:
  - declaração do pacote – declara somente os componentes exportáveis
  - corpo do pacote – contém declarações de todos os componentes escondidos, além dos corpos de procedimentos e funções exportáveis
- A declaração do pacote contém somente as informações necessárias ao usuário para que ele possa utilizar o pacote, dando condições ao compilador de realizar verificações de tipo
- Exemplo: Ada

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

package body trig is
  pi: constant float := 3.1416;
  function sin (x: in float) return float;
    -- comandos para calcular o seno de x
  function cos (x: in float) return float;
    -- comandos para calcular o coseno de x
end trig;
```

7- Encapsulamento...

- Tipos abstratos
  - Considere as seguintes definições de tipos:

    type racional = int * int;
    type data = int * int;

    Qual é o tipo da expressão (1, 12)? É possível comparar um valor do tipo **data** com um valor do tipo **racional**?
  - Quando representamos um tipo através de um outro tipo, algumas dificuldades podem ocorrer:
    - o tipo da representação pode possuir valores que não correspondem a qualquer valor do tipo desejado
    - comparações com resultados incorretos
    - os valores de um tipo podem ser confundidos com valores do outro tipo, salvo se for declarado um novo tipo
  - Um tipo abstrato é um tipo definido através de um grupo de operações
  - Tipicamente, o programador escolhe uma representação para os valores do tipo abstrato e implementa as operações em termos da representação escolhida. A representação é escondida; o módulo exporta somente o próprio tipo abstrato e suas operações

7- Encapsulamento...

- Exemplo: (Ada)

```
package tipo_turma is
  type id_aluno is integer;
  type Turma is limited private;
  procedure inclui_aluno (t: in out Turma; aluno: in id_aluno);
  procedure cria_turma (t: in out Turma; prof: in integer; sala: in integer);
  private
    tam_max: constant integer := 65;
    type Turma is
      record
        sala: integer;
        professor: integer;
        tam_classe: integer range 0..tam_max := 0;
        lista_classe: array (1..tam_max) of id_aluno;
       end record;
end tipo_turma;

package body tipo_turma is
  procedure inclui_aluno (t: in out Turma; aluno: in id_aluno) is
                -- código para inclusão de um aluno na turma
  procedure cria_turma (t: in out Turma; prof: in integer; sala: in integer) is
                -- código para criar uma nova turma
end tipo_turma;
```

7- Encapsulamento...

- Com um tipo abstrato, não interessa se um dado valor do tipo possui várias representações possíveis, pois as representações estão escondidas do usuário. O que é importante é que somente propriedades desejadas dos valores são observáveis, usando as operações associadas ao tipo abstrato
- Uma representação de tipo abstrato sempre pode ser alterada, sem haver a necessidade de realizar alguma alteração externamente ao módulo
- Definir um tipo abstrato exige mais esforço
- É comum um tipo abstrato fornecer operações construtoras, para compor valores do tipo abstrato e operações destrutoras, para decompor tais valores
- Tipos abstratos são similares aos tipos pré-definidos.

  Ex: turmaA: Turma;

7- Encapsulamento...

- Objetos e classes

  Objetos simples

  – O termo objeto é freqüentemente utilizado para uma variável escondida em um módulo (ou o próprio módulo), sendo que esse módulo contém operações exportáveis sobre esta variável. A vantagem disso é que alterações na representação da variável não provocam alterações externas ao módulo. Em geral, a variável é uma estrutura de dados

  – Objeto tem tempo de vida, que em geral, é definido da mesma forma que para variáveis locais

7- Encapsulamento...

Classes de objetos

- Como definir classes de objetos similares?
- Exemplo: (Ada)

```
generic package tipo_turma is
  type id_aluno is integer;
  procedure inclui_aluno (aluno: in id_aluno);
  procedure cria_turma (prof: in integer;  sala: in integer);
end tipo_turma;

package body tipo_turma is
   tam_max: constant integer := 65;
   type Turma is
      record
         sala: integer;
         professor: integer;
         tam_classe: integer range 0..tam_max := 0;
         lista_classe: array (1..tam_max) of id_aluno;
      end record;

  procedure inclui_aluno (aluno: in id_aluno) is
       -- código para inclusão de um aluno na turma
  procedure cria_turma (prof: in integer;  sala: in integer) is
       -- código para criar uma nova turma
  begin
    ...; -- inicializa a turma
end tipo_turma;
```

7- Encapsulamento...

A elaboração do pacote genérico anterior, simplesmente amarra ***tipo_turma*** a uma classe de objetos. Para criar objetos, deve instanciar o pacote genérico:

package turmaA is new tipo_turma;

Assim pode-se acessar esse objeto ***turmaA***:

turmaA.cria_turma(10,222);
turmaA.inclui_aluno(13892);

- Qual a diferença entre um tipo abstrato e uma classe de objetos?
  - Tipos abstratos – operações (funções e procedimentos) possuem um parâmetro a mais $\Rightarrow$ Na chamada de uma operação de um tipo abstrato, todos os argumentos estão explícitos
  - Classes de objetos – instâncias diferentes definem procedimentos distintos, sendo que cada um deles acessa um objeto de dados distinto. Procedimentos e funções têm como que um parâmetro implícito
  - Tipos abstratos são semelhantes aos tipos pré-definidos $\Rightarrow$ a definição de um novo tipo abstrato amplia a variedade de tipos fornecidos ao programador

7- Encapsulamento...

- Genéricos

  Abstrações genéricas

  – Uma abstração genérica é uma abstração sobre uma declaração. Em outra palavras, uma abstração genérica possui um corpo que é uma declaração e uma instanciação genérica é uma declaração que produzirá amarrações através da elaboração do corpo de uma abstração genérica

  – Exemplo: (Ada) – pacotes genéricos também podem ter parâmetros

```
generic
  type Elem is private;
  tam: in positive;

package tipo_pilha is
  type Pilha is private;
  procedure empilha (i: in Elem; s: in out pilha);
  procedure desempilha (i: out Elem; s: in out  pilha);
  private
    type Pilha is
      record
        dados: array (1.. tam) of Elem;
        topo: integer range 0..tam := 0;
      end record;
end tipo_pilha;

package body tipo_pilha is
   procedure empilha (i: in Elem; s: in out pilha);
     -- código para inclusão de elemento na pilha
   procedure desempilha (i: out Elem; s: in out  pilha);
     -- código para retirar elemento da pilha
end tipo_pilha;
```

Essa definição de tipo genérico pode ser instanciada para produzir uma pilha de inteiros ou de turmas:

```
package Pilha_Inteiros is new tipo_pilha (Elem => integer, 100);
package Pilha_Turmas is new tipo_pilha (Elem => Turma, 60);
```