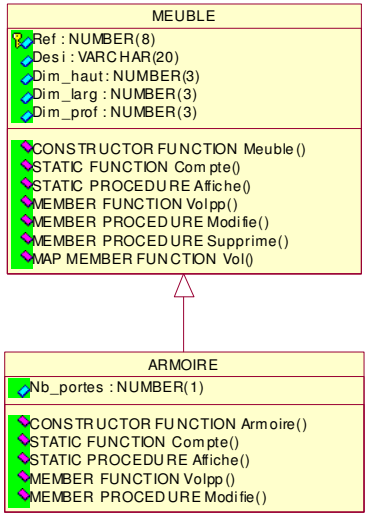
Département Informatique et Statistique, Faculté de SEG, Université Lumière Lyon 2
M2 Informatique spécialité IDS-IIDEE – Année 2011-2012

Bases de données objets – TD n° 4

J. Darmont (*http://eric.univ-lyon2.fr/~jdarmont/*), 13/03/12

Base de données

1. Créer les types T_Meuble et T_Armoire (**attributs uniquement**) correspondant au diagramme de classes UML de la mini-base de données ci-dessous.



2. Définir une table nommée Meuble d'objets de type T_Meuble.

Constructeurs

1. Dans un bloc PL/SQL anonyme, déclarer trois variables de type T_Meuble (m1-m3) et une de type T_Armoire (a1). Dans le code, initialiser m1 à l'aide du constructeur par défaut du type T_Meuble.

```
T_MEUBLE(0, 'Coffre', 100, 100, 50)
```

2. Surcharger le constructeur par défaut de T_Meuble à l'aide d'une fonction prenant en paramètres uniquement la désignation et les dimensions du meuble. Cette fonction doit, en plus des initialisations d'attributs classiques dans un constructeur :

- stocker l'objet meuble créé dans la table Meuble ;
- générer automatiquement la référence du nouveau meuble (10000000 pour la première référence, puis autoincrémentation de 10 en 10).

3. Initialiser m2 et m3 à l'aide de ce constructeur, puis afficher le contenu de la table Meuble à l'aide d'une requête SQL.

```
T_MEUBLE(10000000, 'Meuble téléphone', 110, 50, 30)
T_MEUBLE(10000010, 'Colonne CD', 180, 20, 10)
```

4. Créer un nouveau constructeur pour T_Armoire, similaire à celui de la question 3 pour T_Meuble. Stocker les objets de type T_Armoire dans la table Meuble. Initialiser a1 à l'aide de ce constructeur, puis afficher le contenu de la table Meuble à l'aide d'une requête SQL.

```
T_ARMOIRE(10000020, 'Armoire métal', 190, 130, 70, 2)
```

Méthodes statiques

1. Définir une méthode Compte pour le type T_Meuble, qui renvoie le nombre de meubles dans la table Meuble. Afficher le résultat de l'appel à cette méthode dans le bloc PL/SQL anonyme, pour les types T_Meuble et T_Armoire (pour vérifier que la méthode est bien héritée par T_Armoire).

2. Surcharger la méthode Compte dans le type T_Armoire de manière à ne renvoyer que le nombre d'armoires. Note : comme pour les constructeurs, il est inutile d'employer la clause `OVERRIDING` pour les méthodes statiques. Vérifier le résultat en réexécutant le bloc PL/SQL.

3. Définir une méthode Affiche pour le type T_Meuble qui affiche tous les meubles de la table Meuble au format « référence désignation - Hauteur : xxx, Largeur : xxx, Profondeur : xxx ». Afficher le résultat de l'appel à cette méthode dans le bloc PL/SQL anonyme, pour les types T_Meuble et T_Armoire.

4. Surcharger la méthode Affiche dans le type T_Armoire de manière à afficher en plus l'attribut Nb_portes. Vérifier le résultat en réexécutant le bloc PL/SQL.

Méthodes membres

1. Définir une méthode Volpp pour le type T_Meuble, permettant de renvoyer le volume de l'objet meuble appelant (hauteur x largeur x profondeur). Dans le bloc PL/SQL anonyme, afficher le résultat de cette méthode appliquée à m2 et a1.

2. Surcharger la méthode Volpp dans le type T_Armoire afin de renvoyer le volume de l'objet armoire appelant divisé par le nombre de portes de l'armoire. Dans le bloc PL/SQL anonyme, afficher le résultat de cette méthode appliquée à a1.

3. Définir une méthode Modifie pour le type T_Meuble, permettant de modifier les valeurs de tous les attributs de l'objet meuble appelant. Dans le bloc PL/SQL anonyme, appliquer cette méthode à m2. Afficher ensuite la valeur de l'attribut Dim_haut de m2 (pour vérification), puis afficher les meubles.

```
T_MEUBLE('10000001', 'Meuble téléphone', 115, 45, 30)
```

4. Définir une méthode Modifie pour le type T_Armoire, similaire à la méthode Modifie du type T_Meuble. Dans le bloc PL/SQL anonyme, appliquer cette méthode à a1, puis afficher les armoires.

```
T_ARMOIRE('50000000', 'Armoire bois', 210, 150, 75, 3)
```

5. Définir une méthode Supprime pour le type T_Meuble, permettant de supprimer l'objet meuble appelant. Est-il nécessaire de créer la même méthode spécifiquement pour T_Armoire ? Dans le bloc PL/SQL anonyme, appliquer cette méthode à m3 et a1. Afficher ensuite la valeur de l'attribut Dim_haut de m3 (pour vérification), puis afficher les meubles.

Évolutions de schéma

1. Ajouter au type T_Meuble une méthode de comparaison d'objets MAP nommée Vol, qui effectue le même calcul que Volpp. Dans le bloc PL/SQL anonyme (avant la suppression des objets), comparer les objets m2 et m3, puis m2 et a1 (test d'égalité) et afficher si les meubles sont différents (volumes différents) ou susceptibles d'être identiques (volumes égaux).

2. Ajouter au type T_Meuble un attribut Prix réel.

Vues objet

1. Créer une vue objet nommée Vue_Salle_cours basée sur le type T_Salle_cours et permettant d'afficher toutes les caractéristiques des salles de cours de la table Salle du TD n° 2.

2. Afficher la structure et le contenu de la vue objet Vue_Salle_cours.

3. Définir un déclencheur de type INSTEAD OF permettant, lors d'une insertion dans la vue objet Vue_Salle_cours, d'effectuer l'insertion correspondante dans la table source Salle.

4. Insérer un nouvel objet dans la table Salle par l'intermédiaire de la vue objet Vue_Salle_cours. Vérifier le contenu de la vue objet Vue_Salle_cours et de la table Salle.

5. On souhaite maintenant visualiser les réservations des salles de cours sous la forme de collections.

- Créer un nouveau type T_Resa regroupant les attributs Jour, Heure_début et Heure_fin.
- Créer un nouveau type Tab_Resa table d'objets de type T_Resa.
- Rendre le type T_Salle_cours héritable (NOT FINAL).
- Créer un nouveau type T_Salle_cours_resa héritant de T_Salle_cours et contenant un attribut supplémentaire Reservations de type Tab_Resa.

6. Créer une vue objet nommée Vue_Salle_cours2 similaire à Vue_Salle_cours, basée sur le type T_Salle_cours_resa et affichant en plus sous forme de collection les réservations de chaque salle. Afficher la structure et le contenu de la vue objet Vue_Salle_cours2.

Dictionnaire des données objet

1. Afficher, pour chacun des types que vous avez créés, son nom, le nom de sa superclasse, s'il est « héritable », abstrait, ainsi que le nombre de versions qu'a connu ce type.

2. Afficher les caractéristiques de toutes vos tables objet (nom de la table, type de l'OID, type sur lequel est basée la table, indicateur table imbriquée ou non).

3. En une seule requête, afficher la liste des attributs (nom, type, longueur, précision, attribut hérité ou non) et des méthodes (nom, type, « héritable » ou non, surchargeant une méthode de la superclasse ou non, héritée ou non) du type T_Armoire.

Correction

```
-- Ménage

DROP TABLE Meuble;
DROP TYPE T_Armoire;
DROP TYPE T_Meuble;

-- Interfaces des types

CREATE TYPE T_Meuble AUTHID CURRENT_USER AS OBJECT(
     -- Attributs
     Ref NUMBER(8),
     Desi VARCHAR(20),
     Dim_haut NUMBER(3),
     Dim_larg NUMBER(3),
     Dim_prof NUMBER(3),
     -- Méthodes
     CONSTRUCTOR FUNCTION T_Meuble(   pdesi VARCHAR,
                                      phaut NUMBER,
                                      plarg NUMBER,
                                      pprof NUMBER ) RETURN SELF AS RESULT,
     STATIC FUNCTION Compte RETURN INTEGER,
     STATIC PROCEDURE Affiche,
     NOT FINAL MEMBER FUNCTION Volpp RETURN NUMBER,
     MEMBER PROCEDURE Modifie(    nref NUMBER,
                                  ndesi VARCHAR,
                                  nhaut NUMBER,
                                  nlarg NUMBER,
                                  nprof NUMBER ),
     MEMBER PROCEDURE Supprime
) NOT FINAL
/
show errors

CREATE TYPE T_Armoire UNDER T_Meuble(
     -- Attributs
     Nb_portes NUMBER(1),
     -- Méthodes
     CONSTRUCTOR FUNCTION T_Armoire(  pdesi VARCHAR,
                                      phaut NUMBER,
                                      plarg NUMBER,
                                      pprof NUMBER,
                                      pnbp NUMBER ) RETURN SELF AS RESULT,
     STATIC FUNCTION Compte RETURN INTEGER,
     STATIC PROCEDURE Affiche,
     OVERRIDING MEMBER FUNCTION Volpp RETURN NUMBER,
     MEMBER PROCEDURE Modifie(    nref NUMBER,
                                  ndesi VARCHAR,
                                  nhaut NUMBER,
                                  nlarg NUMBER,
                                  nprof NUMBER,
                                  nnbp NUMBER )
)
/
show errors

-- Altérations de type (1)

ALTER TYPE T_Meuble ADD MAP MEMBER FUNCTION Vol RETURN NUMBER CASCADE;
```

```
-- Table des instances

CREATE TABLE Meuble OF T_Meuble(CONSTRAINT meuble_pk PRIMARY KEY(Ref));

-- Corps des types

CREATE OR REPLACE TYPE BODY T_Meuble AS

     CONSTRUCTOR FUNCTION T_Meuble(   pdesi VARCHAR,
                                phaut NUMBER,
                                plarg NUMBER,
                                pprof NUMBER ) RETURN SELF AS RESULT IS
          n INTEGER;
     BEGIN
          -- Calcul identifiant
          SELECT COUNT(*) INTO n FROM Meuble;
          IF n > 0 THEN
               SELECT MAX(Ref)+10 INTO SELF.Ref FROM Meuble;
          ELSE
               SELF.Ref := 10000000;
          END IF;
          -- Autres attributs
          SELF.Desi := pdesi;
          SELF.Dim_haut := phaut;
          SELF.Dim_larg := plarg;
          SELF.Dim_prof := pprof;
          -- Stockage
          INSERT INTO Meuble VALUES (SELF);
          RETURN;
     END;

     STATIC FUNCTION Compte RETURN INTEGER IS
          n INTEGER;
     BEGIN
          SELECT COUNT(*) INTO n FROM Meuble;
          RETURN n;
     END;

     STATIC PROCEDURE Affiche IS
          CURSOR cmeubles IS SELECT * FROM Meuble;
          m T_Meuble;
     BEGIN
          FOR m IN cmeubles LOOP
               DBMS_OUTPUT.PUT_LINE(m.Ref || ' ' || m.Desi || ' - Hauteur : ' ||
                    m.Dim_haut || ', Largeur : ' || m.Dim_larg || ',
                    Profondeur : ' || m.Dim_prof);
          END LOOP;
     END;

     NOT FINAL MEMBER FUNCTION Volpp RETURN NUMBER IS
     BEGIN
          RETURN SELF.Dim_haut * SELF.Dim_larg * SELF.Dim_prof;
     END;
```



```
     MEMBER PROCEDURE Modifie(    nref NUMBER,
                                  ndesi VARCHAR,
                                  nhaut NUMBER,
                                  nlarg NUMBER,
                                  nprof NUMBER ) IS
     BEGIN
          UPDATE Meuble SET  Ref = nref,
                             Desi = ndesi,
                             Dim_haut = nhaut,
                             Dim_larg = nlarg,
                             Dim_prof = nprof
          WHERE Ref = SELF.Ref;
          SELF.Ref := nref;
          SELF.Desi := ndesi;
          SELF.Dim_haut := nhaut;
          SELF.Dim_larg := nlarg;
          SELF.Dim_prof := nprof;
     END;

     MEMBER PROCEDURE Supprime IS
     BEGIN
          DELETE FROM Meuble WHERE Ref = SELF.Ref;
          SELF := NULL;
     END;

     MAP MEMBER FUNCTION Vol RETURN NUMBER IS
     BEGIN
          RETURN SELF.Dim_haut * SELF.Dim_larg * SELF.Dim_prof;
     END;

END;
/
show errors

CREATE TYPE BODY T_Armoire AS

     CONSTRUCTOR FUNCTION T_Armoire(  pdesi VARCHAR,
                                      phaut NUMBER,
                                      plarg NUMBER,
                                      pprof NUMBER,
                                      pnbp NUMBER ) RETURN SELF AS RESULT IS
          n INTEGER;
     BEGIN
          -- Calcul identifiant
          SELECT COUNT(*) INTO n FROM Meuble;
          IF n > 0 THEN
               SELECT MAX(Ref)+10 INTO SELF.Ref FROM Meuble;
          ELSE
               SELF.Ref := 10000000;
          END IF;
          -- Autres attributs
          SELF.Desi := pdesi;
          SELF.Dim_haut := phaut;
          SELF.Dim_larg := plarg;
          SELF.Dim_prof := pprof;
          SELF.Nb_portes := pnbp;
          -- Stockage
          INSERT INTO Meuble VALUES (SELF);
          RETURN;
     END;
```

```
     STATIC FUNCTION Compte RETURN INTEGER IS
          n INTEGER;
     BEGIN
          SELECT COUNT(*) INTO n FROM Meuble
               WHERE OBJECT_VALUE IS OF(T_Armoire);
          RETURN n;
     END;

     STATIC PROCEDURE Affiche IS
          CURSOR carmoires IS SELECT m.Ref, m.Desi, m.Dim_haut, m.Dim_larg,
               m.Dim_prof, TREAT(OBJECT_VALUE AS T_Armoire).Nb_portes Nbp
               FROM Meuble m
               WHERE OBJECT_VALUE IS OF(T_Armoire);
          a T_Armoire;
     BEGIN
          FOR a IN carmoires LOOP
               DBMS_OUTPUT.PUT_LINE(a.Ref || ' ' || a.Desi || ' - Hauteur : ' ||
                    a.Dim_haut || ', Largeur : ' || a.Dim_larg || ',
                    Profondeur : ' || a.Dim_prof || ', Nombre de portes : ' ||
                    a.Nbp);
          END LOOP;
     END;

     OVERRIDING MEMBER FUNCTION Volpp RETURN NUMBER IS
     BEGIN
          RETURN SELF.Dim_haut * SELF.Dim_Larg * SELF.Dim_prof / SELF.Nb_portes;
     END;

     MEMBER PROCEDURE Modifie(    nref NUMBER,
                                  ndesi VARCHAR,
                                  nhaut NUMBER,
                                  nlarg NUMBER,
                                  nprof NUMBER,
                                  nnbp NUMBER) IS
     BEGIN
          SELF.Ref := nref;
          SELF.Desi := ndesi;
          SELF.Dim_haut := nhaut;
          SELF.Dim_larg := nlarg;
          SELF.Dim_prof := nprof;
          SELF.Nb_portes := nnbp;
          UPDATE Meuble SET OBJECT_VALUE = SELF
               WHERE Ref = SELF.Ref;

     END;

END;
/
show errors
```

-- Invocation des méthodes

```
set serveroutput on
set linesize 200

DECLARE
     m1 T_Meuble;
     m2 T_Meuble;
     m3 T_Meuble;
     a1 T_Armoire;
```



```
BEGIN
     -- Initialisations
     m1 := NEW T_Meuble(0, 'Coffre', 100, 100, 50);
     m2 := NEW T_Meuble('Meuble téléphone', 110, 50, 30);
     m3 := NEW T_Meuble('Colonne CD', 180, 20, 10);
     a1 := NEW T_Armoire('Armoire métal', 190, 130, 70, 2);
     -- Comptes
     DBMS_OUTPUT.PUT_LINE('Nombre de meubles : ' || T_Meuble.Compte);
     DBMS_OUTPUT.PUT_LINE('Nombre d''armoires : ' || T_Armoire.Compte);
     -- Affichages
     T_Meuble.Affiche;
     T_Armoire.Affiche;
     -- Volumes
     DBMS_OUTPUT.PUT_LINE('m2.Volpp = ' || m2.Volpp);
     DBMS_OUTPUT.PUT_LINE('a1.Volpp = ' || a1.Volpp);
     -- Modifications
     m2.Modifie(10000001, 'Meuble téléphone', 115, 45, 30);
     DBMS_OUTPUT.PUT_LINE(m2.Dim_haut);
     T_Meuble.Affiche;
     a1.Modifie(50000000, 'Armoire bois', 210, 150, 75, 3);
     T_Armoire.Affiche;
     -- Comparaisons
     IF m2 = m3 THEN
          DBMS_OUTPUT.PUT_LINE('m2 et m3 sont les mêmes meubles ?');
     ELSE
          DBMS_OUTPUT.PUT_LINE('m2 et m3 ne sont pas les mêmes meubles.');
     END IF;
     IF m2 = a1 THEN
          DBMS_OUTPUT.PUT_LINE('m2 et a1 sont les mêmes meubles ?');
     ELSE
          DBMS_OUTPUT.PUT_LINE('m2 et a1 ne sont pas les mêmes meubles.');
     END IF;
     -- Suppressions
     m3.Supprime;
     a1.Supprime;
     DBMS_OUTPUT.PUT_LINE(m3.Dim_haut);
     T_Meuble.Affiche;

END;
/
```

-- Altérations de type (2)

```
ALTER TYPE T_Meuble ADD ATTRIBUTE (Prix NUMBER(8,2))
CASCADE INCLUDING TABLE DATA;
```

-- Affichage table Meuble

```
SELECT VALUE(m) FROM Meuble m;
```

-- Vues objet

```
CREATE VIEW Vue_Salle_cours OF T_Salle_cours
WITH OBJECT IDENTIFIER(Numero) AS
SELECT    Numero,
          Videoprojecteur,
          TREAT(OBJECT_VALUE AS T_Salle_cours).Capacite,
          TREAT(OBJECT_VALUE AS T_Salle_cours).Retroprojecteur,
          TREAT(OBJECT_VALUE AS T_Salle_cours).Micro
FROM Salle
WHERE OBJECT_VALUE IS OF(T_Salle_cours);

DESC Vue_Salle_cours
```

```
SELECT * FROM Vue_Salle_cours;

CREATE OR REPLACE TRIGGER Insertion_Salle_cours
INSTEAD OF INSERT ON Vue_Salle_cours FOR EACH ROW
BEGIN
     INSERT INTO Salle VALUES(:NEW.OBJECT_VALUE);
END;
/

INSERT INTO Vue_Salle_cours VALUES ('05R', 'N', 50, 'N', 'N');

SELECT * FROM Vue_Salle_cours;

SELECT VALUE(s) FROM Salle s;

CREATE OR REPLACE TYPE T_Resa AS OBJECT(
     Jour VARCHAR(10),
     Heure_debut NUMBER(4,1),
     Heure_fin NUMBER(4,1))
/

CREATE OR REPLACE TYPE Tab_Resa AS TABLE OF T_Resa
/

ALTER TYPE T_Salle_cours NOT FINAL
CASCADE CONVERT TO SUBSTITUTABLE;

CREATE OR REPLACE TYPE T_Salle_cours_resa UNDER T_Salle_cours(
     Reservations Tab_Resa)
/

CREATE VIEW Vue_Salle_cours2 OF T_Salle_cours_resa
WITH OBJECT IDENTIFIER(Numero) AS
SELECT s.Numero,
     s.Videoprojecteur,
     TREAT(OBJECT_VALUE AS T_Salle_cours).Capacite,
     TREAT(OBJECT_VALUE AS T_Salle_cours).Retroprojecteur,
     TREAT(OBJECT_VALUE AS T_Salle_cours).Micro,
     CAST(MULTISET(SELECT Jour, Heure_debut, Heure_fin
          FROM Planning p
          WHERE p.Ref_salle.Numero = s.Numero) AS Tab_Resa) AS Reservations
FROM Salle s
WHERE OBJECT_VALUE IS OF(T_Salle_cours);

DESC Vue_Salle_cours2

SELECT * FROM Vue_Salle_cours2;
```

-- Dictionnaire des données objet

```
SELECT t.TYPE_NAME, SUPERTYPE_NAME, FINAL, INSTANTIABLE,
     COUNT(DISTINCT VERSION#)
FROM USER_TYPES t, USER_TYPE_VERSIONS v
WHERE t.TYPE_NAME = v.TYPE_NAME
GROUP BY t.TYPE_NAME, SUPERTYPE_NAME, FINAL, INSTANTIABLE;

SELECT TABLE_NAME, OBJECT_ID_TYPE, TABLE_TYPE, NESTED
FROM USER_OBJECT_TABLES;
```



```
SELECT ATTR_NAME || ' ' || ATTR_TYPE_NAME || ' ' || LENGTH || ' ' || PRECISION
     || ' ' || INHERITED
FROM USER_TYPE_ATTRS
WHERE TYPE_NAME='ARMOIRE'
UNION
SELECT METHOD_NAME || ' ' || METHOD_TYPE || ' ' || FINAL || ' ' || OVERRIDING
     || ' ' ||  INHERITED
FROM USER_TYPE_METHODS
WHERE TYPE_NAME='ARMOIRE';
```