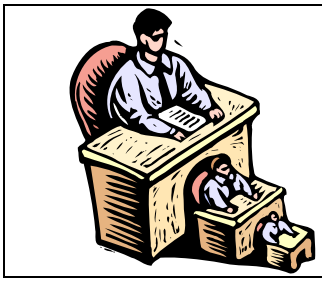
Cours WinDev Numéro 8-4



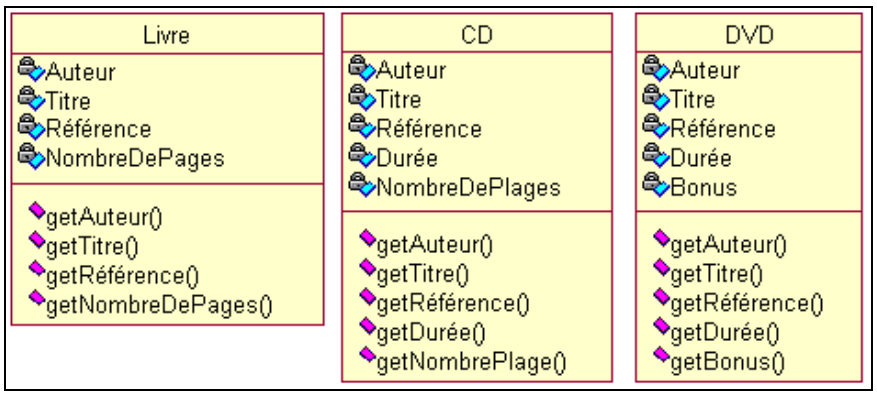
Objectifs : L'héritage (Généralisation – Spécialisation)

Création de classes.
Instanciation dynamique d'objets.
Un peu de théorie.

Pré-requis : L'acquisition parfaite des cours objets précédents.

Généralisation.

Imaginons que nous devions fabriquer un logiciel qui permet de gérer une bibliothèque. Cette bibliothèque comporte plusieurs types de documents ; des livres, des CDs, ou des DVDs. Une première étude nous emmène à mettre en œuvre les classes suivantes.



Nous remarquons que dans les trois types de documents, un certain nombre de caractéristiques se retrouvent systématiquement.

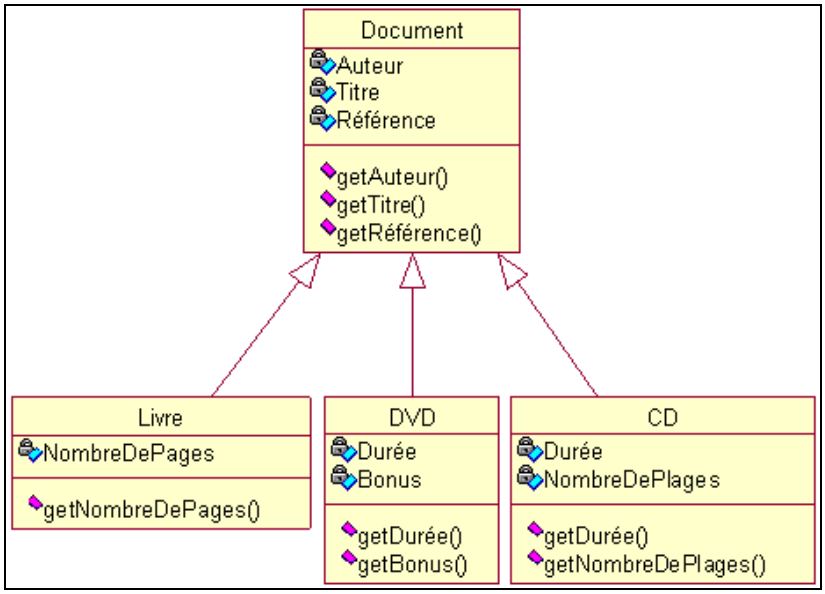
Afin d'éviter la répétition des éléments constituant chacune des classes, il est préférable de factoriser toutes ces caractéristiques communes pour en faire une nouvelle classe plus généraliste.

En effet, nous pouvons dire que, d'une façon générale, et quel que soit le type de document, il comporte au moins un titre, un auteur, etc. il semble aller de soi, que le nom de cette nouvelle classe générale s'appelle justement Document.

Il faut ensuite proposer une relation entre les classes afin de montrer la filiation. Par exemple, il faut bien préciser qu'un Livre est aussi un Document.

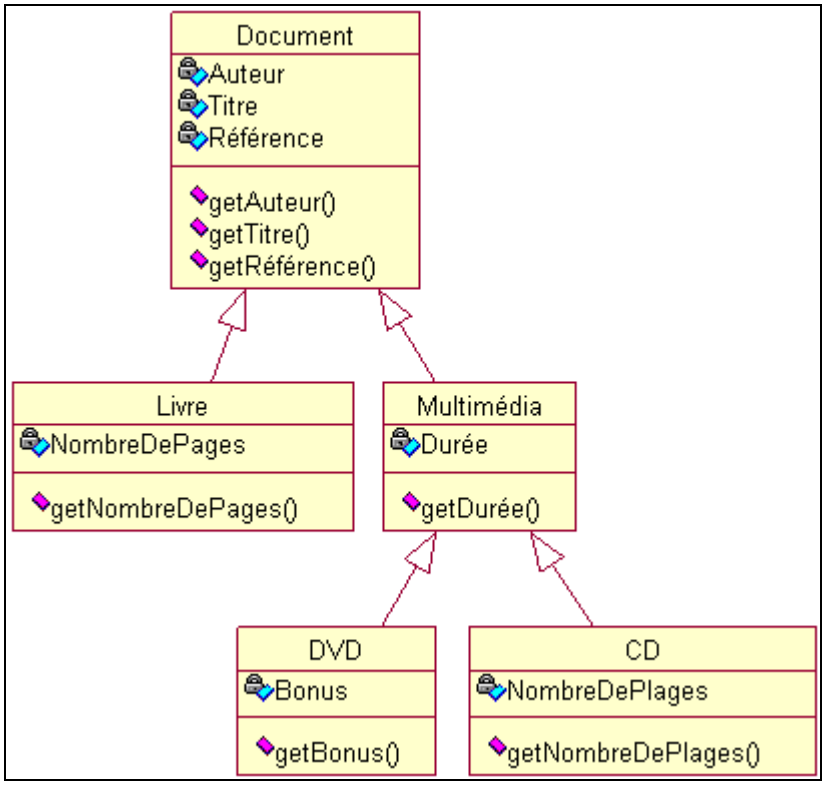
La généralisation se représente par une flèche qui part de la classe fille vers la classe mère.
Par exemple, Un Livre possède, certes un nombre de page, mais en suivant la flèche indiquée par la relation de généralisation, elle comporte également un nom d'auteur, un titre, une référence.

En fait, la classe Livre hérite de tout ce que possède la classe Document, les attributs comme les méthodes.
Finalement, cela correspond exactement à ce que nous avions avant sans l'héritage, sauf qu'avec cette technique, nous évitons toutes les duplications.
Toutes les caractéristiques communes n'apparaissent plus dans chacune des classes filles, alors qu'elles sont bien présentes implicitement.



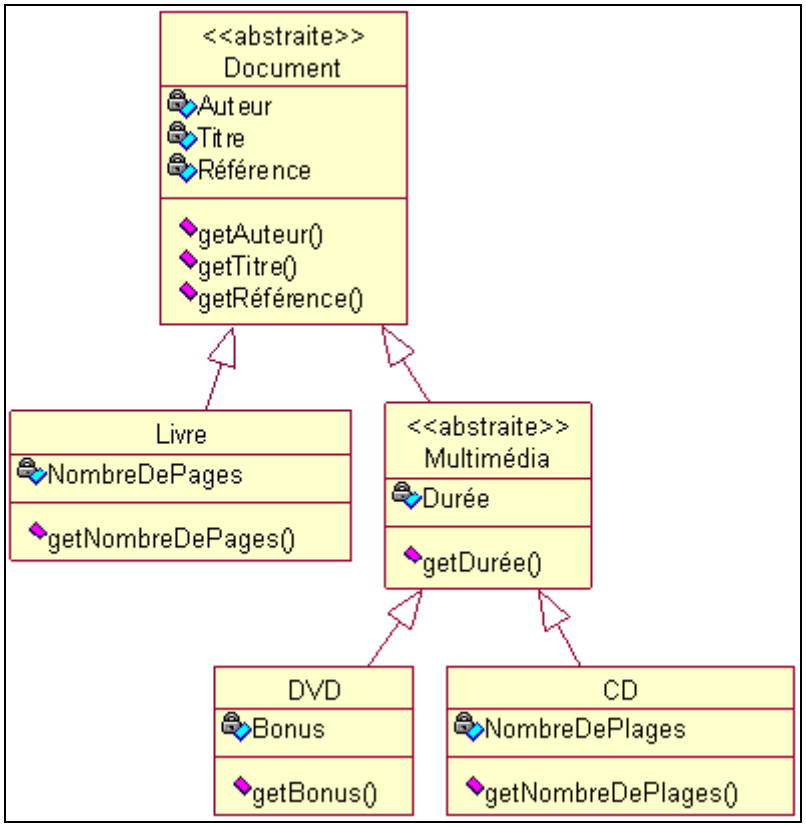
Dans cet exemple, nous avons un seul niveau d'héritage, mais il est bien entendu possible d'avoir une hiérarchie beaucoup plus développée. D'ailleurs, si nous regardons de plus prêt, nous remarquons que nous pouvons appliquer une nouvelle fois la généralisation en factorisant la durée du support CD et du support DVD.

En fait, il s'agit dans les deux cas d'un support commun appelé Multimédia.



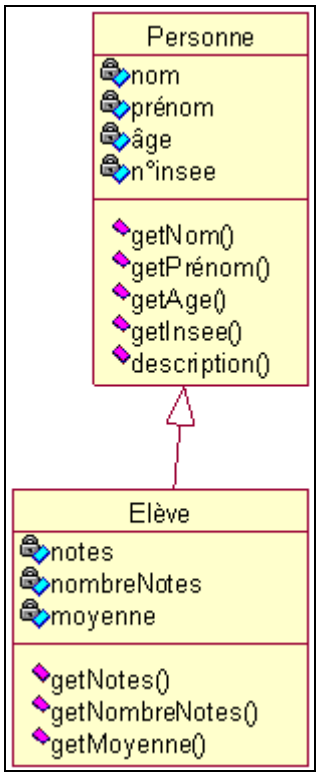
Classes abstraites

Lorsque nous allons mettre en œuvre notre programme de gestion de bibliothèque, nous allons avoir finalement un certain nombre d'objets relatifs à chacun de ces documents.
Toutefois, si nous regardons de plus près, nous n'aurons, par exemple, aucun objet relatif à la classe Document.
En effet, les objets qui intéressent le bibliothécaire, sont les livres, les CDs et les DVDs.
Déclarer un objet de la classe Document n'aurait pas de sens. Il s'agit d'une abstraction. Cette classe n'existe que pour les classes filles qui elles correspondent à quelque chose de concret.
Nous pouvons d'ailleurs appliquer le même raisonnement à la classe Multimédia. Il est d'ailleurs de notre devoir d'empêcher que ces classes puissent fournir des objets. De telles classes sont appelées **abstraites**.
Du cout, les classes classiques sont appelées **classes concrètes**. Nous ne pouvons déclarer des objets que sur des classes concrètes.



Spécialisation.

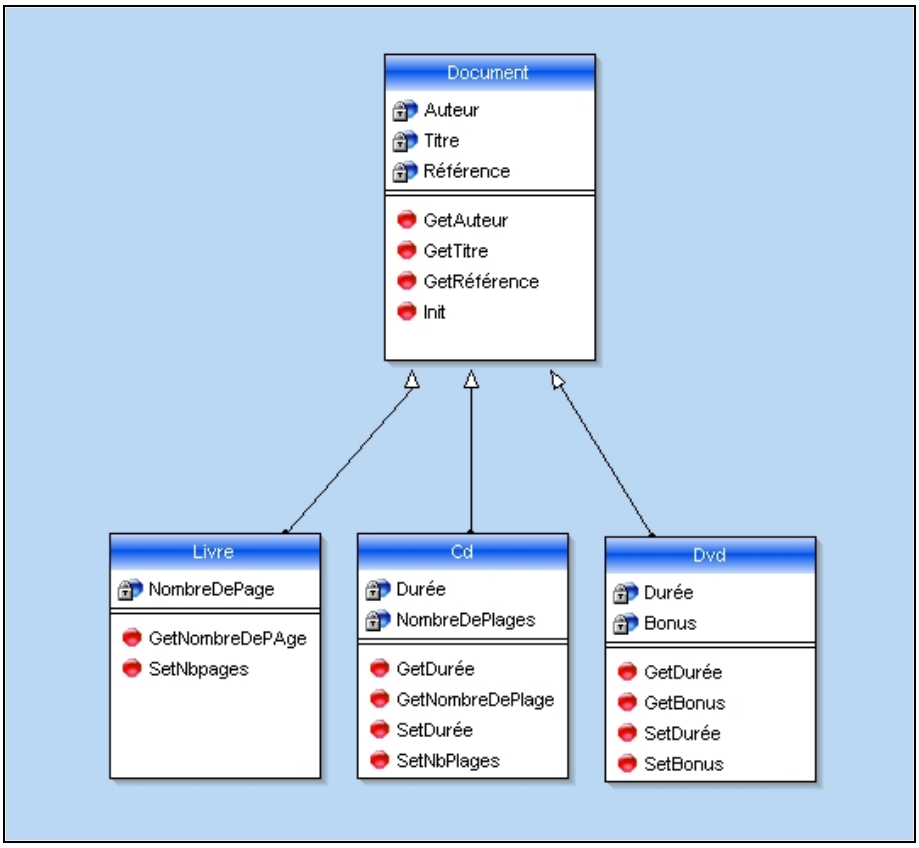
Dans la conception des classes, nous pouvons avoir une démarche inverse de la généralisation, c'est-à-dire, cette fois-ci, de partir plutôt de la classe mère pour aboutir ensuite aux classes filles.
Le concept d'héritage constitue l'un des fondements de la programmation orientée objet. En particulier, il est à la base des possibilités de réutilisation des composants logiciels (en l'occurrence de classes). En effet, il vous autorise à définir une nouvelle classe, dite dérivée, à partir d'une classe existante dite de base. La classe dérivée héritera donc des potentialités de la classe de base, tout en lui ajoutant de nouvelles sans remettre en question la classe de base. Il ne sera pas utile de la recompiler, ni même de disposer du programme source correspondant (exception faite de sa déclaration). Cette technique permet donc de développer de nouveaux outils en se fondant sur un certain acquis, ce qui justifie le terme d'héritage. Comme nous venons de le voir, plusieurs classes peuvent être dérivées de la même classe de base. En outre, l'héritage, n'est pas limité à un seul niveau : une classe dérivée peut devenir à son tour classe de base pour une autre classe. Nous voyons apparaître la notion d'héritage comme outil de spécialisation croissante.
Par exemple, imaginons que nous disposions déjà d'une classe concrète Personne. Nous pouvons alors spécialiser cette classe afin d'obtenir une classe Elève qui reste bien entendu une personne mais qui possède en plus un certain nombre de spécificités comme, par exemple, la gestion des notes. Dans cet exemple, vous remarquez d'ailleurs que les deux classes sont des classes concrètes.



La pratique.

Maintenant que les points théoriques sont posés, passons à la pratique.

Créez un nouveau projet nommé TpObjet8-4 sans analyse. Nous allons faire les classes correspondantes à ce diagramme des classes UML :



Puisque les classes livre, dvd, et cd héritent de document, nous allons commencer par définir Document.

Déclaration de **Document**

```
Document est une classe
    PRIVÉ
        Auteur est une chaîne
        Titre est une chaîne
        Référence est une chaîne
FIN
```

Terminaison de **Document**

Constructeur

```
PROCEDURE Constructeur()
```

Destructeur

```
PROCEDURE Destructeur()
```

Méthode **GetAuteur**

```
PROCEDURE GetAuteur()
RENVOYER :Auteur
```

Méthode **GetTitre**

```
PROCEDURE GetTitre()
RENVOYER :Titre
```

Méthode **GetRéférence**

```
PROCEDURE GetRéférence()
RENVOYER :Référence
```

Méthode **Init**

```
PROCEDURE Init(P_Auteur,P_Titre,P_Ref)
:Auteur=P_Auteur
:Titre=P_Titre
:Référence=P_Ref
```

Document est la classe mère, c'est donc elle qui factorise les méthodes communes. Remarquez la méthode Init, elle remplacera la définition du constructeur dans toutes les classes dérivées, vous voyez le gain qu'apporte la Poo ?

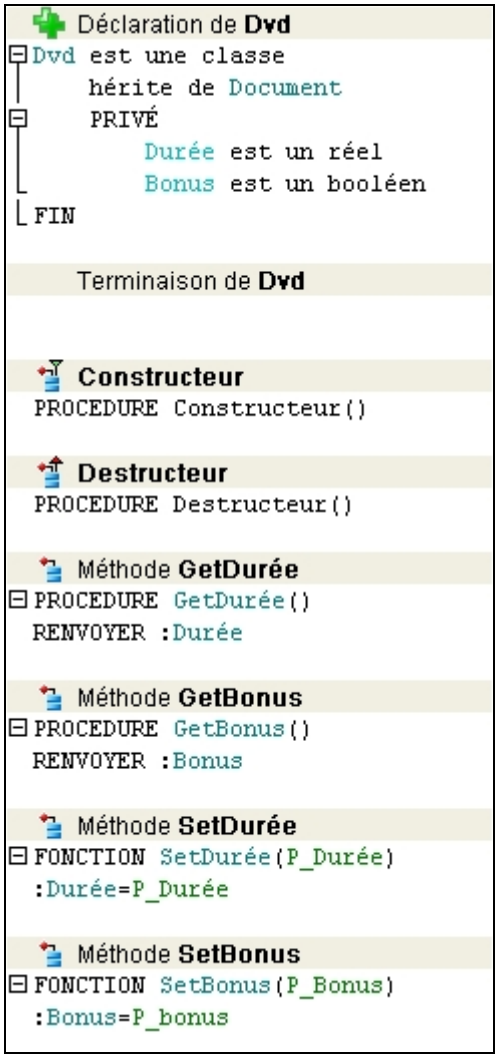
Voici Livre, que du classique (des accesseurs, des mutateurs..):

```
Déclaration de Livre
Livre est une classe
    hérite de Document
    PRIVÉ
        NombreDePage est un entier
FIN

Terminaison de Livre


Constructeur
PROCEDURE Constructeur()

Destructeur
PROCEDURE Destructeur()

Méthode GetNombreDePAge
PROCEDURE GetNombreDePAge()
RENVOYER :NombreDePage

Méthode SetNbpages
FONCTION SetNbpages(P_Nbpages)
:NombreDePage=P_Nbpages
```

Cd :

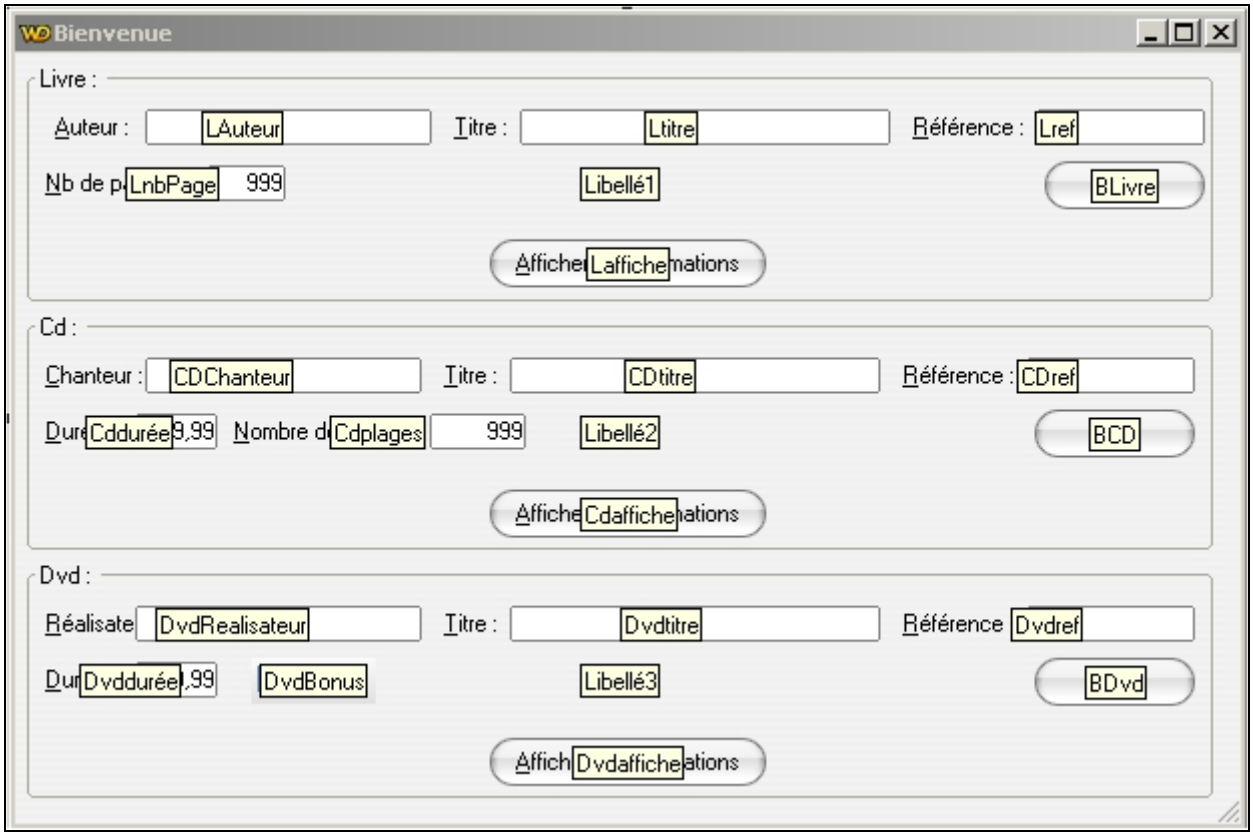
```
Déclaration de Cd
Cd est une classe
    hérite de Document
    PRIVÉ
        Durée est un réel
        NombreDePlages est un entier
FIN

Terminaison de Cd


Constructeur
PROCEDURE Constructeur()

Destructeur
PROCEDURE Destructeur()

Méthode GetDurée
PROCEDURE GetDurée()
RENVOYER :Durée

Méthode GetNombreDePlage
PROCEDURE GetNombreDePlage()
RENVOYER :NombreDePlages

Méthode SetDurée
FONCTION SetDurée(P_Durée)
:Durée=P_Durée

Méthode SetNbPlages
FONCTION SetNbPlages(P_NbPlages)
:NombreDePlages=P_Nbplages
```

Dvd :


Déclaration de Dvd
Dvd est une classe
hérite de Document
PRIVÉ
Durée est un réel
Bonus est un booléen
FIN
Terminaison de Dvd
Constructeur
PROCEDURE Constructeur()
Destructeur
PROCEDURE Destructeur()
Méthode GetDurée
PROCEDURE GetDurée()
RENVOYER :Durée
Méthode GetBonus
PROCEDURE GetBonus()
RENVOYER :Bonus
Méthode SetDurée
FONCTION SetDurée(P_Durée)
:Durée=P_Durée
Méthode SetBonus
FONCTION SetBonus(P_Bonus)
:Bonus=P_bonus

Voici la seule et unique fenêtre du projet :

La même avec les noms des champs qui la composent :



Passons au code de la fenêtre :

Les objets seront des objets dynamiques déclarés global à la fenêtre :

```
Déclarations globales de Départ
livl est un objet Livre dynamique
Dvdl est un objet Dvd dynamique
cdl est un objet Cd dynamique
```

Voici le code du bouton **Créer** de la zone livre :

```
Clic sur BLivre
livl=allouer un Livre
livl:Init(LAuteur,Ltitre,Lref)
livl:SetNbpages(LnbPage)
```

Comme vous le voyez liv1 utilise la méthode Init définie précédemment dans la classe document et agit sur les attributs auteur, titre et référence. Comme tout est hérité init, auteur, titre, référence appartiennent bien à livre, nous n'avons pas eu besoin de les réécrire, magique !

Voici le code du bouton **Afficher les informations** de la zone livre :

```
Clic sur Laffiche                                   Erreur : manuel   Exception : manuel
Info(livl:GetTitre()+" écrit par : "+livl:GetAuteur()+" Nombre de pages : "+livl:GetNombreDePAge()+" Référence : "+livl:GetRéférence())
```

Voici le code du bouton **Créer** de la zone Cd :

```
Clic sur BCD
cdl=allouer un Cd
cdl:Init(CDChanteur,CDtitre,CDref)
cdl:SetDurée(Cddurée)
cdl:SetNbPlages(Cdplages)
```

Voici le code du bouton **Afficher les informations** de la zone Cd :

```
Clic sur Cdaffiche                                  Erreur : manuel   Exception : manuel
Info(cdl:GetTitre()+" Composé par : "+cdl:GetAuteur()+" Durées : "+cdl:GetDurée()+" Nombre de plages : "+cdl:GetNombreDePlage())
```

Voici le code du bouton **Créer** de la zone Dvd :

```
Clic sur BDvd
Dvdl=allouer un Dvd
Dvdl:Init(DvdRealisateur,Dvdtitre,Dvdref)
Dvdl:SetDurée(Dvddurée)
Dvdl:SetBonus(DvdBonus)
```

Voici le code du bouton **Afficher les informations** de la zone Dvd :

Clic sur **Dvdaffiche** Erreur : manuel Exception

```
Bonusp est une chaîne

SI Dvdl:GetBonus()=1 ALORS
    bonusp="Oui"
SINON
    bonusp="Non"
FIN

Info(Dvdl:GetTitre()+" réalisé par : "+Dvdl:GetAuteur()+" Bonus : "+bonusp+" Référence : "+Dvdl:GetRéférence())
```

Voilà, c'est fini, je vous laisse remplir les champs, cliquer et observer les comportements. Je pense que vous devriez être séduit par l'élégance de la Programmation Orientée Objet.

A bientôt pour le Polymorphisme.