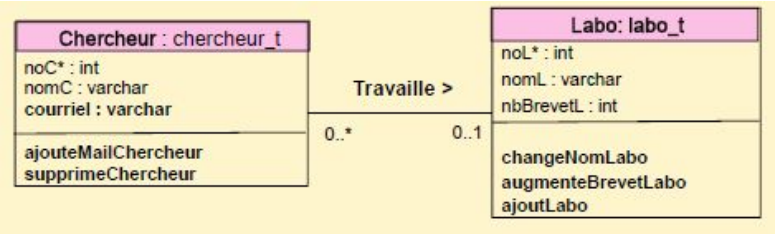
Dr. Rim Samia Kaabi

Méthodes STATIC, MEMBER et les constructeurs

1

Schéma incluant des méthodes




```
ALTER TYPE  labo_t ADD MEMBER Function resetNbBrevet (nb IN int) Return number Cascade ;
ALTER TYPE  labo_t DROP MEMBER Function LePlusGrosLabo  Return Number Cascade ;
```

2

Déclaration

```
CREATE TYPE chercheur_t as Object (
    noC NUMBER,
    nomC varchar(40),
    courriel : varchar (40) ,
    MEMBER Procedure AjouteMail,          -- le paramètre pas essentiel
    MEMBER  Procedure SupprimeChercheur,
    MEMBER  Procedure MAJAdresse ( adresse IN varchar)
    /
```

3

Appel de la méthode AjouterMail()

- Create Table Chercheur of chercheur_t;

Le **select** peut-être autorisé sans restriction, car il ne met pas en danger la cohérence de la base. Sauf le Select …FOR UPDATE.

- La variable v_o est de type chercheur_t : -- applicatif

```
Declare
v_o  chercheur_t;
Begin
Select Value(c) into v_o From Chercheur c Where c.noC = 100;
v_o. courriel := 'abc@utm.com' ;
 v_o.AjouteMail ;
End;
```

La méthode AjouteMail exécute dans son body le DML suivant:

```
Update Chercheur  c  SET c.courriel = self.courriel  Where c.noC = self.noC;
```

AjouterMail() sans droit de lecture

La variable v_o est de type chercheur_t : -- applicatif PL/SQL

```
Declare
v_o chercheur_t := New chercheur_t ( null, null, ,,,,,) – création objet de travail
noCh int := 100;
courriel varchar2(50) := 'abc@utm.com';
Begin
v_o.AjouteMail 2(noCh, courriel);
End;
```

La méthode dit dans ce cas effectuer l'initialisation de la variable objet v_o:

```
Select Value(c) into v_oc From Chercheur c Where c.noC = noCh;
```

Ensuite modifier l'objet lu:

```
v_oc.courriel := courriel ;
```

Et finalement faire un insert pour le rendre persistant:

```
insert into Chercheur (v_oc);
```

Création d'un type avec son interface

```
Create or Replace Type personne_t as Object (nasP int, dateNaissP Date,
villeP varchar2(50),
    Member Function GetAge Return Number,
     Member Procedure PutDateNaiss ,
    Member Procedure ajoutPersonneC)
   /
```

L'âge sera calculé par la méthode GetAge sur appel par un objet ou une variable de type Personne.

```
Create Table PersonneC of personne_t;
```

6

Ajout/ suppression des méthodes dans le type

Alter type personne_t DROP Member Procedure putDateNaiss Cascade;

Cascade sous-tend que la modification au type sera propagée aux sous-types d'une hiérarchie d'héritage.

Ajout de la signature putDateNaiss:

obligatoire

Alter type personne_t ADD Member Procedure putDateNaiss Cascade;

7

Implémentation du body

/*Respecter l'ordre des méthodes de la spécification et dans la création du type */

```
Create or Replace Type Body personne_t as
   Member Function GetAge return Number IS    -- signature du body
     v_ageP number(4,2) ;
     v_naissP Date;
     Begin
     Select p.naissP into v_naissP From Personne p
        where p.nasP = self.nasP ; -- un seul objet retourné avec l'usage de la clé
     v_ageP := (Sysdate - v_naissP) /365 ;
     Return v_ageP;
     End GetAge;
```

8

Implémentation du body

```
Member Procedure PutDateNaiss  IS
     BEGIN
     Update PersonneC  p set p.dateNaissP = self.dateNaissP
                    where p.nasP = self.nasP ;
     End putDateNaiss;

Member Procedure ajoutPersonneC  IS
     BEGIN
     Insert into PersonneC values (self);          éventuellement   /* NULL;  */
     /* Insert into PersonneC values(personneC_t(null, null, ....)) */
     End  ajoutPersonneC;
     End;
```

Attention: Lors de la création de l'interface d'une classe, tous les *bodys* sont ajoutés ou aucun ne l'est !!!

** self est un paramètre par défaut avec toute méthode Member

Appel de la méthode

```
Select   p.getAge()
 From     Personne p   --alias  nécessaire pour représenter l'objet
 Where  p.nasP = 2345;
```

10

Sortes de méthodes

•Méthode MEMBER

Appel par une variable objet instanciée au préalable

Function: nomFonction (…) return type

IS ….code fonction RETURN var ou self

END;

Procédure: nomProc (…)

IS …..code de la procéd

END;

•Constructeur d'objet

•Méthode STATIC

11

Méthode STATIC

•Invoquée par le nom du type et non par un objet

•Exemple: nom_type.nom_méthode(liste param)

•Utilisation interdite du SELF

12

Méthode STATIC

```
Create or Replace Type labo_t as Object (noL int, nomL varchar2(50), nbBrevet int,
STATIC Function changeNomL (nouvNomL IN varchar2, ancNomL IN
        varchar2) RETURN number )
/
Create table Labo of labo_t;
```

La méthode changeNomL a une signature comprenant 2 paramètres IN et un paramètre de retour pour la fonction. (*Notez l'absence de la longueur du type*)

Ajout d'un objet:

```
Insert into Labo values ( labo_t(50, 'nom1', 123)) ;  -- constructeur implicite
```

13

Appel d'une Méthode STATIC dans un bloc PL/SQL

```
Create or Replace type body labo_t as
    STATIC Function changeNomL (nouvNomL IN varchar2, ancNomL IN varchar2) RETURN
      number  IS
    Begin
    Update Labo Set nomL = nouvNomL Where nomL = ancNomL;
    Return 1;    -- succès
    End;
    End;
    /
```

Une méthode STATIC est appelée comme une fonction classique sans exiger la création ou la recherche d'un objet pour l'appeler:

```
    Declare   -applicatif
    res int;
    Begin
    res := labo_t.changeNomL('nom2', 'nom1');  -- retour 1 si bien exécutée
    If res = 1 Then COMMIT;  Else ROLLBACK;
    End if;
    End;
   /
```

nomDuType.nomDeLaFonction

Méthode constructeur

- Méthode pour les objets non persistants
- Initialisation des objets instanciés
- Deux types de constructeurs:
  - Par défaut: ne nécessite pas de définition dans la classe
  - Explicite: concrétisé par une méthode CONSTRUCTOR

15

Méthode constructeur

- Le type de retour d'une méthode Constructor est celui de l'objet d'appel: SELF
- Si aucun constructeur explicite n'est défini dans un type, alors la création des objets est faite obligatoirement par le constructeur implicite soit le nom du type:

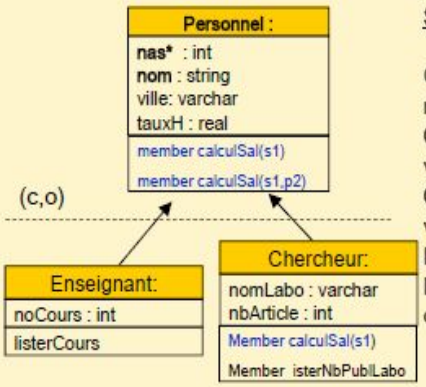
  employe-_t(345, 'Mohamed', 2)

  pour un objet de Employe (nas, poste, expProf)

- Si un constructeur explicite est défini, le New y fera appel en utilisant sa signature
- Des valeurs par défaut peuvent être définies sinon elles sont des nulls

16

Définition de 2 constructeurs



Surcharge du constructeur du type Personnel_t

Create type personnel_t as Object (nas int, nom varchar(50), ville varchar(50), tauxH real, **Constructor Function personnel_t**(nas int, nom varchar, ville varchar) return SELF as Result, **Constructor Function personnel_t** (nas int, nom varchar) return SELF as Result, Member procedure calculSal(s1 out real), Member Procedure calculSal(s1 out real, p2 out real)) /

*Il y a un taux horaire conventionné à 25$ si la ville n'est pas fournie ; sinon le taux horaire est 40$.
** La procédure calculSal a 1 ou 2 paramètres fournissant le salaire basé sur le taux horaire par défaut (25$) et effectue le calcul des charges

Body des constructeurs surchargés

```
Create or Replace type body personnel_t as
Constructor Function  personnel_t(nas int, nom varchar, ville varchar) return SELF as Result IS
Begin
Self.nas  := nas;
Self.nom := nom;
Self.ville := ville;
Self.tauxH := 40.00;
Return;
End;
Constructor Function  personnel_t(nas int, nom varchar) return SELF as Result IS
Begin
Self.nas  := nas;
Self.nom := nom;
Self.ville := 'inconnue' ;
Self.tauxH := 25.00;
Return;
End;
```

Body des constructeurs surchargés

```
Member procedure calculSal (s1 out  real)  IS
Begin
  Null;
End;
Member Procedure calculSal (s1 out real, p2 out real )  IS
Begin
  Null;    -- Stub
End;
End;
/
```

19

Ajout d'objets

Insert into Personnel values (personnel_t(4535, 'Paul'));
⇒ l'objet créé est : [453, 'Paul', 'inconnue ' , 40.00]

Insert into Personnel values (personnel_t(776, 'Laurie-Anne', 'Montréal'));
⇒ l'objet créé est : [776, 'Laurie-Anne', 'Montréal', 25.00]

⇒ Insert into Personnel values (personnel_t(120, 'Delphine', 'Québec', 55.00));

⇒ Select * from Personnel;

NAS	NOM	VILLE	TAUXH
453	Paul	inconnue	40
776	Laurie-Anne	Montréal	25
120	Laurie-Anne	Québec	55