# *Використання VBA*

# Ласкаво просимо в Visual Basic for Applications

Протягом багатьох років глава корпорації Microsoft Білл Гейтс (Bill Gates) висловлював думку об BASIC як про найбільш просту мову високого рівня для всіх типів прикладних програм. Ця точка зору привела в кінцевому результаті до появи Visual Basic for Applications. Уперше використавши Visual Basic for Applications в Excel і Project, фахівці корпорації Microsoft потім застосували його у всіх своїх найбільш популярних пакетах, включаючи Word, Mail і PowerPoint.

Перша частина цієї книги знайомить з Visual Basic for Applications і його зв'язками з Excel, подальші частини з особливостями мови і програмуванням на Visual Basic.

# Що таке Visual Basic for Applications?

Visual Basic for Applications є-новітнім втіленням Microsoft Visual Basic 3.0 для Windows. Фахівці корпорації Microsoft перенесли Forms Designer з Visual Basic, зв'язали Visual Basic з Excel, а потім додали діалогові таблиці замість Forms Designer. Додатково вони зробили всі команди і об'єкти Excel (елемента, таблиці і т.п^ доступними для Visual Basic, що дозволило Visual Basic управляти всіма функціями Excel. Таким чином, в цей час Visual Basic є "супермакромовою" для Excel, що базується на добре відомих командах і синтаксисі BASIC.

Доповнюючи Excel, Visual Basic є також макромовою для Project, а незабаром стане і макромовою для Word, Mail і PowerPoint. Якщо ви намучилися з різними мовами і операційними системами, то можете зітхнути вільно. Більшість з випущених в останні роки основних прикладних пакетів були реалізовані зі своїми власними макромовами, і всі ці макромови відрізнялися один від одного і від всіх відомих мов програмування. Створення корпорацією Microsoft універсальної макромови значною мірою позбавляє від головного болю, пов'язаного з розробкою призначених для користувача команд і процедур.

Той, хто добре знайомий з старішими версіями BASIC, такими як BAS1CA і GW-BASIC, може передбачити, що Visual Basic це примітивна мова програмування, що не має сучасних конструкцій і структур даних. Однак це не так. Хоч в Visual Basic дотримуються синтаксис і правила програмування на BASIC і виконуються програми, написані на BASICA і GW-BASIC, Visual Basic є сучасною мовою програмування, повністю оснащеною всіма необхідними структурними операторами і типами даних. Крім того, в Visual Basic застосовуються об'єкти, що робить його використання при розробці прикладних програм бо.чсг зрозумілим і легким. І хоч рівень об'єктно-орієнтованого програмування на Visual Basic не так високий, як на З++, Visual Basic управляє Excel, маніпулюючи Excеl-об'єктами. Використовуючи Visual Basic, не можна створювати нові класи об'єктів, але можна впритул наблизитися до рішення цієї задачі.

# Що можна зробити на Visual Basic for Applications?

Visual Basic не тільки полегшить створення макропроцедур, але і спільно з OLE 2.0 дозволить створювати "мегапрограми”, об'єднуючі і координуючі зусилля багатьох могутніх прикладних програм. Наприклад, можна створити програму підготовки документа, яка буде використати Excel для доступу до бази даних, генерації графіків і таблиць, Word для включення цих графіків і таблиць в текст звіту, Mail для розсилки звіту, і все це натисненням однієї кнопки. Visual Basic буде "клеєм", який все об'єднає і упорядкує. Оскільки вказані програми є об'єктно-орієнтованими, будь-яка програма на Visual Basic буде мати доступ до всіх команд і даних в них за рахунок доступу до об'єктів цих програм.

# Що таке об'єкти?

Що являють собою об'єкти в комп'ютерному коді? Відомо, що таке фізичний об'єкт. Ваш маніпулятор миша, клавіатура і комп'ютер такі ж фізичні об'єкти, як і ваша чашка кави. Як же всередині вашого комп'ютера код перетворюється в об'єкт?

Об'єктно-орієнтоване програмування це найбільш сучасний стиль в розробці комп'ютерних програм. Цей стиль використовується тому, що сучасне конструювання програм намагається добитися певної специфічної мети. Комп'ютерні програми повинні бути:

* що перевіряються
* що модернізуються
* що повторно використовуються
* переносимими

Програми повинні бути такими, що перевіряються, щоб ви могли пересвідчитися, що вони роблять саме те, що повинні робити. Програми повинні бути такими, що модернізуються, щоб ви або будь-хто інший могли зрозуміти програму досить добре, щоб внести коректні зміни. Програми повинні бути повторно такими, що використовуються, щоб ви, лише одного разу написавши процедуру, потім могли повторно використати її при будь-якій необхідності. І, нарешті, програми повинні бути переносимими, щоб перенести їх на різні платформи з мінімальними переробками.

Сучасна практика програмування шукає досягнення цих цілей в модульності комп'ютерних програм. Модульні комп'ютерні програми при розробці розбиваються на окремі частини, що іменуються модулями. Кожний модуль виконує специфічні, суворо певні функції перетворення і має доступ тільки до тих даних, які необхідні для даного перетворення. Код модуля, розроблений зі суворо певним інтерфейсом для інших модулів програми, дуже легкий для відладки, супроводу і розуміння. Іншим аспектом модульності є його замкненість, що додає упевненість в тому, що будь-які зміни, які ви вносите в код модуля, вплинуть тільки на функції цього модуля і ні на що інше. Таким чином, ці особливості полегшують відладку, супровід і розуміння коду.

Об'єктно-орієнтоване програмування максимально використовує принцип модульності. Програмний об'єкт в ООП називається "контейнером". Контейнер включає в себе дані і код, який знає, як маніпулювати цими даними. Іншими словами, програмний об'єкт це просто блок, що складається з коду і даних. Він може бути представлений як реальний фізичний об'єкт, наприклад у вигляді командної кнопки. Програмний об'єкт володіє специфічною функцією перетворення і містить весь необхідний код і всі необхідні дані для реалізації цієї функції, але його внутрішня робота прихована від стороннього ока.

У більш традиційному програмуванні комп'ютерна програма розроблялася у вигляді функціональних наборів рядків, в яких блоки даних передавалися від модуля до модуля і кожний модуль модифікував або використав ці дані своїм способом. Модулі не містили даних, а тільки коди для модифікації будь-яких переданих даних. Якщо програма випадково передасть такій процедурі невірні дані, процедура все одно обробить їх, повернувши вам "сміття", або взагалі викличе аварію вашої системи.

У об'єктно-орієнтованому програмуванні дані і код, який маніпулює цими даними, об'єднані в структурі, званій об'єкт. Замість того, щоб передавати дані від модуля до модуля для виконання різних обчислень, ви посилаєте повідомлення об'єкту, що містить ці дані, і таким чином доручаєте йому виробити обчислення. Передати об'єкту погані дані неможливо, бо дані, з якими об'єкт маніпулює, включені в цей об'єкт. Більш того не можна викликати або виконати об'єкт можна тільки послати йому повідомлення з ввічливим проханням виконати одну із задач, що вирішуються ним (звичайно, якщо об'єкт не знайде можливості здійснити необхідну маніпуляцію, то, швидше усього, йому доведеться ввічливо повідомити вам, що він не може задовольнити ваш запит)

Загальні приклади об'єктів Visual Basic і Excel це таблиці, області осередків, командні кнопки, текстові вікна, робочі папки, діаграми і модулі.

# Властивості і методи

Програмний об'єкт володіє певними властивостями і методами. Властивості це видимі характеристики об'єкта, а методи операції перетворення цих даних. Видимими характеристиками називаються дані, які можуть бути доступні поза об'єктом. Властивостями вважаються дані, якими об'єкт маніпулює АБО які дозволяють контролювати, як об'єкт виглядає або як він себе веде. Наприклад, властивість Value текстового вікна це текст, який вводиться в це вікно. Властивість шрифту Color управляє тим, як він виглядає, а властивість елемента меню Enabled управляє доступністю вибору цього елемента.

Викликати об'єкт таким же чином, як і підпрограму неможливо. Для зміни одного з властивостей об'єкта або для виконання одного з його методів необхідно послати об'єкту повідомлення. Доступ до властивостей і методів, що містяться в об'єкті, можна отримати, вказавши ім'я цієї властивості або методу після імені об'єкта. Ім'я об'єкта відділяється від імені властивості або методу точкою. Наприклад, якщо Power є об'єктом, то доступ до його властивості Value забезпечить наступна конструкція:

Power.Value

Якщо продовжити цю конструкцію знаком рівності (=) н числом, то об'єкт змінить значення вказаної властивості. Щоб набути значення цієї властивості, вмістіть цю конструкцію праворуч від знаку рівності.

Коли виконується метод, він може змінити значення лише властивості даного об'єкта, але не інших об'єктів. Метод може тільки запитати який-небудь об'єкт про зміну деякої його властивості.

Розглянемо як приклад об'єкта кнопку включення живлення на моніторі вашого комп'ютера (може бути у вас замість кнопки вимикач, але в цьому випадку це не має значення і можна уявити собі, що це кнопка). Коли ви натискаєте цю кнопку, внутрішній механічний з'єднувач проводить включення, приєднуючи вхід до виходу. Також абсолютно не важливо, який вигляд вимикача використаний В вашому моніторі (відомо, що існує велика кількість різних видів вимикачів). І те, як працює ваш вимикач, і його пристрій вас не повинні цікавити (пластикова оболонка вимикача оберігає його від вашої цікавості). Важливе лише те, чи сполучені вхід і вихід перемикача н залежності від стану кнопки.

Програмна реалізація такого перемикача в основному працює точно так само. Коли ви вибираєте його за допомогою миші, він міняє свій стан з "Включений" на "Вимкнений" або навпаки. Програмний перемикач не забезпечує механічного з'єднання, але володіє властивістю, що іменується Value, яке, приймає значення "Включений" або "Вимкнений". Обидва вимикачі призначені для одних і тих же цілей.

Натискаючи на кнопку за допомогою покажчика миші, ви змінюєте значення властивості Value з "Включений" на "Вимкнений" або навпаки. Крім того змінюється зовнішній вигляд кнопки, імітуючи таким чином натиснення.,

Саме натиснення на кнопку нічого не включає (рівно як і натиснення вимикача монітора). Вимикач на вашому моніторі сполучений шнуром з джерелом живлення. Механічний перемикач просто з'єднує вхід і вихід. Програмна кнопка робить те ж саме. Вона не встановлює ваш віртуальний монітор в стан "Включений" або "Вимкнений", а лише змінює властивість Value на "Включений" або "Вимкнений". Інші об'єкти, наприклад об'єкт "Віртуальний екран", повинні перевірити стан вказаної властивості, щоб визначити, включатися чи ні.

# Успадкування

Точно так само, як об'єкт-вимикач є частиною об'єкта-монітора, програмний об'єкт може бути частиною іншого, більшого програмного об'єкта. Коли об'єкти об'єднані, виявляються спеціальні властивості, відомі під назвою "успадкування". Коли один об'єкт є частиною іншого об'єкта, він успадковує всі властивості і методи попереднього, замість дозволяючи доступ до своїх властивостей і методів. Цей принцип властивий і механічному перемикачу (будучи частиною монітора, він за допомогою натиснення кнопки передає монітору по спадщині свою властивість "Включений "/"Вь1ключеи" і метод зміни цієї властивості).

Для доступу до властивостей і методів об'єкта, що є складовою частиною інших, більш великих об'єктів, треба визначити кожний із збірних об'єктів починаючи з самого крайнього контейнера, а потім поставити точку і указати наступний (внутрішній) складовий об'єкт. Потім знову точка і наступний внутрішній об'єкт, і так доти, поки не дійдете до об'єкта, до властивостей або методів якого ви намагаєтеся добратися. Таким чином, вийшов список об'єктів, вказаних зліва направо в наступному порядку: від самого крайнього (головного) і аж до цікавлячого об'єкта. Завершують цю конструкцію точка і ім'я властивості або методу, до якого ви хочете отримати доступ. Наприклад, об'єкт-монітор, що іменується MyMonitor, включає об'єкт-кнопку Power, отже, MyMonitor успадковує властивість Value об'єкта Power. Для доступу до властивості Value використовується наступна конструкція:

MyMonitor.Power.Value

Якщо MyMonitor міститься в об'єкті MyHouse, ваш доступ може бути оформлений наступною конструкцією:

MyHouse.MyMonitor.Power.Value

 І так можна продовжувати для кожного включеного об'єкта. Об'єкт MyHouse **тепер** успадкував можливості кнопки.

# Класи

Кожний об'єкт специфічний і унікальний. Клас же описує всі об'єкти певного типу. Таким чином класи є більш високим рівнем визначення об'єкта. Ви використовуєте клас для створення об'єктів. Наприклад, клас Button ви Використовуєте для створення специфічних об'єктів Button. Кожний об'єкт Button, що створюється вами, відрізняється від всіх інших об'єктів Button. Кнопка Poweri є об'єктом класу Button, так само, як і кнопка Power2. Хоч і Poweri, і Power2 належать класу Button, це різні об'єкти.

# Пошук об'єктів

Пошук об'єктів в Visual Basic часто не просте заняття, оскільки програми ца Visual Basic можуть об'єднуватися з декількома прикладними програмами, кожна з яких має власну бібліотеку об'єктів.

# Інтерактивна допомога

Використання інтерактивної підказки навряд чи не самий легкий спосіб виявити об'єкт. Вікно Help не тільки містить перелік об'єктів, але і розкаже **вам,** що вони собою представляють і як їх використати. Інтерактивна підказка викликається, як і будь-яка прикладна програма під Windows, отже ви можете відкрити декілька файлів Help одночасно. Додатково, коли відображається тема, що підказується, натисненням командної кнопки On Top ви можете зберегти вікно Help у верхній частині вашої таблиці, щоб потім читати його в процесі роботи.

Для доступу до інтерактивної підказки Excel задайте директиву Help/Contents. З'явиться вікно Help, в якому Visual Basic відведений окремий розділ "Programming with Visual Basic". Щоб ознайомитися з вмістом цього розділу (мал. 2.1), укажіть "Programming with Visual Basic". Перелік об'єктів Excel, доступних в Visual Basic, приведений в підрозділі "Programming Language Summary".

# Доступ до об'єктів Excel з Visual Basic

Excel фірми Microsoft є бібліотекою об'єктів, які знаходяться всередині Врікладних програм Excel. Як ви і передбачаєте, об'єктами Excel є таблиці, робочі папки, діаграми і області осередків. Бібліотеки функцій таблиць Ekcel, меню і піктографічне меню також є об'єктами. Visual Basic for Applications являє собою метод, що модифікує операції Excel.

# Використання наборів об'єктів

Для доступу до більшості об'єктів в Visual Basic використовуються набори. Набір це контейнер для всіх об'єктів певного класу. Наприклад, набір Workbooks об'єднує все поточні відкриті робочі папки, а набір Worksheets **всі** таблиці у відкритих робочих папках. Оскільки Visual Basic нс може отримати доступ до робочих папок або таблиць на ім'я, необхідно використати набори для приєднання імені об'єкта. Нижче перераховані декілька що найчастіше використовуються наборів:

Sheets Всі сторінки будь-яких типів в робочій папці

Workbooks Все поточні відкриті робочі папки

Worksheets Всі таблиці в робочій папці

Charts Всі сторінки діаграм в робочій книзі

Отримати доступ до будь-якого компонента набору можна двома шляхами: по номеру **або** на ім'я об'єкта. Наприклад, якщо у вас є робоча нанка з ім'ям Sales, доступ до таблиць цієї робочої папки забезпечують наступні оператори:

Workbooks( "Sales").Worksheets(2)

Workbooks( "Seles").Worksheets( "West")

Якщо West друга таблиця в наборі Worksheets, то обидві приведені посилання вказують на одну і ту ж сторінку. Якщо при доступі до набору використати точне ім'я об'єкта, то буде забезпечений доступ саме до цього об'єкта при кожному зверненні до набору. Таким чином, раніше, ніж звернутися до набору, необхідно знати ім'я об'єкта.

Використовуючи порядковий номер, можна циклічно опитувати всі компоненти набору шляхом простої зміни значення номера, і тоді немає необхідності знати, як називається об'єкт, щоб отримати до нього доступ.

І нарешті, якщо не вказувати конкретний об'єкт в наборі, вважається, що вам необхідний доступ до всього набору. У цьому випадку ви можете змінити деякі глобальні властивості об'єктів набору, додати або видалити деякі об'єкти.

Для доступу до компонентів набору дуже зручно використати оператор For Each мови Visual Basic. Оператор For Each повторює блок операторів для кожного компонента набору.

# Що таке типи даних?

Тип даних визначає, яким чином величина зберігається в пам'яті комп'ютера. Задаючи тип даних, ми також накладаємо деякі обмеження на величину, що зберігається, а саме: як велика вона можливо, скільки розрядів може містити і чи може вона мати дробову частину. Існує декілька способів розміщення числа в фіксованій кількості байтів пам'яті. Наприклад, ви можете вибрати тип даних, який використовує всі банти як просте ціле число, що дозволить отримувати числа з максимальним дозволом (тобто з найбільшою кількістю розрядів) Проте, цілі числа не можуть мати дробової частини і не можуть являти собою дуже великі величини. Щоб усунути це обмеження, необхідно використати деякі байти для зберігання інформації про розташування десятеричної точки, а інші для зберігання розрядів числа, що означають. Таким чином, ради зберігання дробових чисел і великих величин доводиться жертвувати деякої що дозволяє здатністю. Можна було б використати більше байтів для зберігання числа з тим же дозволом і показником міри, але це зажадає більше пам'яті

Визначаючи тип даних для величини, ви визначаєте також співвідношення між розрядністю, розміром і пам'яттю, що використовується.