Abstract Data Types

- Type representation and operations on that type are defined together.
- Representation is hidden from user of the type -- objects of type t can only be manipulated by operations defined for t.
- Advantages of user-defined ADTs
  - → encapsulation
  - → protection
  - → extensibility
- We'll look at three languages:
  - → Simula 67
  - → Ada
  - → Modula-2

Simula 67: Classes

- A class consists of:
  - → variable declarations
  - → procedure declarations
  - → code (for initialization)
- If C is a class with variables $x_1 \ldots x_n$ and procedures $p_1 \ldots p_k$, an instance of C is a dynamically created object, say r.

```
ref (C) r;
...
r :- new C;
...
. . .r.x_i. . .
. . .r. p_j (y_1 . . . y_m). . .
```

Stack Example

```
class stack;
    begin
        integer array a(1 . . 100);
        integer top;

     boolean procedure empty;
         ...
     end;

     procedure push (element);
     ...
     end;

     procedure pop;
     ...
     end;

     procedure look;
     ...
     end;

                              <-- initialization code
    top := 0;
    end stack;
```

Using the Stack Class

```
ref (stack) s1,s2;
. . .
s1 :- new stack;
s2 :- new stack;
s1.pop;          -- error
s1.push(5);
. . .
s1.look;         -- 5
. . .
s2.look ;        -- error
. . .
```

→ **But no protection!**

```
s2.a(4) := 1000;
s1.top := 0;
```

allowed, but unsafe.

Inheritance in Simula

- **If x is a subclass of y, then instances of x have all of x's attributes plus all of y's attributes.**

  → **x *inherits* the attributes of y.**

- **Example: defining a heterogeneous stack**

```
class stack_mem
  begin ref(stack_mem) next_mem
    next_mem :- none
  end stack_mem;
```

Example Continued: Define stack

```
class stack;
  begin
    ref (stack_mem) first;
    ref (stack_mem) procedure top
      top :- first;
  procedure pop;
    if not(empty) then
      first :- first.next_mem;
  boolean procedure empty;
    empty := (first = = none);
  procedure push(e);
    ref(stack_mem) e;
  begin
    if first =/= none then
      e.next_mem :- first;
    first :- e;
  end
first :- none;
end stack;
```

Example Continued: Stackable Objects

- **Stackable objects must be instances of a *subclass* of stack_mem:**

```
stack_mem class complex(. . .)   -- declare complex as
                                    subclass of stack_mem

...

end complex
```

- **Another example:**

```
class mammal;

mammal class dog;

mammal class cat;

dog class golden_retriever;
```

Packages in Ada

- Two parts:
  - → specification: provides interface, defines visibility.
  - → body: provides implementation
- Important:
  - → Support separate compilation so that if package p1 uses package p2, p1 can be compiled given only the specification part of p2.

Package Example

```
package stack is                                    -- the specification
   type stacktype;
   function empty (s: in stacktype)
                        return boolean;
   procedure push (e: in integer;
                        s: in out stacktype);
   procedure pop (s: in out stacktype);
   function top(s: in stacktype)
                        return integer;
end stack;

package body stack is                               -- the body
   type stacktype is . . .
   function empty (. . .) is . . .
...
```

Package stack (continued)

- Does our separate compilation rule hold:
  - → No!
  - → Definition for stacktype must be in the interface too.
- Problem: We didn't want stacktype's definition to be exported.
  - → Solution: Divide the specification into a *public* part and a *private* part.

New Specification for stack

```
package stack is                                    -- the visible part
    type stacktype is private;
    function empty(. . .) . . .
    procedure push . . .
    ...
private                                             -- the private part
    type list_type is array (1. .100) of int;
    type stacktype is
        record
            list : list_type;
            top : integer range 0. .100 := 0
        end record;

end stack;
```

Using Packages

```
with stack;
procedure p is
    s : stack.stacktype;
    begin
  ..
    stack.push(4,s);
    ...stack.top(s)...;
    ...
    end
```

OR...

```
with  stack; use stack;
procedure p is
    s : stacktype;
    begin
    ...
    push(4,s);
    ...top(s)...;
    ...
    end
```

Modules in Modula-2

- **Very similar to Ada packages, but only pointer types can be exported.**

```
Definition module stack;                    -- public
    type stacktype;
    procedure empty . . .
    ...
end stack;

Implementation module stack;                -- private
    type stacktype = pointer to record
                        list : . . .
                        topsub: . . .
```

Modula-2 Modules (continued)

- **What are the repercussions of this design decision?**
  - → separate compilation is easy (+)
  - → module must supply a creation/initialization routine (-)
  - → extra use of pointers (-)