## MODÉLISER LES OBJETS GRÂCE À UML

Le sigle « UML » signifie ***U****nified* ***M****odeling****L****anguage*, que l'on peut traduire par « langage de modélisation unifié ». Il ne s'agit pas d'un langage de programmation, mais plutôt d'une méthode de modélisation. La méthode Merise, par exemple, en est une autre.

En fait, lorsque vous programmez en orienté objet, il vous sera sans doute utile de pouvoir schématiser vos classes, leur hiérarchie, leurs dépendances, leur architecture, etc. L'idée est de pouvoir, d'un simple coup d'œil, vous représenter le fonctionnement de votre logiciel.

Présentation d'UML

Avec UML, vous pouvez modéliser toutes les étapes du développement d'une application informatique, de sa conception à la mise en route, grâce à des diagrammes. Il est vrai que certains de ces diagrammes sont plus adaptés pour les informaticiens, mais il en existe qui permettent de voir comment interagit l'application avec son contexte de fonctionnement… Et dans ce genre de cas, il est indispensable de bien connaître l'entreprise pour laquelle l'application est prévue. On recourt donc à un mode de communication compréhensible par tous : UML.

Il existe bien sûr des outils de modélisation pour créer de tels diagrammes:

* boUML,
* ObjectAid UML
* Together,
* Poseidon,
* Pyut
* etc.

Avec ces outils, vous pouvez réaliser les différents diagrammes qu'UML vous propose :

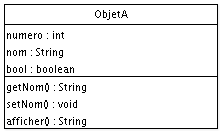
* le diagramme de classes ; c'est de celui-là que nous allons nous servir. Il permet de modéliser des classes ainsi que les interactions entre elles ;
* les diagrammes de séquences, eux, permettent de visualiser le déroulement d'une application dans un contexte donné ;
* et d'autres encore…

La figure suivante représente un exemple de diagramme de classes.



Modéliser ses objets

À présent, nous allons apprendre à lire un diagramme de classes. Vous avez deviné qu'une classe est modélisée sous la forme représentée sur la figure suivante.

Classe en UML

Voici une classe nommée ObjetA qui a comme attributs :

* numero de type int ;
* nom de type String ;
* bool de type boolean.

Ses méthodes sont :

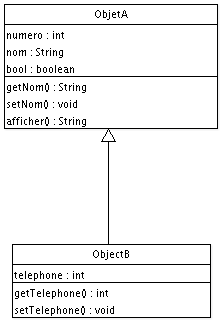
* getNom() qui retourne une chaîne de caractères ;
* setNom() qui ne renvoie rien ;
* afficher() qui renvoie également une chaîne de caractères.

Modéliser les liens entre les objets

Les interactions sont modélisées par des flèches de plusieurs sortes.

**Cas 1**

Sur le diagramme représenté à la figure suivante, vous remarquez un deuxième objet qui dispose, lui aussi, de paramètres. L' ObjetB possède également les attributs et les méthodes de la classe ObjetA. D'après vous, pourquoi ? C'est parce que la flèche qui relie nos deux objets signifie «extends ». En gros, vous pouvez lire ce diagramme comme suit : l'ObjetB *hérite de* l'ObjetA, ou encore ObjetB *est un* ObjetA.



**Exemple**

**class** EtreHumaine2 {

**public** **boolean** sexe;

**public** String nom;

**public** String profession;

**public** **double** sal;

// méthode constructrice

**public** EtreHumaine2(**boolean** s, String n,String p, **double** sl) {

sexe = s;

nom = n;

profession = p;

sal=sl;}

**public** **void** sePresenter() {

String s = (sexe) ? "homme" : "femme";

System.***out***.println("Mon nom est : " +nom);

System.***out***.println("Je suis un(e) " + s);

System.***out***.println("Ma profession : " + profession+"\n");

System.***out***.println("Mon salaire: " + sal+"\n");

}

}

**class** Etudiant **extends** EtreHumaine2

{ // méthode constructrice

**public** Etudiant (**boolean** s, String n,**double** sl) {

**super** (s,n,"etudiant",sl);

}

}

**class** Medecin **extends** EtreHumaine2

{ // méthode constructrice

**public** Medecin (**boolean** s, String n,**double** sl) {

**super** (s,n,"Medecin",sl);

}

}

**class** Test {

**public** **static** **void** main(String[] args) {

EtreHumaine2 moi = **new** EtreHumaine2(**true**,"Kiss", "enseignant",46000);

moi.sePresenter();

Etudiant et1 = **new** Etudiant(**false**,"Claire",12000);

et1.sePresenter();

Etudiant et2 = **new** Etudiant(**true**,"Dupond",20000);

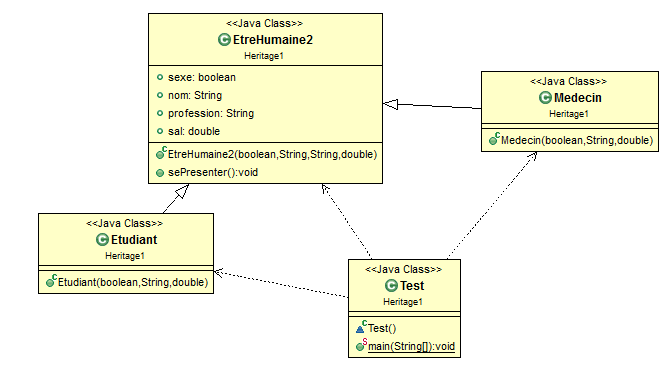
et2.sePresenter();

Medecin me = **new** Medecin(**true**,"Jack",120000);

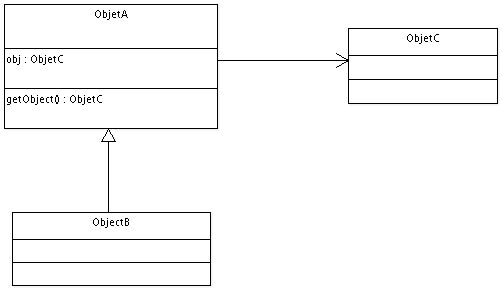
me.sePresenter();

}

}



**Cas 2**



Dans cet exemple simpliste, nous avons toujours notre héritage entre un objet A et un objet B, mais dans ce cas, l'ObjetA (et donc l'ObjetB) possède une variable de classe de type ObjetC, ainsi qu'une méthode dont le type de retour est ObjetC (car la méthode retourne un ObjetC). Vous pouvez lire ce diagramme comme suit : l'ObjetA*a un*ObjetC (donc *une seule* instance d'ObjetC est présente dans ObjetA).

Voici le code Java correspondant à ce diagramme.

**Fichier ObjetA.java**

public class ObjetA{

protected ObjetC obj = new ObjetC();

 public ObjetC getObject(){

  return obj;

 }

}

**Fichier ObjetB.java**

public class ObjetB extends ObjetA{

**Fichier ObjetC.java**

public class ObjetC{

}

**Exemple 2**

**class** EtreHumaine2 {

**public** **boolean** sexe;

**public** String nom;

**public** String profession;

**public** **double** sal;

**protected** Nationnalite On=**new** Nationnalite();

// méthode constructrice

**public** EtreHumaine2(**boolean** s, String n,String p, **double** sl) {

sexe = s;

nom = n;

profession = p;

sal=sl;}

**public** **void** sePresenter() {

String s = (sexe) ? "homme" : "femme";

System.***out***.println("Mon nom est : " +nom);

System.***out***.println("Je suis un(e) " + s);

System.***out***.println("Ma profession : "+ profession+"\n");

System.***out***.println("Mon salaire: "+ sal+"\n");

System.***out***.println("Ma nationnalité: "+ On.Nation+"\n");

System.***out***.println("Ma langue maternelle: "+ On.langue+"\n");

}

**public** Nationnalite getOn()

{

**return** On;

}

}

**class** Etudiant **extends** EtreHumaine2

{

// méthode constructrice

**public** Etudiant (**boolean** s, String n,**double** sl) {

**super** (s,n,"etudiant",sl);

}

}

**class** Medecin **extends** EtreHumaine2

{

// méthode constructrice

**public** Medecin (**boolean** s, String n,**double** sl) {

**super** (s,n,"Medecin",sl);

}

}

**class** Nationnalite {

String Nation,langue;

}

**class** Test {

**public** **static** **void** main(String[] args) {

EtreHumaine2 moi = **new** EtreHumaine2(**true**,"Kiss", "enseignant",46000);

moi.sePresenter();

Etudiant et1 = **new** Etudiant(**false**,"Claire",12000);

et1. getOn().Nation="Canada";

et1. getOn().langue="Francais";

et1.sePresenter();

Etudiant et2 = **new** Etudiant(**true**,"Dupond",20000);

et2. getOn().Nation="USA";

et2. getOn().langue="Anglais";

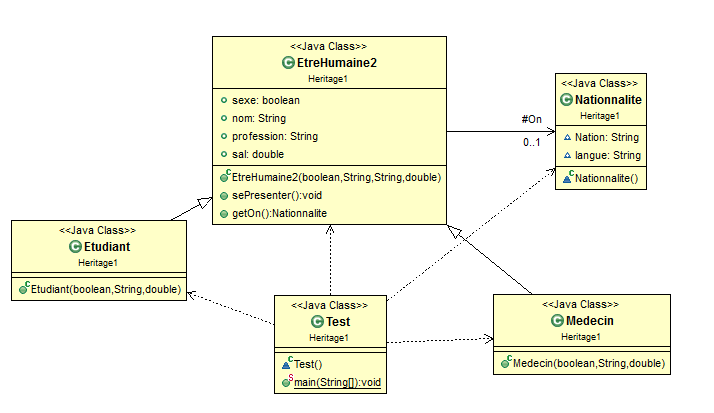
et2.sePresenter();

Medecin me = **new** Medecin(**true**,"Jack",120000);

me.sePresenter();

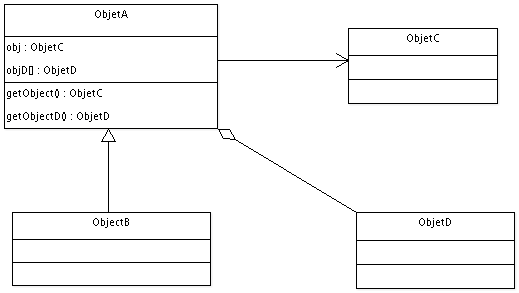
}

}



**Cas 3**

Il reste une dernière flèche que nous pouvons mentionner, car elle ne diffère que légèrement de la première. Un diagramme la mettant en œuvre est représenté sur la figure suivante.



Ce diagramme est identique au précédent, à l'exception de l'ObjetD. Nous devons le lire comme ceci : l'ObjetA *est composé de plusieurs instances d'*ObjetD. Vous pouvez d'ailleurs remarquer que la variable d'instance correspondante est de type tableau…

Voici le code Java correspondant :

**Fichier ObjetA.java**

public class ObjetA{

 protected ObjetC obj = new ObjetC();

protected ObjetD[] objD = new ObjetD[10];

public ObjetC getObject(){

return obj;

   }

  public ObjetD[] getObjectD(){

 return objD;

 }

}

**Fichier ObjetB.java**

public class ObjetB extends ObjetA{

}

**Fichier ObjetC.java**

public class ObjetC{

}

**Fichier ObjetD.java**

public class ObjetD{

}