31 - Surcharge de sous-programmes

Programmation Concurrente - LI330
Université P. & M. Curie - année scolaire 2013/2014

PrC

Signature d'un sous-programme

- La signature d'un sous-programme permet de le repérer de manière unique dans un ensemble de sous-programmes

- Les critères possibles pour déterminer cette signature
  1) Utilisation de l'identificateur du sous-programme
  2) Utilisation du type de valeur rendue
  3) Utilisation des paramètres du sous-programme

- Selon ce que l'on utilise, on peut repérer les sous-programme de manière plus ou moins fine
  - En Ada, surcharge possible
    - Signature sur la base de (1), (2), (3)
  - Certains langages ne permettent pas la surcharge (ou selon certaines conditions)

Surcharge

- C'est la possibilité de réattribuer un nom de sous-programme
  - Cela implique que l'on puisse différentier les sous-programmes entre eux
    - Utilisation des 3 critères (identificateur, valeur de retour, type des paramètres)
  - Utilisation d'une notation pointée
    `nom_paquetage.nom_sous-programme`
- On peut surcharger
  - Des procédures ou fonctions définies dans un paquetage
  - Des opérateurs prédéfinis du langage (sous réserve de certaines contraintes)
    - Nombre d'arguments
    - Types rendus

Surcharge sur le paquetage Complexe (1)


```
with Mes_Definitions_Numeriques;
use  Mes_Definitions_Numeriques;

-- Unité de gestion de nombres complexes
package Complexe is

   -- Définition de la structure de données "complexe"
   type Un_Complexe is record
      R, I : Un_Reel := 0.0;
   end record;

   -- Définition des primitives de manipulation d'un complexe
   function Nouveau_Complexe (Re, Im : Un_Reel) return Un_Complexe;
   function "+" (C1, C2 : Un_Complexe) return Un_Complexe;
   function "-" (C1, C2 : Un_Complexe) return Un_Complexe;

   -- Définition des E/S sur un complexe
   procedure Get (C : out Un_Complexe);
   procedure Put (C : in Un_Complexe);

end Complexe;
```

Surcharge d'opérateurs du langage

Surcharge sur le paquetage Complexe (2)


```
with Ada.Text_Io;
Use Ada.Text_Io;


package body Complexe is

   function Nouveau_Complexe (Re, Im : Un_Reel) return Un_Complexe is
   begin
      return (Re, Im );
   end Nouveau_Complexe;

   function "+" (C1, C2 : Un_Complexe) return Un_Complexe is
   begin
      return (C1.R + C2.R, C1.I + C2.I);
   end "+";

   function "-" (C1, C2 : Un_complexe) return Un_Complexe is
   begin
      return (C1.R - C2.R, C1.I - C2.I);
   end "-";
```

Surcharge sur le paquetage Complexe (3)


```
   procedure Get (C : out Un_Complexe) is

   begin
      Put_line ("Partie reelle : ");
      Get (C.R);
      Put_line ("Partie imaginaire : ");
      Get (C.I);
   end Get;

   procedure Put (C : in Un_Complexe) is

   begin
      Put (C.R);
      Put (" + ");
      Put (C.I);
      Put_line (" * i");
   end Put;

end Complexe;
```

Surcharge sur le paquetage Complexe (4)


```
with Complexe, Ada.Text_Io;
use Complexe, Ada.Text_Io;

procedure Teste_Complexe is

   C1, C2, C3, C4 : Un_Complexe;

begin
   -- Saisie de deux nombres complexes
   Put_Line ("Entrez C1");
   Get (C1);
   Put_Line ("Entrez C2");
   Get (C2);
   -- Opérations sur ces nombres
   C3 := C1 + C2;
   C4 := C1 - C2;
   -- Affichage du résultat
   Put ("C3 = ");
   Put (C3);
   Put ("C4 = ");
   Put (C4);
end Teste_Complexe;
```

Notes sur la surcharge

Il faut procéder à une surcharge par possibilité d'invocation

```
-- Min sur des réels
function Min (A : in Un_Reel;
              B : in Un_Reel) return Un_Reel is ...
-- Min sur des entiers
function Min (A : in Un_Entier;
              B : in Un_Entier) return Un_Entier is ...
-- Variables
E1, E2, E3 : Un_Entier;
R1, R2, R3 : Un_Reel;
-- ...
E1 := Min (E2, E3); -- Ca compile
R2 := Min (R2, R3); -- Ca compile
E1 := Min (R2, E3); -- Ca ne compile pas (pas de surcharge pour réel x entier)
R2 := Min (E2, R3); -- Ca ne compile pas (pas de surcharge pour entier x réel)
```

Surcharge d'opérateurs en Ada

- Ada: possibilité de surcharger les opérateurs suivants
  - +, -, *, /, abs, mod, rem, **, &, and, or, xor, not
    - Fonctions avec 2 arguments
  - =, <, <=, >, >=
    - Fonction rendant un booléen avec 2 arguments de même type

- Equivalence d'écriture
  - "+" (A, B)
  - A + B