Inkapsling, modularitet

- Paket
- Abstrakta datatyper
- Klasser och objekt
- Generiska moduler

Inkapsling, Modularitet

- Konstruktion av stora program
- Innehåller
  - Typer, Konstanter, Procedurer, Funktioner, Osv.
- Exporterar
  - Komponenter som ska vara synliga utanför modulen.
- Gömmer
  - Komponenter i modulen som endast är tillgängliga i modulen.

Enkla paket

- Grupp av deklarerade komponenter
- Kan ses som en inkapslad mängd bindingar
- Ada: **package** I **is** D **end** I
- ML : **structure** I = **struct** D **end**
- Exporterar alla deklarationer

Information Hiding

- Package deklaration
  - Endast de komponenter i modulen som ska exporteras
- Packages body innehåller implementationen

```
package trig is
  function sin (x:Float) return Float;
  function cos (x:Float) return Float;
end trig
package body trig is
  pi : constant := 3.14...;
  function norm.....;
  function sin ...;
  function cos ...;
end trig
```

Information hiding ML

- *signature* innehåller deklarationer som ska exporteras
- strukturer innehåller implementationen

```
signature TRIG =
  sig
      val sin : real -> real
      val cos : real -> real
  end
```

(forts.)

```
structure Trig : TRIG =
  struct
      val pi = ...
      fun norm x ...
      fun sin ...
      fun cos ...
  end
```

Abstrakta datatyper

- Vi måste välja en representation
- Den representationen kan ha egenskaper som inte är önskvärda för den nya typen

```
datatype rational = rat of int * int;
val zero = rat (0,1)
and one = rat (1,1);
fun op ++ (rat (m1,n1),rat (m2,n2) =
        rat (m1*n2+m2*n1,n1*n2)
```

Exempel *Rational*

- Typen rational =
  {rat (m,n) | n,m ∈ integer)}
- Önskvärt rational =
  {rat (m,n) | n,m ∈ Integer; n> 0 ,
  m och n har inga gemensamma faktorer}

Problem

- Om typ S representerar en önskad typ T
- S kan ha värden som den T inte har
  $x \in S$ men $x \notin T$
- S kan ha flera värden som motsvarar samma värde hos T
  $x,y \in S$ och $x \neq y$ men $z \in T$ och $z = x$ och $z = y$
- Värden av den önskade typen kan förväxlas med värden av representationstypen
  - Använd nytyp deklaration

Abstrakta datatyper

- Abstrakta datatyper undviker dessa problem
- En abstrakt datatyp är:
  - En typ
  - En grupp operationer på typen.
  - Värden av typen definieras endast indirekt

ML

```
abstype rational = rat of (int * int)
with
    val zero = rat (0,1)
    val one = rat (1,1)
    fun op // (m,n) =
        if n > 0 then rat (m,n)
        else ..... (* Error*)
    and op ++ (rat(m1,n1),rat(m2,n2)) =
            rat (m1*n2 + m2*n1,n1*n2)
    and op == (rat(m1,n1),rat(m2,n2)) =
            m1*n2 = m2*n1
end
```

- Endast implementatören kan se representationen

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;
```

Ada

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

use directory_type;
homedir : Directory;
insert (homedir, me, 6042);
```

Objekt och Klasser

- Enkla objekt
  - En variabel tillsammans med exporterade operationer
  - Variabeln är gömd, den är endast åtkomlig genom operationerna
  - Variabelns representation kan förändras utan att någonting utanför modulen behöver ändras
- Klasser
  - En grupp objekt med samma egenskaper
  - Definition av klassen, objekten blir instanser av klassen

Objekt i Ada

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

```
package body directory_object is
   type dirnode;
   type DirPtr is access DirNode;
   type DirNode is      record
                          entryname          : Name;
                        entrynumber: Number;
                        left,right     : DirPtr;
                      end
   root : DirPtr
   procedure insert (newname : in Name;
                     newnumber : in Number) is ... ;
   procedure lookup (oldname : in Name;
                     oldnumber : out Number;
                     found : out Bolean) is ... ;
```

```
begin
   ....; -- initialize the directory
end directory_object;

directory_object.insert(me,6044);
directory_object.lookup(me,mynumber,ok);
```

Objektklasser

- Ada: **generic package** istället för **package**
- För att skapa enskilda objekt:
  package homedir **is new** directory_class

Objekt och abstrakta datatyper:

- Abstrakta datatyper liknar inbyggda typer
- Annorlunda notationen, explicita eller implicita värden
- Abstrakta datatyper passar i alla programmeringsparadigm
- Objekt passar endast i en imperativ stil.

Generisk abstraktion

- En abstraktion över deklarationer
- Har en kropp som är en deklaration
- En **generisk instansiering** är en deklaration som producerar bindningar genom att använda den generiska abstraktionens kropp

Ada: generiska paket

```
generic
        capacity: in Positive;
        type Item is private;
package queue_class is
        procedure append (newitem: in Item);
end queue_class;
```

```
package body queue_class is
        items : array (1..capacity) of Item;
        size, front, rear : Integer range 0..capacity;
    procedure append (newitem: in Item) is …

begin
        front :=1; rear :=0;
end queue_class;


package line_buffer is
                new queue_class (120, characters);
```

Beroende parametrar

- **generic**
  type Item **is** private
  type Sequence **is array** (Integer **range** <>) **of** Item
  with function precedes (x,y:Item) **return** Boolean
- **package** sorting **is**
  procedure sort (seq: **in out** Sequence);
  procedure merge.....
- **end** sorting;

- **type** Floatsequence **is array** (Integer **range** <>) **of** Float;
- **package** ascending **is**
  new sorting (Float, Floatsequence, "<=");

ML

- Abstrakta datatyper i ML är automatiskt generiska
- Tack vare polymorfisk typinferens kan även funktioner parametriseras m.a.p. typer

Konstruktion av stora program

- Stort program kräver inte bara ett programspråk
  - Administration, många personer är inblandade
  - Metodologier för stora program
  - Verktyg som automatiserar rutinarbetet

Konstruktion av stora program (forts.)

- Modularisering är ett nyckelbegrepp.
- Givet ett programspråk
  - Vilka fysiska moduler finns i språket?
  - Hur kan fysiska moduler kombineras?
  - Kan specifikation och implementation separeras?
  - Kan man separatkompilera olika fysiska moduler?