OOP en VFP

Burt Rosen (extrait de la News letter avril 2006 vfug)

Quelques principes de base de OOPS -programmation orientée objet. Ces quelques lignes si elles font référence a VFP, s'appliquent également a tous les programmes orientés objet.

Les principes de OOPS sont difficiles a saisir surtout pour des termes tels que : Héritage (Inheritance), Encapsulation et Polymorphisme (IEP). Le but de ces quelques lignes est d'essayer de simplifier la compréhension de ces concepts.

Objets

Avant d'aborder la description de ces termes, il est nécessaire de comprendre les effets des IEP sur les choses.
IEP affecte les objets.
Un objet est un élément dans un programme qui peut être :

- un élément de base sur une forme,
- la forme elle-même
- la collection de tous les objets sur une forme
- une code
- des données (data)

ceci indépendamment de toute image visuelle.

L'objet est donc l'élément de base de la programmation orientée objet. Il seront expliqués d'une manière plus approfondie ci-dessous.

Si vous vous souvenez des cours scolaires de chimie, le monde physique peut-être représenté par une collection de molécules. Une molécule est un atome, tel l'Hélium (He), ou une collection d'atomes comme H_2O c'es à dire l'eau. Une molécule est l'équivalent d'un objet VFP. L'objet consiste en un simple élément comme le « text box » ou il peut être extrêmement complexe en comprenant des multiples objets de différents types comme une « form ».

Chaque objet dans VFP aura un type. Les types d'objets sont du genre boite de dialogue (textbox), boite de contrôle (checkbox) collections personnalisées et beaucoup d'autres. Ces différents types sont définis dans les classes de bases. Les classes de bases de VFP sont des définitions de tous les différentes classes primitives qui peuvent exister. En terme de chimie, se sont les atomes. Vous créez des objets complexes (molécules) en combinant différents types de classes de bases et ce de différentes manières.

Revenons à notre exemple chimique, une des molécules les plus complexe est le groupe des Hydrocarbonés. En terme simple, ces molécules sont faites d'hydrogène et de carbone. Le moyen pour différencier les différents hydrocarbonés est la quantité d'hydrogène et de carbone de chaque molécule et également l'ordre des atomes d'hydrogène et de carbone.
Pour ceux qui n' aurais pas reconnus ces termes, les hydrocarbonés sont les carburants tel l'essence, le diesel, etc.

De manière similaire, les objets d'un programme peuvent être construits en combinant une série d'autres objets pour créer un ensemble. Jetons un œil sur une « form », elle peut contenir les objets suivants : Boite de dialogue(Editbox), des étiquettes (Label), des boutons de commandes (CommandButton), des boites d'éditions (EditBox), des boutons d'options(OptionButton) etc. La « form » peut également avoir une série d'objets «composés comme les commande de groupe(commandGroup) soit 2 ou plusieurs boutons de commandes). Tous ensemble cette [***collection***] d'objets forme un nouvel objet qui représente notre « form » Collection est en gras car ce mot en VFP est une classe de base.
Une Collection est un objet contenant d'autres objets.

Un mot d'avertissement est ici nécessaire. **N'utilisez pas** dans vos applications les classes de bases de VFP. Car ces classes de base ne peuvent pas être modifiées.(dans l'application)

L'argument principal d'utiliser des classes, est de créer une hiérarchie dans laquelle vos standards personnels peuvent être mémorisés. Si vous écrivez vos programmes sans une ou plusieurs classes intermédiaires vous n'aurez pas la possibilité d'introduire la moindre variation dans la classe de base.

En construisant un ensemble de classe intermédiaire, vous construisez en fait un cadre de travail.(framework)

Pourquoi la classe de base est elle importante ?
C'est simple – tous les objets doivent avoir une classe de base.

La classe de base est le point de départ des tous les objets. Un donc objet ne peut donc exister sans elle. Cela peut engendrer un peu de confusion lorsque l'on regarde un objet qui contient de multiples objets. Si nous prenons notre « form », la « form » elle-même a une classe de base. Chaque éléments qui la compose ont également une classe de base.
La clé du concept a retenir ici, est que chaque éléments contenus dans la « form » auront leur classe de base propre.
Chaque objet doit avoir une et une seule classe de base.
Tous les objets contenus peuvent avoir la même classe de base (tels les boites de dialogues(TextBox) si tous les objets contenus sont des boites de texte), ou les objets contenus auront différentes classes de base (comme les ensembles(collection) d'étiquettes(Label) et boites de dialogues(TextBox))
La classe de base définira les ***Propriétés*** et les ***Évènements*** qui sont inhérents pour tous les objets basés sur la classe de base.

Propriétés, Évènements et Méthodes - PEM

Tous les objets auront des Propriétés, Évènements et Méthodes (PEM).

Les **propriétés** pour une boite de dialogue(TextBox) comprendront la description entière des propriétés comme : la couleur, la position, dimensions et autres paramètres qui spécifierons la nature de ce TextBox spécifique.

Un **évènement** est une action qui se déclenche pendant l'exécution du programme. Tel :

[Click]	lorsque l'utilisateur click sur cet objet
[Init]	lorsque l'objet est initialisé
[Destroy]	lorsque l'objet est détruit

Tous ces évènements peuvent être liés à du code que l'on appelle des Méthodes.

Une **méthode** est une fonction ou procédure incorporée dans une classe.

Elle peut être invoquée par programmation, ou exécutée lorsqu'un évènement spécifique se produit.
Il est très facile de confondre les Évènements et les Méthodes. Dans le l'analyseur de classe (Class Browser) vous trouverez un onglet « Méthodes ». Les méthode sont les endroits ou vous sauvegarderez le code des fonctions qui seront lancées lorsque celles-ci seront appelées. Quelques unes de ces méthodes sont aussi des évènements – et qui se passeront automatiquement car appelées par VFP telles : « Init, Load, Error et Destroy » D'autres méthodes seront-elles définies par l'utilisateur comme « CheckForValidSSN ». Lorsqu'un évènement se déclenche, VFP exécutera le code écrit dans la méthode ayant le même nom que l'évènement. Les méthode définies par l'utilisateur doivent être invoquées directement par du code programme, soit par un bouton sur la « form », soit un appel au sein d'une autre méthode.

Inhéritance

Nous avons abordés les principes de bases des classes et PEM, l'on peut aborder l'héritage (Inheritance).

Si nous pensons comme en génétique, tout le monde a des parents, grands-parents, arrière-grands-parents, etc. Comme tous les êtres humains,(je présume que tous vous êtes des êtres humains-parfois j'ai des doutes me concernant), nous possédons tous des traits et caractéristiques de nos ancêtres. Même si nous ne savons pas qui sont nos parents, nous avons hérité de leur matériel génétique.

Le concept OOP d'héritage est similaire. Tous les objets héritent d'une ou plusieurs classes, tout au long du cheminement vers la classe objet de base.
Qu'est-ce que cela signifie ?
Cela veut dire que tous les PEM dans la chaine d'héritage sont effectif dans l'objet, pour autant qu'il n'ait pas été marqué comme modifié. En d'autres mots, un «TextBox» a certaines propriétés et évènements hérités. Ceux-ci sont absolus et définis dans la classe de base TextBox.

Supposons que nous ayons une classe MonTextBox. Cette classe est basée (hérite) de la classe de base TextBox. MonTextBox héritera automatiquement de toutes les propriétés de la classe de base.
L'on peut modifier la couleur de la police de caractère pour que le texte soit en rouge au lieu de noir. L'on peut également ajouter des méthodes qui seront exécutées lorsque certains évènements, tel « INIT » sont exécutés.

Maintenant, nous créons une nouvelle classe "AppTextBox". Nous la créons non pas en utilisant la classe de base TextBox, mais bien avec « MonTextBox ».
En réalisant cette sous-classe, nous héritons des Propriétés et Méthode spécifiées dans MonTextBox.
En plus les PEM non modifiés dans MonTextBox, héritent automatiquement des valeurs définies dans la classe TextBox de base.

Que veux dire l'héritage d'un point de vue pratique?
Celà signifie que l'on peut réaliser des changements universels dans une classe intermédiaire.
Lorsque l'on utilise une classe intermédiaire, notre application héritera de tous les paramètres standards qu'elle comprend. Par exemple le comportement par défaut d'un TextBox est un champ texte actif pour afficher un texte noir sur fond blanc. Le même mais inactif affiche le texte en gris foncé sur fond gris clair. Je trouve cette combinaison particulièrement difficile a lire.(ce n'est pas parce que je deviens vieux mais parce que mes yeux sont trop expérimentés !)
J'ai modifié ma classe TextBox intermédiaire (sous-classe) et modifé l'instruction
« DisabledForeGroundColor » en noir. Maintenant lorsque j'utilise un TextBox dans mon application, J'ai les couleurs standard de base en mode actif, et le texte en noir sur fond gris clair en mode inactif, au lieu du gris sur gris clair

Encapsulation

L'encapsulation est un concept intéressant. Si vous pensez a une chenille dans un cocon, la chenille y est encapsulée. C'est-à-dire enfermée et protégée du monde extérieur.

C'est le second élément majeur en OOP. Chaque objet est encapsulé et protégé du monde externe.
L'encapsulation nous permet de sauver des données(propriétés) comme partie d'un objet, ou du code. (méthodes) Comme programmeur, l'on a la possibilité de définir ce que le monde externe peut ou ne peut pas voir à l'intérieur des objets.
L'on peur ouvrir les Propriétés et les Méthodes pour y accéder, ou les fermer, ou une combinaison des deux.

Mais, et c'est un mais critique, les propriétés et méthodes restent encapsulées dans l'objet. Si un objet a une propriété appelée « Valeur », celle-ci est unique a cet objet et n'entrera pas en conflit avec la propriété appelée « Valeur » dépendante d'un autre objet. Prenons l'exemple de 2 objets, objet1 et objet2 les deux ont une propriété « Valeur »

object1.value

Est complètement different et un champ séparé que

object2.value

Les deux sont des champs différents et n'entrerons pas en conflit l'un l'autre.

Une autre chose importante a se rappeler , c'est la différence entre un objet et une instance de l'objet. Nous créons par exemple un objet « MonSSN » basé sur un objet SSN. Ensuite nous créons un deuxième objet « Votre SSN » également basé sur l'objet SSN. Ce sont des instances séparées de l'objet et non rien a voir l'un l'autre. Donc même si les deux sont basé sur le même objet, ils sont encapsulés vis-à-vis l'un de l'autre.

Polymorphisme

Finalement nous arrivons au Polymorphisme.
Simplement di, cela signifie que ces deux objets avec le même nom ne sont pas nécessairement le même objet. Pour simplifier cette situation, supposons que nous ayons une forme, et que sur celle-ci se trouve un bouton « Impression ». Ce dernier aura un évènement « Click » associé. L'on peut y placer du code pour imprimer automatiquement un rapport en relation avec la forme

Maintenant l'on utilise une deuxième forme qui possède également un bouton « Impression » mais celui-ci nous amène un menu de choix de rapport pouvant être lancés.

Nous avons maintenant 2 formes (form1 et form2), chaqu'une ont un bouton "Impression" et un évènement « click » mais les deux évènements « Click » sont complètement différents. L'on parle alors de polymorphisme. Deux objets de même nom ne sont pas nécessairement le même objet. Ils peuvent avoir des fonctions différentes.

Hiérarchie de Classe et Hiérarchie Objet

L'un des point de confusion les plus fréquemment rencontrés lors de discussion avec d'autres programmeurs, c'est la différence entre la hiérarchie de classe et la hiérarchie objet.

La hiérarchie de classe définit la chaine d'héritage.
La hiérarchie d'objet, définit la chaine d'encapsulation.

Pensez comme ceci: la première est celle de votre héritage génétique. C'est la définition de vos parents, grands-parents, arrière-grands-parents.
Vous héritez des PEM de votre classe hiérarchique.

Votre hiérarchie objet est comme au travail, Votre travail, dépend d'un département, lui-même d'un bureau, d'une région, d'une société. Votre travail est encapsulé par votre département, lui-même par un bureau etc..

Reprenons l'exemple cité d'une forme avec 2 pages onglets avec un TextBox sur la deuxième page. Si l'on doit référencer ce TextBox, on le fera de la manière suivante :

oForm.oPageFrame.oPage2.oTextBox

LeText Box héritera de MonTextBox, qui hérite lui du TextBox de la classe de Base.

Conclusions

Dans cet article, nous avons vu les definitions de base de l'héritage, l'encapsulation et le polymorphisme intentionnellement cette explication était très générale et sans détails.
En espérant qu'elle vous auras fournis les bases de la compréhension et plus important l'explication de la programmation par objet.

Trad. Guy Bonemme