Want to follow along?

http://www.cs.york.ac.uk/~ndm/ads.pdf

Ada: Generics

Neil Mitchell

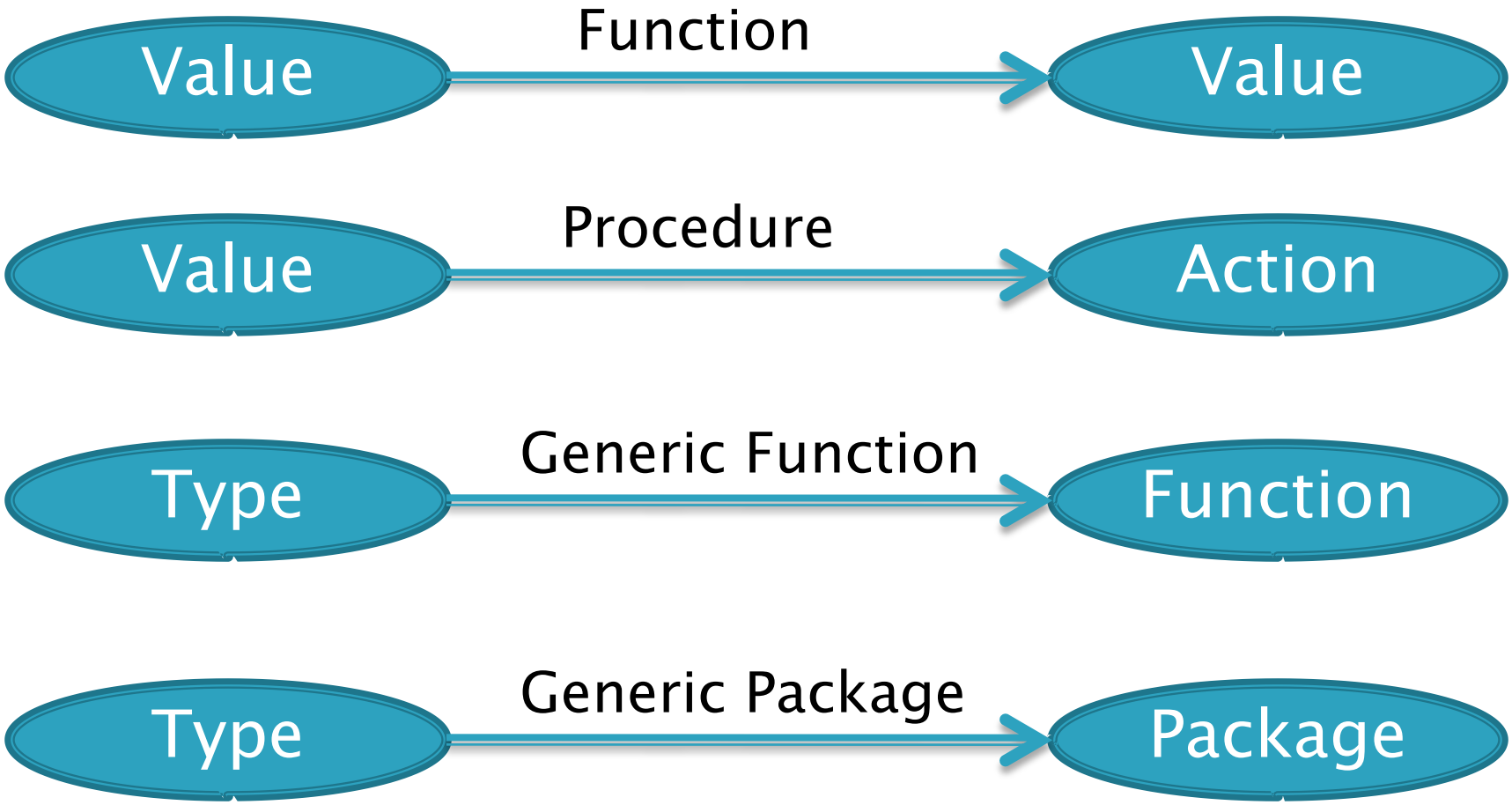
Again, again, again

- Dave writes a linked list package for characters
- Sue writes a linked list package for integers
- Ed writes a linked list package for booleans

- What if Dave had written a linked list package for <anything>?
- Sue and Ed could have gone to the pub!

- **Generics** allows Dave to do this

Generic?


Value
Function
Value
Value
Procedure
Action
Type
Generic Function
Function
Type
Generic Package
Package

Swap_Float

```
procedure Swap_Float(X,Y: in out Float) is
    T : Float;
begin
    T := X;
    X := Y;
    Y := T;
end;
```

Swap (generic procedure)

```
generic
    type Item is private;
procedure Swap(X,Y: in out Item);

procedure Swap(X,Y: in out Item) is
    T: Item;
begin T := X; X := Y; Y:= T; end Swap;

-- an instantiation, which we use
procedure Swap_Float is new Swap(Float);
```

Generic package specification

```
generic
    type Element is private;
package List is
    type List is private;
    Nil : constant List;
    function Null_Query(L : List) return Boolean;
    function Cons(Head : Element; Tail : List)
                    return List;
    ... -- other useful methods
private
    ... -- as before
end List;
```

Note:
Save as “list.ads”

Generic package body

```
package body List is
    function Null_Query (L : List) return Boolean is
    begin return L = Nil;
    end Null_Query;

    function Cons(Head : Element; Tail : List)
                  return List is
    begin
        return new Cell'(Content => Head
                         ,Next => Tail);
    end Cons;
end List;
```

Note:
Save as “list.adb”

Using a generic package

```
with List; -- import

procedure Test is
    -- instantiate
    package List_Integer is new List(Integer);

    -- use
    Ns : List_Integer.List := List_Integer.Nil;
begin
    Ns := List_Integer.Cons(Head => 6, Tail => Ns);
    ...;
end Test;
```

Note:
Save as “test.adb”

Generic type parameters

```
generic
   type Element is <something>;
package List is
```

- **limited private** = use as parameter type, declare variables
- **private** = **limited private** + assign and test for equality
- **(<>)** = **private** + treat as discrete type (T'First, T'Range, etc)

Procedures as Parameters

- **Request:** print a list
- `procedure Put(L : List);`

- Impossible!
- An item in list is *generic*
- We don't know how to write it to the screen

- **Solution**: the *user* tells us how
- `with procedure Element_Put(E : in Element);`

Implementing Put

```
generic
    type Element is private;
    with procedure Element_Put(E : in Element);
package List is
    procedure Put(L : in List);
    ... -- as before
end List;

procedure Put(L : in List) is
begin
    if not Null_Query(L) then
        Element_Put(Head(L));
```

Using Put

```
with Ada.Text_IO, List;
procedure Test is
    package List_Char is new List
        (Element => Character
        ,Element_Put => Ada.Text_IO.Put);

    Hi : List_Char.List := ...;
begin
    List_Char.Put(Hi);
end Test;
```

One package, one value?

- Each List package provides an unlimited number of values:
  - Every operation takes a List parameter
  - Which data structure to operate on
  - Abstract Data Type (ADT)

- An alternative is to have one value in one package
  - No more saying which data structure
  - Can sometimes be simpler
  - Abstract Object

Abstract Object: Specification

```
generic
    type Element is private;
    with procedure Put(E : in Element);
package One_List is
    -- Note: no exported type or constants!
    -- All the state is in the body

    -- A new facility (replaces constant Nil)
    procedure Reset;
    function Null_Query return Boolean;
    procedure Cons(Head : in Element);
    ...
    -- Note: no private section!
end One_List;
```

Abstract Object: Body

```
with List;
package body One_List is
    package List_Element is new List
            (Element => Element, Put => Put);
    The_List : List_Element.List;

    function Null_Query return Boolean is
    begin return List_Element.Null_Query(The_List);
    end Null_Query;

    procedure Cons(Head : in Element) is
    begin The_List := List_Element.Cons(Head, The_List);
    end Cons;

begin
    Reset;
end One_List;
```

Abstract Object: Use

```
with One_List;
procedure Classify is
    package Marks is new One_List(Integer, Num_Out);
    package Age   is new One_List(Natural, Num_Out);
begin
    Marks.Cons(99); -- cheated, not caught
    Marks.Cons(70); -- revised hard
    Marks.Cons(-5); -- cheated, caught, expelled
    Marks.Put;
    Marks.Reset;    -- get rid of last years marks

    if Age.Null_Query then
        Age.Cons(21);
```

What to do now?

- Work through the exercises
  - No more of us talking at the start of practicals
  - Go from where you are
  - Get as far as you can

- If you are struggling
  - Stick your hand up, get some help *now*
  - You *will* be expected to be able to program Ada for the open assessment
  - You *will not* get Ada help during that time