1 CONCEPT DE BASE

1.1 Visual Basic 6

1.1.1 Pour quoi faire ?

VB est un langage de type RAD : « Rapid Application Development ». Il permet de créer des applications rapidement grâce à une conception simple de l'interface utilisateur, et un code plus simple que des langages de plus bas niveau de type C.
Il dispose de nombreux objets prédéfinis destinés à la gestion de données. C'est pour cette raison qu'il nous intéresse particulièrement.

VB n'est pas destiné au développement d'applications exigeant des performances élevées (graphisme, calculs complexes…).
C'est un langage particulièrement adapté à la manipulation de données stockées dans tout type de base (SQL Server, Access, Oracle, DB2, Ingre…).

1.1.2 Les versions disponibles

VB6 renforce les évolutions apparues dans VB5. Il est possible de compiler en code natif 32 bits exclusivement (Windows 32 bits). Il intègre aussi la technologie ActiveX qui est une technologie dans laquelle Microsoft a englobé les concepts objets. La création de contrôle ActiveX était impossible en VB4, il fallait les faire en C++.

L'éditeur permet de travailler en mode MDI (Multiple Document Interface) ou SDI (Single Document Interface), et contient une aide à la saisie (propriétés et méthodes). On peut ouvrir plusieurs projets simultanément, ce qui facilite par exemple de débogage de composants ActiveX personnalisés. Il est possible de créer ses propres assistants et modèles de feuille ou de projet.
Il faut aussi mentionner VBA (VB pour application) qui permet de programmer les logiciels de la suite Office. L'éditeur VBA depuis Office 2000 est très proche de VB 6. VBA est un langage interprété.

L'évolution des produits de la suite Visual Studio 6, dont fait partie VB6, se poursuit surtout avec l'apparition de nouveaux objets, notamment d'accès aux données ou dédiés à Internet. Quelques compléments (concepteurs) permettent d'accéder à des fonctionnalités directement dans l'interface de développement. Les passages de paramètres sont enrichis et acceptent des tableaux et des types personnalisés.

Le service pack 6 (SP6) est le dernier disponible pour visual studio 6. Ce dernier devrait être maintenu par Microsoft jusqu'en 2008.

1.1.3 « .NET »

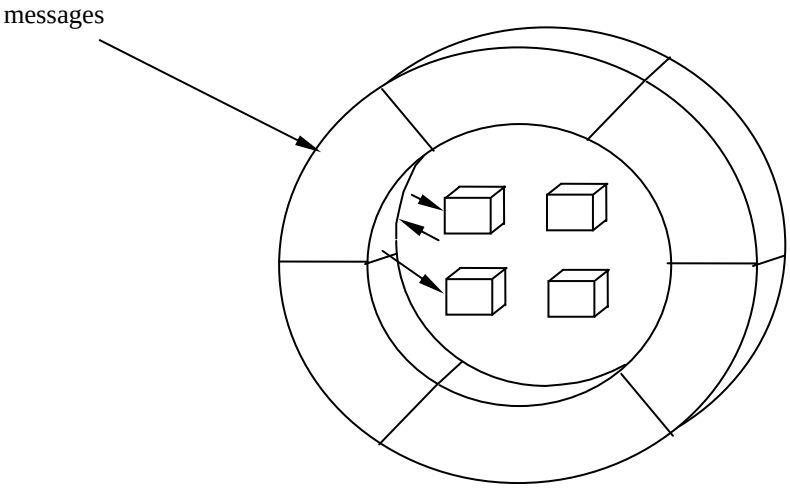
Cet environnement est assez différent de Visual Studio 6. VB a beaucoup évolué en intégrant les concepts objets qui lui manquaient. Il a aussi gagné en complexité et l'environnement de développement nécessite des machines puissantes.

Bien qu'assez différent de VB6, son apprentissage sera facilité par une bonne connaissance de VB6 qui permet d'appréhender les concepts de base plus simplement.

1.2 Programmation orientée objet POO

L'approche orientée objet permet de représenter plus fidèlement la réalité d'un modèle fonctionnel.
Les objets réels possèdent une face visible ainsi qu'une partie cachée (une machine à café). La partie visible présente les moyens d'interaction avec l'extérieur : bouton de sélection de la boisson, paiement, récupération de la boisson. La partie cachée ne nous intéresse pas tant que l'objet nous rend le service attendu. C'est le principe de **l'encapsulation**.

Un objet est un programme qui contient des variables, des procédures et des fonctions accessibles ou non par d'autres programmes qui les exploitent.



Méthode accédant aux données

Les **propriétés** sont les variables publiques qui définissent des états statiques de l'objet. Les variables publiques sont visibles du programme appelant (extérieur), alors que les variables privées sont utilisées pour le fonctionnement interne de l'objet.

Les **méthodes** sont des actions (aspect dynamique) exécutées par l'objet sur demande du programme qui les appellent. Il s'agit de procédures ou fonction publiques "exposées" par l'objet. La visibilité des procédures et fonctions suit le même schéma que les variables. Il est possible de passer des paramètres aux procédures et fonctions, et récupérer des valeurs retournées par les fonctions. Encore une fois, les procédures cachées effectuent des traitements internes qui interagissent avec les procédures visibles.

Certains objets peuvent contenir plusieurs objets de même type, on parle alors de **collection**. Les collections possèdent également des propriétés et des méthodes qui permettent de manipuler les objets qu'elles contiennent.

Les conventions veulent que les collections portent le nom des types d'objets qu'elles contiennent auquel ont ajoute un "s". Par exemple, la collection "DVDs" contient des objets "DVD".

Par ailleurs, la notion de collection laisse apparaître que des objets sont contenus dans d'autres. Les objets sont effectivement hiérarchisés. Certains objets ne sont même accessibles qu'au travers de leur "**conteneur**". Mais attention, un conteneur n'est pas une collection. Nous verrons plus loin que le recours au conteneur pour accéder à un objet évite toute ambiguïté.

Enfin, pour utiliser un objet, il faut le déclarer. Cette déclaration crée une **instance** de cet objet. C'est-à-dire une copie basée sur un modèle d'objet possédant des propriétés, procédures et fonctions. Un type d'objet est appelé une **classe**. Une instance de classe est une variable objet de la classe en question.

Considérons les plans d'une maison. Il s'agit d'une définition qui n'a pas d'existence réelle. C'est la **classe** Maison. Si on construit une ou plusieurs maisons à partir de ce plan, on crée une ou des **instances** de la classe maison.

La syntaxe utilisée pour manipuler les objets se présente comme suit :

```
Objet.Propriete
Objet.Methode [Parametre]
```

Ex:

L'objet MAISON contient plusieurs objets PIECE, qui contiennent plusieurs objets MEUBLE.

Créons un objet de type MAISON, contenant une PIECE, contenant plusieurs meubles :

```
Declare MaMaison comme Maison
Declare Cuisine comme Piece
Declare Four comme Meuble
Declare Frigo comme Meuble
Declare Meubles Comme Collection
```

On peut envisager les propriétés et méthodes suivantes :

```
Four.Largeur = 60
Frigo.Couleur = « Blanc »
Frigo.Largeur = 90

Four.Porte.Ouvrir
Four.Chauffer 210
```

On pourrait ajouter le Four et le Frigo à la collection **Meubles**, afin de pouvoir agir sur l'ensemble des objets issus de la classe Meuble au travers de la collection. Des fonctionnalités permettent de parcourir une collection à l'aide de boucles.

```
Meubles.Ajouter Four
Meubles.Ajouter Frigo
```

On peut préciser qu'un objet est contenu dans un autre. Par exemple, on ajoute des objets Meuble à une PIECE. La méthode Attacher de l'objet Cuisine, instance de la classe PIECE le permet :

```
Cuisine. Attacher Four
Cuisine. Attacher Frigo
```

Il faut noter la nuance qui existe entre la collection Meubles et la classe Pièce. La pièce est un « conteneur. » Elle rassemble plusieurs objets de types différents. La collection ne peut contenir que des objets **Meuble** indépendamment des pièces où ils se trouvent. Il s'agit de pouvoir manipuler plusieurs objets de même classe plus facilement, sachant que le nombre d'objets peut varier (à l'aide de boucles).

On peut également accéder aux objets d'une collection par un index (boucles) pour connaître tous les membres de la collection :

```
For i = 0 to Meubles.Membres.Nombre - 1
        Afficher Meubles.Membres(i).Nom
Next i
```

Cet exemple utilise une boucle allant de 0 au nombre d'individus moins un contenu dans la collection pour afficher la propriété Nom de chaque membre, à l'aide de la procédure Afficher.

Remarquons que les classes Frigo et Four possèdent une propriété ayant le même nom : *Largeur*. C'est le **polymorphisme**. Une propriété ou une méthode est constitutive d'une classe. Chaque classe est un programme élémentaire, et les propriétés et méthodes de deux classes sont dans des « portées » différentes. Ceci explique que des propriétés et méthodes de classes différentes peuvent avoir le même nom. On définit toutes les propriétés et méthodes d'un objet lors de la conception de la classe.

Par exemple, deux pièces d'un jeu d'échec peuvent avoir une même méthode *Deplacer* sans qu'il y ait de confusion, bien que le déplacement soit différent selon les pièces (cavalier, pion). En effet, les méthodes *Deplacer* sont programmées dans deux classes distinctes.

```
        Cavalier.Deplacer
        Pion.Deplacer
```

Un dernier point porte sur deux méthodes particulières des objets que sont le **constructeur et le destructeur**. Il s'agit de méthodes qui sont exécutées lors de la création d'une instance de l'objet (initialisation), puis lors de la destruction de l'instance. Ce code est exécuté automatiquement. Il permet par exemple d'initialiser des propriétés (Frigo.Couleur = « Gris ») au démarrage, ou de fermer des fichiers, de déconnecter une base etc. à la fermeture.

1.3 Exécution d'un programme VB

Un programme séquentiel se déroule selon trois phases majeures :

- l'initialisation (variables, saisie de paramètres etc.) ;
- boucle d'exécution (tant que pas quitter) ;
- Fin.

Dans un programme VB, l'utilisateur est au centre de l'exécution car c'est lui qui déclenche les procédures par ces actions sur l'interface.

Les objets possédant une interface peuvent être "activés" par l'utilisateur en répondant à des **évènements**. Les évènements auxquels répondent les objets dépendent de la nature des objets en question. Ainsi, on trouvera fréquemment *Sur Click*, *Sur Activation*, *Sur Ouverture*, *Sur Changement*…. Le déroulement n'est donc pas linéaire.
Les objets "surveillent" leurs évènements en permanence. Lorsque le programmeur a positionné du code sur un événement, celui-ci s'exécute lorsque l'événement est déclenché.

L'utilisateur n'est pas le seul à pouvoir déclencher des évènements, le programme le fait également. Par exemple, le fait de modifier le contenu d'une zone de texte par programme déclenche l'événement *Sur Changement* de la zone de texte modifiée. Cet événement est donc déclenché par l'utilisateur et par le programme.

Il faut bien maitriser le déclenchement des événements lors de l'écriture de son programme pour éviter des comportements imprévus, et donc non gérés.

L'essentiel

Assimiler le vocabulaire lié à la programmation objet ainsi que les concepts de base.
Bien comprendre ce qu'est une méthode et une propriété d'un objet.
Identifier clairement les relations qui lient une classe et une instance de classe.
Comprendre la notion de conteneur et le moyen d'accéder à un objet au travers de son parent.

Assimiler l'exécution événementielle par rapport à l'exécution séquentielle.