La programmation Objet

L'approche orientée objet / Méthode de programmation

Méthode de conception et de réalisation de logiciels

Abstraction des données

manipuler des objets sans connaître leur représentation physique, permettre aussi une uniformité dans la manipulation (extension du langage pour l'ajout de nouveaux types de données).

Modularité

modification d'un module avec un minimum de répercussion sur les autres modules

Réutilisabilité

ne pas refaire le monde constamment ; création et utilisation de bibliothèques d'objets génériques, aisément extensibles

Lisibilité, clarté des programmes

contrôle de la complexité et de la maintenance des programmes

Les concepts Objet

Notion 1 : OBJET

Définition

C'est un élément autonome de l'univers du discours, capable d'être identifé et caractérisé : O(id, valeur). De part son autonomie, il possède un comportement qui lui est propre (liste d'opérations qu'il peut réaliser)

- ### Identifiant d'un objet

  C'est le moyen de repérer un objet de façon intrinsèque indépendamment de son lieu (adresse) et de ses caractéristiques du moment

- ### Caractérisation d'un objet

  C'est l'attribution d'un type (domaine de base ou complexe) et l'instanciation de ce type(assignation d'une valeur)

- ### Méthodes

Notion 2 : METHODE

Définition : programme qui décrit le comportement d'un objet

Spécification

- signature : nom de la méthode + liste des paramètres

- corps : ensemble d'instructions (code) définissant sa sémantique

Exemple

***Objet* : "Jean"**

***Identification* : id = @1**

***Caractérisation* : type Personne (nom string, prenom string, age integer)**

=> instance de type : (Fontaine, Jean, 40)

***Méthodes* : { Chanter(); Manger(nourriture); Anniversaire(); GetAge(); }**

Notion 3 : CLASSE D'OBJETS

Définition : ensemble d'objets ayant tous le même type et le même comportement

Exemple

La classe Personne regroupe toutes les personnes qui comme "Jean" sont de type Personne, et qui *chantent*, qui *mangent*, qui *ont un anniversaire* et qui *rendent leur âge* lorsqu'on leur demande.

```
Class Personne  {
            nom, prenom  string;
            age integer;

            Chanter();
            Manger(nourriture);
           …  };
```

Notion 4 : ENCAPSULATION

Définition : C'est la protection des propriétés (attributs et méthodes) d'un objet vis à vis de l'extérieur. Implicitement tout est protégé (privé) sauf lorsqu'on déclare explicitement le droit d'accès (public). Tous les attributs et méthodes déclarées publiques constituent l'interface de l' objet (les services qu'il peut rendre).

```
Class Personne  {
    private :
        nom, prenom  string;
        age integer;
        adresse   record (n° integer, rue string, code integer, ville string);
    public :
            Chanter();
            Manger(nourriture);     …   };
```

NB : Généralement tous les attributs caractérisant un objet sont cachés, seules les méthodes sont publiques (séparer la définition de l'implémentation)

Notion 5 : ENVOI DE MESSAGES

Définition : Seul moyen de communication avec un objet ou entre objets. C'est l'invocation d'une méthode sur un objet spécifique.

Spécification

- **receveur : objet cible (identifiant)**
- **message : nom de la méthode + liste des paramètres**

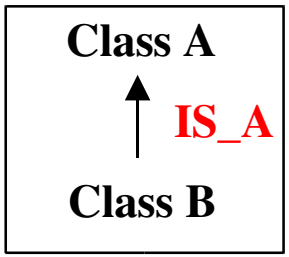
Exemple

Soit l'objet Jean = O(@1, (Fontaine, Jean, 40), {chante();})

```
@1.chante();      // accès à la méthode chante de l'objet Jean
@1->chante();     // invocation de la méthode chante sur l'identifiant représentant
                  // Jean
@1.age;           // interdit : il faut respecter le principe d'encapsulation
```

Notion 6 : HERITAGE

Définition : Il représente un lien sémantique de type IS_A. Il sert à modéliser le concept de généralisation ou de spécialisation entre classes. Il peut être simple ou multiple.



Généralisation : partager (factoriser) un comportement commun à plusieurs classes, même s'il n'est pas possible de créer des instances de la classe générique.

Spécialisation : spécialiser le comportement des objets d'une classe, en rajoutant de nouvelles propriétés (attributs &méthodes), ou en redéfinissant une propriété déjà existante.

EXEMPLE

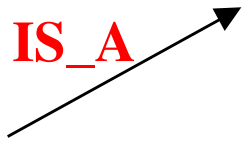
Classe Forme

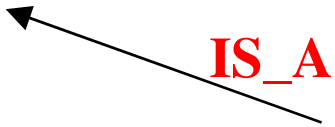
attributs : couleur;

méthodes : Colorier (c)

couleur =c ;

Déplacer (dx,dy);





Classe Carré

attributs : x1, y1, x2, y2 ;

méthodes :

Déplacer (dx,dy)

x1= x1 + dx; x2= x2 + dx;

y1= y1 + dy; y2= y2 + dy;

Afficher();

Classe Cercle

attributs : x, y, rayon ;

méthodes :

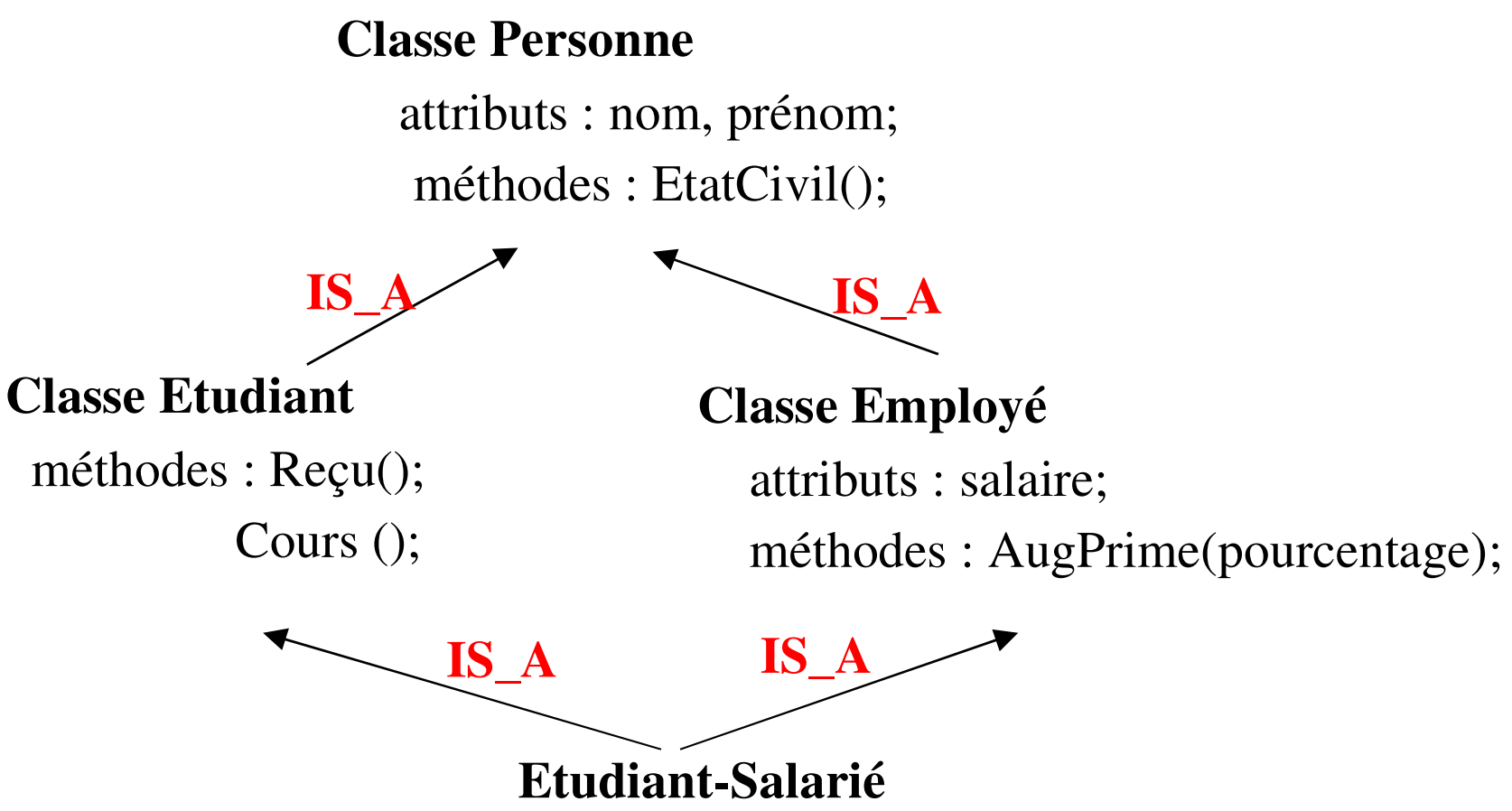
Déplacer (dx,dy)

x = x + dx;

y = y + dy;

Afficher();

HERITAGE MULTIPLE



Notion 7 : LA SURCHARGE

Surcharge : Un même nom de fonction peut désigner plusieurs fonctions différentes, si chacune a une liste de paramètres distincte de celles des autres.

int max (int, int);

double max(double, double);

int max(int, int, int);

Invocation : En cas d'ambiguïté, un algorithme de détermination de la meilleure solution est appliqué. Il met en jeu les conversions standards et définies entre types.

Notion 8 : LE POLYMORPHISME

- **Les langages traditionnels tels que Pascal sont fondés sur le principe que les valeurs du langage ont un type unique qui peut être déterminé à la compilation du programme. De tels langages sont dits *monomorphes***

- **Les langages *polymorphes* sont des langages dans lesquels une valeur, et notamment une fonction, peut appartenir à plusieurs types à la fois.**

EXEMPLE

Classe Forme

attributs : couleur;

méthodes : Colorier (c)

couleur =c ;

Déplacer (dx,dy);

IS_A

IS_A

Classe Carré

attributs : x1, y1, x2, y2 ;

méthodes :

Déplacer (dx,dy)

x1= x1 + dx; x2= x2 + dx;

y1= y1 + dy; y2= y2 + dy;

Afficher();

Classe Cercle

attributs : x, y, rayon ;

méthodes :

Déplacer (dx,dy)

x = x + dx;

y = y + dy;

Afficher();

POLYMORPHISME ...

Exemple

Rectangle rec;
Carré car;
Rectangle* ref = **&**car;
cout **<<** car.Afficher();
cout **<<** rec.Afficher();
cout **<<** ref->Afficher();

Le polymorphisme introduit la notion de typage *statique* et *dynamique*

car est *statiquement* et *dynamiquement* un carré
rec est *statiquement* et *dynamiquement* un rectangle
ref est *statiquement* un rectangle, *mais dynamiquement un carré* =>
Quelle méthode faut-il invoquer?

Notion 9 : LIAISON TARDIVE DE METHODES

Définition (late binding) : C'est privilégier le typage dynamique. Le choix de la méthode à invoquer est réalisé à l'exécution du programme. Cela ne peut se faire au moment de la compilation.

Exemple :

```
Carré car;
Rectangle* ref = &car;
cout << car.Afficher();       // appel de la méthode Carre. Afficher
cout << ref->Afficher();     // appel de la méthode Carre. Afficher
```

Problèmes des langages compilés ayant un typage fort : le type de l'objet est perdu après sa conversion vis à vis des messages. La détermination de la méthode à appeler est entièrement statique.

Solution: toute méthode doit être une méthode connue au niveau du type statique du receveur dans la hiérarchie, mise en place d'un mécanisme de fonctions virtuelles

Notion 10 : LA GENERICITE

Définition : C'est une forme de polymorphisme paramétrique. Il permet de prendre en compte différents types dans la même spécification.

Exemple :

L'objet Pile_Char { empiler(char); char depiler(); …}

L'objet Pile_Int { empiler(int); int depiler(); …}

=> Pile[T] {empiler(T); T depiler(); …} *// T est un type paramétré*

NB : la généricité est un mécanisme permettant de mettre en facteur la structure et le comportement d'objets génériques (pile, liste, ensemble,…). Il permet une forme "dénaturée" d'héritage. Cependant, il ne peut pas être supporté à l'aide de ce dernier.

BILAN

Les modèles orientés objet sont vastes et variés. Les concepts mis en oeuvre dans l'approche objet sont pour la plupart une synthèse des concepts développés en :

- **Programmation**

  Types et Types Abstraits de données

  Notion de modules, de packages

  Ada, Simula, Smalltalk, LISP, …

- **Intelligence Artificielle**

  Modèles de représentation des connaissances (réseaux sémantiques)

  Hiérarchie de classes - Héritage, agrégation-généralisation-spécialisation