В. П. Семеренко

ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

Міністерство освіти і науки України
Вінницький національний технічний університет

В. П. Семеренко

ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

Навчальний посібник

Вінниця
ВНТУ
2010

УДК 681.3.06.(075)
ББК 32.973.26-018.1
С32

Рекомендовано до друку Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 10 від 28.05.2009)

Рецензенти:
В. А. Лужецький, доктор технічних наук, професор
С. В. Юхимчук, доктор технічних наук, професор
О. О. Коваленко, кандидат технічних наук, доцент

Семеренко, В. П.

С32 Візуальне програмування : навчальний посібник / В. П. Семеренко. – Вінниця : ВНТУ, 2010. – 113 с.

Навчальний посібник присвячено вивченню мови програмування Visual Basic з використанням пакета Microsoft Visual Studio 2005/2008. Основну увагу приділено технології розробки візуального інтерфейсу для програм Windows Forms. Детально викладені питання роботи з векторною і растровою графікою з використанням класів GDI+, роботи з файлами і потоками введення-виведення, роботи з базами даних в ADO.NET. Посібник призначений для студентів напряму підготовки "Комп'ютерна інженерія" для вивчення дисциплін "Візуальне програмування", "Паралельне програмування", а також може бути рекомендований студентам інших спеціальностей, пов'язаних з вивченням сучасного програмного забезпечення, зокрема для напряму підготовки "Інформаційна безпека".

УДК 681.3.06.(075)
ББК 32.973.26-018.1

ЗМІСТ

	Вступ	5
	Знайомство з середовищем розробки програм *Visual Studio 2008*	7
В1	Основні поняття про проекти *Visual Studio 2008*	7
В2	Проекти типу *Console Application* та *Windows Form Application*	9
В3	Виконання та налагоджування програми	11
	Тема №1 – Основи програмування мовою *Visual Basic*	13
1.1	Ієрархія програм на *Visual Basic*	13
1.2	Типи даних	13
1.3	Змінні та константи	15
1.4	Операції і математичні методи	17
1.5	Підпрограми та функції	19
1.6	Масиви	20
1.7	Організація введення та виведення даних	21
1.8	Організація розгалужень у програмі	23
1.9	Організація циклів у програмі	25
1.10	Оформлення програмного коду	26
	Порядок виконання роботи	27
	Контрольні запитання	27
	Тема № 2 – Базові елементи інтерфейсу	28
2.1	Загальні відомості про інтерфейс	28
2.2	Типи інтерфейсів	28
2.3	Батьківське і дочірні вікна *MDI*-інтерфейсу	29
2.4	Меню	31
2.5	Контекстне меню	32
2.6	Панель інструментів	32
2.7	Рядок стану	33
2.8	Інтерфейс типу провідника	34
	Порядок виконання роботи	35
	Контрольні запитання	36
	Тема № 3 – Основні елементи керування	37
3.1	Елементи керування в середовищі *Windows*	37
3.2	Елементи керування для введення і виведення даних	39
3.3	Елементи вибору варіантів	41
3.4	Робота з кількома формами	47
	Порядок виконання роботи	49
	Контрольні запитання	49
	Тема № 4 – Робота з графікою	50
4.1	Класи для роботи з графікою	50
4.2	Графічна система координат	50
4.3	Структури простору імен *System.Drawing*	51

4.4 Створення ліній ... 52
4.5 Створення базових контурних фігур ... 53
4.6 Створення складних контурних фігур ... 55
4.7 Зафарбовування фігур ... 57
4.8 Виведення тексту ... 59
4.9 Виведення растрових зображень ... 61
Порядок виконання роботи ... 62
Контрольні запитання ... 62
Тема № 5 – Робота з файлами і потоками введення-виведення 63
5.1 Файли і потоки введення-виведення у *Visual Basic* ... 63
5.2 Основні операції з текстовими файлами за допомогою класів ... 63
5.3 Операції з файлами за допомогою потоків введення-виведення 67
5.4 Діалогові вікна для пошуку файлів ... 71
5.5 Робота з каталогами ... 72
5.6 Використання об'єкта *My* для роботи з файлами і каталогами 74
Порядок виконання роботи ... 75
Контрольні запитання ... 75
Тема № 6 – Робота з базами даних ... 76
6.1 Нова технологія роботи з даними у *Visual Studio* ... 76
6.2 Подання даних в *ADO.NET* ... 77
6.3 Об'єктна модель *ADO.NET* ... 78
6.4 Створення запитів до БД за допомогою мови *SQL* ... 80
6.5 Програмування БД на основі провайдера *OLE DB* ... 82
Порядок виконання роботи ... 84
Контрольні запитання ... 93
Тема № 7 – Багатопотокове програмування мовою *Visual Basic* ... 98
7.1 Поняття багатопотоковості ... 94
7.2 Створення нового потоку виконання ... 95
7.3 Пріоритети потоків ... 96
7.4 Основні та фонові потоки ... 98
7.5 Стани потоків ... 99
7.6 Синхронізація паралельних потоків ... 101
7.7 Використання механізму блокувань в задачах синхронізації ... 103
7.8 Використання класу *Monitor* в задачах синхронізації ... 105
7.9 Асинхронні делегати ... 106
7.10 Пули потоків ... 108
Порядок виконання роботи ... 109
Контрольні запитання ... 109
Глосарій ... 110
Література ... 112

Вступ

Мова програмування *Basic* була створена ще на зорі комп'ютерної техніки і використовувалась як навчальна мова з подальшим переходом до більш складного програмного забезпечення. Багато мов програмування і операційних систем, які з'явились навіть пізніше *Basic*, вже давно стали надбанням історії. Причиною такого довгого існування цієї мови програмування є те, що вона постійно розвивається, не відстаючи ні в чому від сучасних комп'ютерних технологій. З широким використанням системи *Windows* мова *Basic* отримала можливості для створення графічного інтерфейсу і стала першою мовою візуального програмування – *Visual Basic*.

Сьогодні версія *Visual Basic 2008* дає можливість розв'язувати будь-які сучасні задачі розробки програмного забезпечення: бізнес-програми, ігри, *Internet*-програми, бази даних. З тієї самої першої версії мови залишилася, мабуть, тільки її назва, всі інші складові зазнали суттєвих змін. В наш час *Visual Basic* вже не вважається навчальною мовою шкільного рівня, а знання самої мови та її діалектів (*VBA, VBSript*) стає необхідністю для програміста будь-якого рівня.

Разом з мовами *C++* та *C#* мова *Visual Basic* входить до трійки найсучасніших мов програмування, які є основою найпотужнішого програмного пакета компанії *Microsoft – Visual Studio 2008*. В основу цього пакета покладена технологія *NET*, яка є подальшим розвитком попередньої концепції програмування, що базувалася на *COM*-об'єктах (*Visual Studio 6.0*).

Технологія NET базується на загальномовному середовищі виконання (*Common Language Runtime – CLR*), в якому виконуються прикладні програми та набори бібліотек, які іменують бібліотекою класів *NET Framework*. Ця бібліотека класів містить декілька тисяч класів, згрупованих за ієрархічним принципом в так звані "простори імен" – *namespaces*. *NET Framework* версії 2.0, яка реалізована в пакеті *Visual Studio 2005*, підтримує 4 мови програмування (*C++, C#, Visual Basic, Java*), а *NET Framework* версії 3.0 пакета *Visual Studio 2008* підтримує три мови програмування (*C++, C#, Visual Basic*).

Багатомовність базується на використанні проміжної мови програмування (*Microsoft Intermediate language – MSIL*), в яку компілюється початковий код високорівневої мови програмування. Код цієї проміжної мови при виконанні програми відображається на машинний код за допомогою оперативного компілятора (*just-in-time – JIT*).

Однією з головних задач сучасного програмного забезпечення є ефективне керування даними. У *Visual Studio 2008* реалізована нова модель доступу *ADO.NET* (*ActiveX Data Object.NET*). Технологія *ADO.NET* являє собою набір класів, які забезпечують високу продуктивність і масштабування, можливість керувати даними від різних джерел даних.

Такими джерелами даних можуть бути бази даних, *Web*-сервіси, об'єкти, що визначаються користувачем.

Пакет *Visual Studio 2008* можно з успіхом використовувати також для створення Internet-програм на основі технології *ASP.NET*. Використовуючи новий візуальний інструмент *Visual Web Developer* можна створювати власні *Web*-сторінки і *Web*-сайти.

Для *Windows*-програм, орієнтованих на *CLR*, за основу графічного інтерфейсу користувача взято *Windows Forms*. Подібно іншим середовищам швидкого розроблення програм (*C++ Builder, Delphi*) *Windows Forms* надає всі можливості для візуального програмування – меню, панель інструментів, дочірні вікна, великий вибір елементів керування (компонентів). Перевагою *Windows Forms* є забезпечення максимальної сумісності мов програмування *C++, C#* та *Visual Basic* в задачах створення графічного інтерфейсу. *Windows*-програми для цих мов програмування, які використовують однакові класи *NET Framework*, матимуть лише незначні синтаксичні відмінності. Можна інтегрувати в одному проекті окремі модулі, написані на різних мовах програмування. Наприклад, створити клас мовою *Visual Basic*, а потім похідний від нього клас мовою C#. Тому, засвоївши будь-яку мову із цієї чудової трійки, можна легко перейти до програмування іншою мовою.

Візуальне програмування – одне з останніх досягнень сучасного програмування. Його ідея полягає в максимальному використанні готових блоків, причому не блоків програмного коду, як у об'єктно-орієнтованому програмуванні, а вже готових виконуючих блоків (графічних елементів керування – компонентів). Створення візуальної програми можна порівняти з крупноблоковою технологією будівництва будинку. Спочатку є деяка вільна площина (форма), на яку програміст перетягає готові інтерфейсні елементи: кнопки, перемикачі, списки, елементи графіки, меню, діалогові вікна ("будівельні блоки"). Після створення "скелета" програми ("каркасу будинку") необхідно написати невеликий програмний код, який буде виконуватись після заранні визначених подій: натиснення кнопки, вибору пункту меню тощо. *Windows*-програми – це програми, які керуються подіями, тобто програми постійно знаходяться в очікуванні того, що станеться якась подія, для якої вже заготовлено деякий програмний код – обробник події.

Візуальне програмування хоча і значно зменшує, спрощує процес підготовки програм, але не відміняє необхідності високого професіоналізму програміста. Основні зусилля сучасного програміста зміщуються із рутини налагоджування складного програмного коду в творчу сферу програмування і навчальний посібник має на меті допомогти в цьому.

Знайомство з середовищем розробки програм Visual Studio 2008

В1 Основні поняття про проекти *Visual Studio 2008*

Visual Basic 2008 є складовою частиною програмного пакета *Visual Studio 2008 (VS 2008).*

VS 2008 – це інструмент для швидкого розроблення програм, який дозволяє редагувати програми, виконувати і налагоджувати їх в операційній системі *Windows*.

VS 2008 існує у вигляді різних редакцій. Для програм, які розглядаються в навчальному посібнику можна використати будь-яку редакцію: від *VS 2008 Express Edition*, яка має мінімальні можливості, і до *VS 2008 Professional Edition*, яка має максимальні можливості. Для інсталяції будь-якої із редакцій *VS 2008* знадобиться процесор з тактовою частотою не нижче 1 ГГц, мінімум 256 Мбайт оперативної памяті та декілька гігабайтів вільної дискової памяті.

Підготовка текстів програм, їх компіляція, налагоджування та виконання здійснюється у інтегрованому середовищі розробки (*Integrated Development Environment – IDE*). Це середовище надає однакові можливості для підготовки програм для всіх мов програмування (*C++, C#, Visual Basic*) і включає багато інтегрованих складових, зокрема головне меню, панель інструментів (*Toolbars*), конструктор форм (*Designer*), оглядач розв'язків (*Solution Explorer*), панелі елементів керування (*Toolbox*). Якщо потрібна складова частина *IDE* невидима, тоді достатньо викликати пункт *View* головного меню і далі вибрати цю складову. Точний розмір і форма вікон та інструментів залежать від того, як було налаштоване середовище розробки конкретного користувача. У *VS 2008* підтримується багато нових можливостей для роботи з вікнами: прикріплення (*dock*), автозбереження та інші.

Всі програми, які створюються у *VS 2008*, організовуються у проекти, а проекти об'єднуються у розв'язки. Проект містить велику кількість файлів різного типу: початкові, проміжні і кінцеві (виконувані). Існує також спеціальний файл проекту (*vbproj*) і файл розв'язку (*sin*), які містять службову інформацію відповідно про склад проекту та склад розв'язку. Корисно знати, що піктограма файла розв'язку містить маленьку цифру, яка вказує на номер версії програмного пакета (цифра 8 відповідає *VS 2005*, а цифра 9 – *VS 2008*). Візуально структуру проекту можна побачити у діалоговому вікні *Solution Esplorer*. В цьому вікні можна вибрати будь-яку частину програми і відобразити її у вікні редактора коду.

На рис. 1 показане головне вікно *VS 2008*, яке з'являється відразу після запуску. На ньому можна виділити декілька основних об'єктів: головне меню, стандартну панель інструментів, початкову сторінку (*Start Page*).

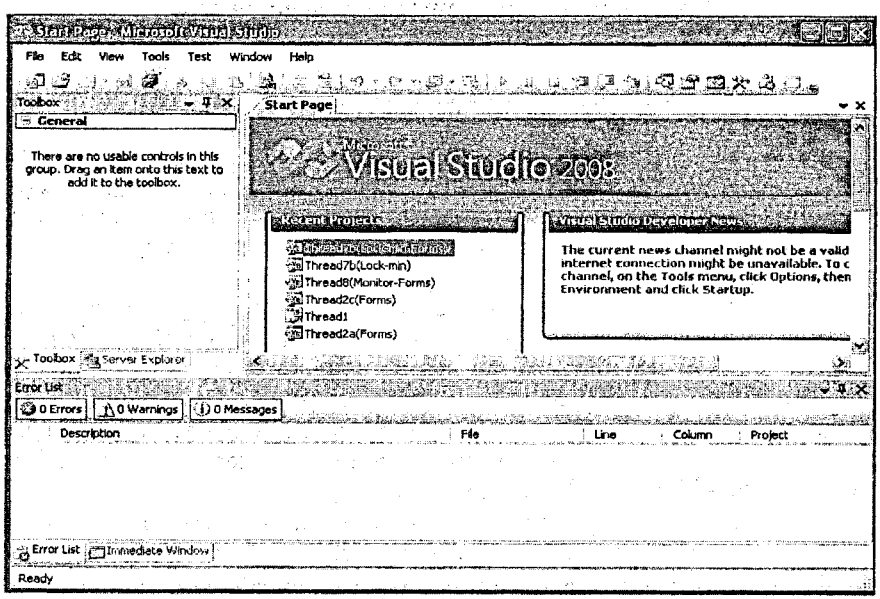

Рисунок 1 – Головне вікно *VS 2008*

В діалоговому вікні початкової сторінки показані назви декількох проектів, які відкривалися останнім часом. Для створення нового проекту необхідно в меню *File* вибрати послідовно пункти *New* та *Project*.

VS 2008 підтримує різні види проектів. Розглянемо деякі основні проекти, які можна створити для мови програмування *Visual Basic* (рис. 2):

Console Application (Консольна програма) – програма, яка не використовує графічні інтерфейси і являє собою окремий виконуваний файл;

Windows Forms Application (*Windows*-програма) – програма, яка підтримує традиційний *Windows*-інтерфейс і використовує форми *Windows* з елементами керування;

Class Library (Бібліотека класів) – клас або компонент, який може входити до складу інших програм;

Windows Forms Control Library (Бібліотека елементів керування *Windows*) – є аналогом *ActiveX Controls* у *Visual Basic 6*;

Empty Project (Пустий проект) – створюється проект, який містить лише необхідну для зберігання структуру файлів, все інше додається вручну;

WPF Application (*WPF*-програма) – проект для створення програм з використанням *WPF*-технології.

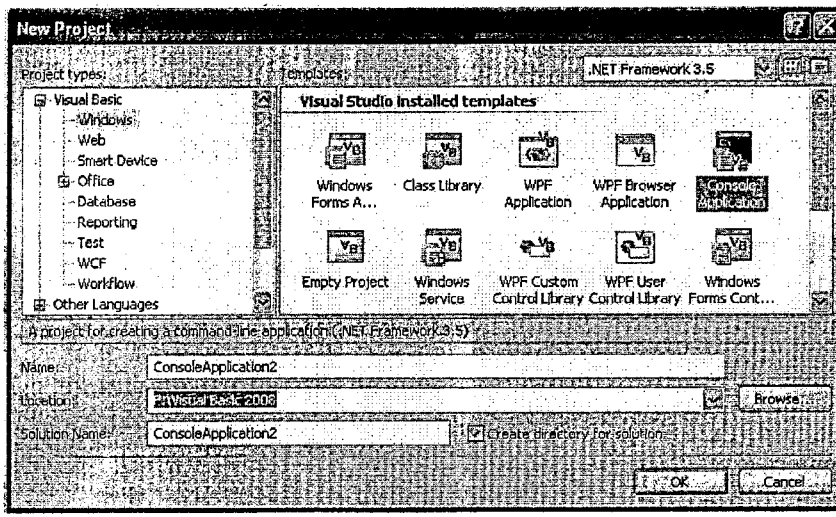

Рисунок 2 – Головне вікно *VS 2008*

Для створення нового проекту необхідно вибрати тип проекту, вказати ім'я проекту і папку, де він буде зберігатися (якщо не влаштовують ті дані, які *VS 2008* надає за замовчуванням). Після закриття вікна "*New Project*" відкривається нове вікно відповідно з вибраним типом проекту.

В2 Проекти типу *Console Application* та *Windows Form Application*

Найпростішим типом проекту є *Console Application*. Програма називається консольною, тому що взаємодія з нею відбувається через клавіатуру і дисплей, тобто використовується лише текстовий інтерфейс. На рис. 3 показане діалогове вікно, яке відповідає проекту *Console Application*. Це вікно є вікном редактора коду програми, де вже створена структура найпростішої консольної програми, яка складається з одного модуля і однієї процедури. Далі можна вводити свої дані і оператори.

Для переважної більшості програм навчального посібника передбачено використання типу проекту *Windows Form Application*. Цей тип проекту використовує всі графічні можливості *Windows*, тому є найбільш придатним для вивчення візуального програмування. Якщо в консольній програмі можна безпосередньо працювати з клавіатурою і дисплеєм, то для *Windows*-програм забороняється безпосередній доступ до апаратних засобів комп'ютера, операції введення-виведення здійснюються тільки через спеціальні функції операційної системи – функції *API*

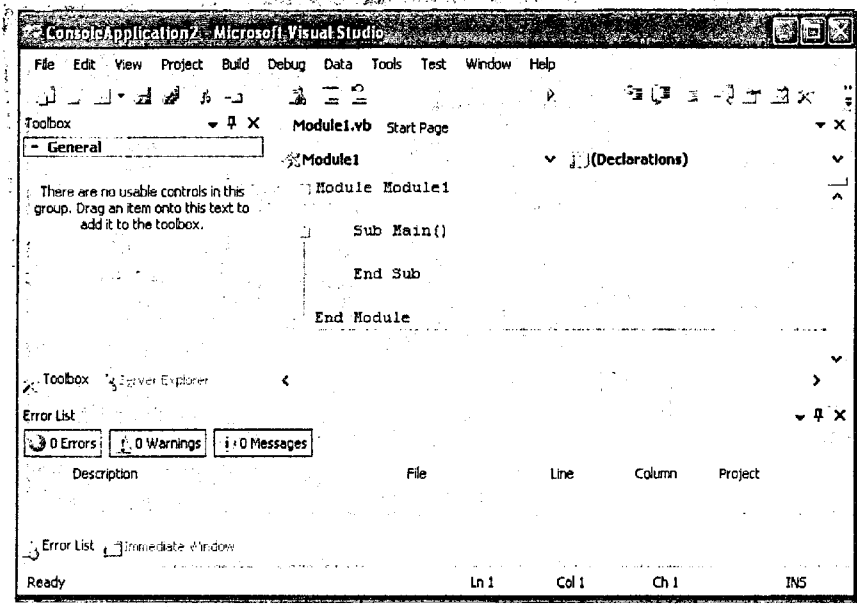

Рисунок 3 – Вікно проекту *Console Application*

(*Application Program Interface*). Однак в цьому випадку можна забезпечити багатозадачний режим роботи, коли кожна програма, системна чи прикладна, працює в своєму вікні. А головне, можна скористатись всіма видами графіки: векторної, растрової, тривимірної.

Проект *Windows Form Application* забезпечує графічний інтерфейс *Windows Forms,* тобто інтерфейс, який виконується в середовищі *CLR.* Базовим елементом цього інтерфейсу є форма – спеціальне вікно, на яке переміщаються елементи керування (ЕК), які забезпечують зручний інтерфейс з користувачем. Пакет *VS 2008* надає кілька десятків ЕК, які знаходяться на панелі елементів керування (*Toolbox*).

На рис. 4 показане діалогове вікно, яке відповідає проекту *Windows Form Application.* Після створення такого проекту подальше розроблення програми зводиться до таких дій.

1. Інтерактивне створення графічного інтерфейсу користувача на вкладці *Form1.vb [Design]* (конструктор форми) вибором ЕК у вікні *Toolbox* і переміщенням їх на форму.

2. Зміна властивостей форми та ЕК у вікні *Properties* (властивості).

2. Написанням програмного коду обробників подій для визначених ЕК. Підготовка програмного коду здійснюється у вікні редактора коду, в яке можна зайти при подвійному натисненні лівої клавіші миші на відповідному ЕК в конструкторі форми.

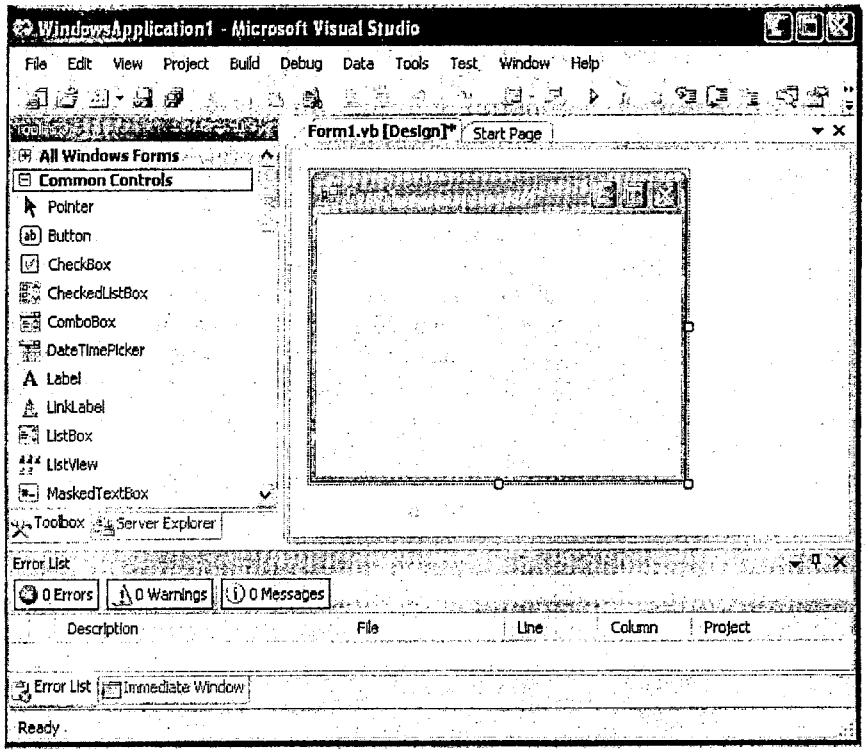

Рисунок 4 – Вікно проекту *Windows Form Application*

В3 Виконання та налагоджування програми

Перед тим, як виконати програму, її спочатку необхідно відкомпілювати. Оскільки програма є складовою частиною проекту, потрібно відкомпілювати весь проект. Цей процес, який також називають ще Збіркою (*Building*) проекту, виконується вибором пункту *Build Solution* в меню *Build* (можна також натиснути клавішу *F7*). В результаті початковий файл програми буде скомпільовано у виконуваний файл.

Запустити програму на виконання можна одним із трьох способів: в меню Відлагодження (*Debug*) вибрати пункт *Start Debugging,* натиснути на однойменну кнопку в панелі інструментів або натиснути клавішу *F5*. Якщо відразу запустити на виконання тільки що набрану програму, то її попередня компіляція виконається автоматично.

Звичайно, не завжди вдається з першого разу отримати потрібні результати. Найчастіше після компіляції в нижній частині вікна редактора коду програми ми отримуємо список синтаксичних помилок програми. Якщо синтаксична помилка була виявлена не в результаті запуску на

компіляцію, а після запуску на виконання, блокується вся подальша робота з програмою. Для зняття блокування необхідно вибрати в меню *Debug* пункт *Stop Debugging*. Тільки після цього можна продовжувати налагоджування і подальші запуски програми на виконання. Тому і рекомендується спочатку відкомпілювати програму, а потім запускати її на виконання.

Однак програма може видавати неправильні результати і при логічних помилках в алгоритмі програми. В таких випадках рекомендується виконати програму в покроковому режимі з аналізом проміжних значень всіх змінних. Покроковий режим запускається, якщо вибрати в меню *Debug* пункт *Step Into* або натиснути клавішу *F11*. При кожному подальшому натисненні клавіші *F10* виконується один оператор програми. Одночасно в нижній частині вікна редактора коду програми з'являється вікно *Locals* (локальні змінні), в якому відображаються поточні значення змінних програми (рис. 5). Можна не тільки переглядати значення змінних, але і змінювати їх. Варто звернути увагу, що глобальні змінні переглянути в покроковому режимі не можна.

VS 2008 надає також і інші засоби для налагоджування програм.

Розглянуті засоби компіляції та налагоджування програм у пакеті *Visual Studio 2008* практично не відрізняються від аналогічних засобів пакета *Visual Studio 2005.*

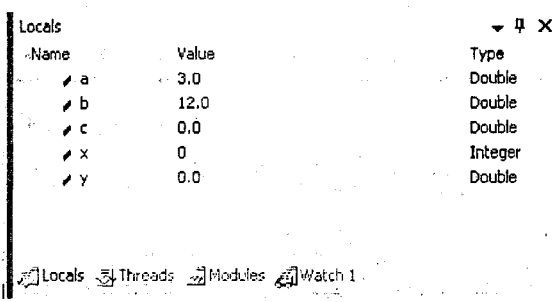

Рисунок 5 – Вікно *Locals*

ТЕМА № 1

Основи програмування мовою Visual Basic

Зміст теми. Знайомство з основними елементами мови програмування *Visual Basic*: типами даних, змінними, константами, операціями та базовими операторами. Показані особливості написання найпростіших програм з розгалуженнями, циклами, введенням та виведенням даних.

Теоретичні відомості

1.1 Ієрархія програм на Visual Basic

Основними блоками, з яких будується програма на *Visual Basic* є оператор, процедура, модуль, проект, рішення.

Оператор – це найменша одиниця програмного коду, яка може бути виконана. Оператор може об'являти змінну (оператори опису) або виконувати якусь дію (оператори виконання).

Процедура – це логічно закінчена частина програмного коду, яка складається з окремих операторів і до якої можна звертатись по імені. Розрізняють два види процедур: підпрограми (*Subroution*) і функції (*Function*).

Програмний код у *Visual Basic* зберігається в програмних модулях, які можуть бути трьох видів: модуль форми, стандартний модуль та модуль класу.

Проста програма, що складається з однієї форми, містить, як правило, тільки модуль форми. Модуль форми може містити об'яви змінних, констант, типів даних, процедур обробки подій.

В більш складних, стандартних, модулях підпрограми і функції, що часто повторюються, можуть бути виділені в окремий програмних код, спільний для всіх. Стандартні модулі можуть також містити об'яви глобальних змінних, зовнішніх процедур і процедур загального характеру, доступних для інших модулів даної програми.

При використанні засобів об'єктно-орієнтованого програмування створюються модулі класів.

Проект (*Project*) складається зі всіх модулів, форм і зв'язаних з програмою об'єктів. Розвязок (*Solution*) об'єднує один або декілька проектів.

1.2 Типи даних

Основні типи даних наведені в табл. 1.1.

Для збереження цілих значень використовуються типи даних *SByte, Short, Integer, Long* для чисел зі знаком та *Byte, UShort, UInteger, Ulong для* беззнакових.

Таблиця 1.1 – Типи даних *Visual Basic*

Тип даних	**Що зберігає**	**Займає в пам'яті**
Boolean	*True, False*	Довільно
Char	1 символ в *Unicode*	2 байт
String	Текст, до 2 млрд. символів в *Unicode*	Довільно
Date	Дата від 01/01/0001 до 12/31/9999 і час від 0:00:00 до 23:59:59	8 байт
Byte	Дуже малі беззнакові двійкові числа від 0 до 255	1 байт
SByte	Дуже малі двійкові числа зі знаком від -128 до 127	1 байт
Short	Малі цілі числа зі знаком від -32 268 до 32 767	2 байт
UShort	Малі беззнакові цілі числа від 0 до 65 535	2 байт
Integer	Цілі числа зі знаком від -2 147 483 648 до 2 147 483 647	4 байт
UInteger	Беззнакові цілі числа від 0 до 4 294 967 295	4 байт
Long	Великі цілі числа зі знаком від -9 223 372 036 854 775 808 до 9 223 372 036 854 775 808	8 байт
ULong	Беззнакові великі цілі числа від 0 до 18 446 744 073 709 551 615	8 байт
Single	Число з плаваючою комою від -3,4028235E38 до -1,401298E-45 для від'ємних значень і від 1,401298E45 до 3,4028235E38 для додатних значень	4 байт
Double	Число з плаваючою комою подвійної точності Від -1,79769313486231570E308 до 4,94065645841246544E-324 для від'ємних значень і від 4,94065645841246544E-324 до 1,79769313486231570E308 для додатних значень	8 байт
Decimal	Число з фіксованою комою 29 розрядів, із них 28 розрядів для дробової частини числа	16 байт
Object	Посилання на об'єкт	Довільно

Змінні типів *Short, UShort* займають менший обсяг памяті, однак обчислення формул, які містять типи *Integer*, відбуваються швидше, ніж формул з даними інших цілих типів.

Типи даних *Single* і *Double* зберігають дробові числа X у форматі

$$X = mE^p,$$

де m – мантисса числа,

p – порядок числа,

E – основа характеристики, E=10.

Наприклад, 3,72E-5 означає число $X = 3{,}72 \times 10^{-5} = 0{,}0000372$. Ці типи даних дозволяють подавати дуже малі і дуже великі значення чисел.

Тип даних *Decimal* містить дробові числа з фіксованою комою. Цей тип даних рекомендується використовувати для складних обчислень з високою точністю, оскільки в них не виникають похибки округлення, які можуть мати місце при використанні типів даних *Single* і *Double.*

Тип даних *Date* зберігає дату у форматі "місяць/день/рік" і час у форматі "година:хвилина:секунда".

Тип даних *Object* може зберігати різні дані і змінювати їх тип під час виконання програми.

При виконанні спільних дій з різними типами даних (наприклад, в арифметичних виразах) виникає задача перетворення типів даних. Перетворення типів даних і зведення їх до одного заданого типу здійснюється автоматично. В складних випадках можна здійснювати перетворення типів даних явно з використанням спеціальних функцій (табл. 1.2).

Таблиця 1.2. – Функції перетворення типів даних

Фупкція	**В який тип перетворюється**
CInt()	*Integer*
CDbl()	*Double*
Cbyte()	*Byte*
CLng()	*Long*
CSng()	*Single*
CBool()	*Boolean*
CStr()	*String*
CDate()	*Date*

1.3 Змінні та константи

Під змінними і константами в мовах програмування розуміють поіменовані ділянки памяті для збереження даних, з якими працює програма. *Visual Basic* підтримує явне та неявне оголошення змінних.

Перш ніж використати змінну чи константу, її треба обов'язково описати (оголосити). Для змінних допускається тільки їх явне оголошення.

При явному оголошенні ім'я і тип змінної вказується на початку програмного коду, модуля або процедури за допомогою такого оператора:

<Область видимості> <Ім'я змінної> As <Тип даних>

Ім'я змінної повинно починатися із букви і може містити будь-які букви, цифри, символи підкреслення, але не можна використовувати пропуски і знаки пунктуації.

Область видимості визначає, із якої частини програми можна звернутись до змінної і задається такими операторами: *Public, Dim, Private, Static*. Якщо використати ключове слово *Dim* (означає *dimension* - розмір), тоді областю використання змінної буде лише та процедура або модуль, де вона була описана. Такі змінні ще називають локальними (закритими). Якщо оголошення змінної здійснюється за допомогою ключового слова *Static* (Статичний), тоді вона також буде локальною, однак після виконання процедури чи модуля, де вона була описана, її значення збережеться. Якщо на початку модуля оголосити змінну за допомогою ключового слова *Private* (Закритий), тоді вона буде доступною лише в процедурах даного модуля. Використання ключового слова *Public* (Загальнодоступний) означає, що відповідна змінна буде глобальною, тобто доступною в будь-якій частині програмного коду. Всередині процедур не можна використовувати оператори *Public* та *Private*. Розглянемо різні області видимості на такому прикладі програмного коду.

```
Module Module1
        Public a1, b1 As Integer
        Private c2  As Double
Sub Proc1()
        Dim e, f As Short
        . . .
End Sub
Sub Proc2()
        Static sum As Long
        . . .
End Sub
End Module
Module Module2
Sub Proc3()
        Dim str2 As String
        . . .
End Sub
End Module
```

В даному випадку змінні оголошені *a1, b1* як глобальні, і тому доступні у всіх трьох процедурах. Змінна *c2* буде доступною лише для процедур *Proc1() Proc2()* і закритою для процедури *Proc3()*. Змінні *e, f,*

mas, str є локальними і доступними тільки в тих процедурах, де вони оголошені.

Неявне оголошення означає, що оголошення змінної відбувається при першому її використанні у програмному коді. За замовчуванням встановлено режим явного оголошення змінних. Для зміни цього режиму потрібно на самому початку програмного коду записати

Option Explicit Off

Варто відмітити, що перевагою є явне оголошення змінної, оскільки це зменшує ймовірність появи помилок у програмі.

Ще з попередніх версій *Visual Basic* збереглася можливість оголошення змінних для найбільш поширених типів даних за допомогою суфікса. Замість ключового слова з наступним записом типу даних можна дописати до імені змінної суфікс (табл. 1.3), який вказує на тип даних.

Таблиця 1.3 – Суфікси для типів диних

Тип даних	Суфікс
Integer	%
Long	&
Single	!
Double	#
String	$

1.4 Математичні операції та методи

У *Visual Basic* операція записується у вигляді спеціального символу або ключового слова у виразі, який об'єднує два оператори. У табл. 1.4 наведені логічні операції, у табл. 1.5 – математичні операції, у табл. 1.6 – операції відношень.

Якщо в одному виразі міститься кілька математичних операцій, тоді вони обчислюються згідно з установленими пріоритетами (табл. 1.7).

У *Visual Basic 2005* була включена нова бібліотека класів *NET Framework* 2.0. В цій бібліотеці є також клас *System.Math*, методи якого дозволяють виконувати складні математичні обчислення. Для підключення класу *System.Math* необхідно на самому початку вікна редактора коду помістити оператор *Imports System.Math.*

Таблиця 1.4 – Логічні операції

Операція	Опис
And	Логічне множення (ТА)
Or	Логічне додавання (АБО)
Xor	Виключне АБО
Not	Логічне заперечення (НІ)

Таблиця 1.5 – Математичні операції

Операція	Опис
+	Додавання
–	Віднімання
*	Множення
/	Ділення
\	Цілочислене ділення
Mod	Остача від ділення
^	Піднесення до степеня
&	Конкатенація (об'єднання) рядків

Таблиця 1.6 – Операції відношень

Операція	Опис
<	Менше
>	Більше
<=	Менше або дорівнює
>=	Більше або дорівнює
<>	Не дорівнює
=	Дорівнює

Таблиця 1.7 – Пріоритети операцій

Порядок пріоритету	Операції
1	( ) значення в круглих дужках
2	^ піднесення до степеня
3	- зміна знака числа
4	* / множення, ділення
5	\ цілочислене ділення
6	Mod остача від ділення
7	+ – додавання, віднімання

В табл. 1.8 містяться деякі найбільш поширені методи цього класу. Аргумент *n* в наведених методах означає число, змінну чи оператор, який необхідно передати в метод для відповідної обробки.

Таблиця 1.8 – Математичні методи

Метод	Призначення
Abs(n)	Повертає абсолютне значення числа *n*
Atan(n)	Повертає арктангенс числа *n* в радіанах
Cos(n)	Повертає косинус кута *n*, який задається в радіанах
Sin(n)	Повертає синус кута *n*, який задається в радіанах
Exp(n)	Повертає константу, піднесену до степеня *n*
Sign(n)	Повертає –1, якщо *n<0*, 0, якщо *n=0*, +1, якщо *n>0*
Sqrt(n)	Повертає квадратний корінь із числа *n*
Tan(n)	Повертає тангенс кута *n*, який задається в радіанах

1.5 Підпрограми та функції

У програмах часто виникає необхідність багатократного виконання однакової послідовності операторів. В цьому випадку рекомендується таку групу операторів оформити у вигляді процедури і викликати її кожного разу по імені процедури.

Як вже відзначалося раніше, у *Visual Basic* є два види процедур: підпрограми (*Subroution*) і функції (*Function*).

Базовий синтаксис підпрограми має вигляд

```
Sub <SubroutineName> ([аргументи])
    Оператори процедури
End Sub
```

Програміст повинен записати ім'я підпрограми (*SubroutineName*) і, при необхідності, передати у підпрограму аргументи, обов'язково вказавши їх тип. За замовчуванням *Visual Basic* додає до кожного аргументу ключове слово *ByVal*, яке свідчить про передачу аргументів за значенням. Тобто, по суті передається лише копія аргументу і всі зміни значень аргументів не будуть назад передані в основний програмний код. Процедура також не повертає ніяких результатів своєї роботи. Розглянемо приклад підпрограми *Sum*, яка визначає суму двох чисел.

```
Sub Sum (a,b As Integer)
    Dim y As Integer
    y=a+b
End Sub
```

Базовий синтаксис функції має вигляд

```
Function <FunctionName> ([аргументи]) As Type
    Оператори функції
   [Return значення]
End Function
```

Основна відмінність функції від підпрограми полягає в тому, що функція повертає результат своєї роботи одним із двох способів: за допомогою необов'язкового оператора *Return* або через своє ім'я (*FunctionName*). Саме тому потрібно обов'язково в заголовку функції вказувати тип повертаного значення. Розглянемо приклад функції *Sum*, яка визначає суму двох чисел.

```
Function Sum (a,b As Integer) As Integer
    Dim y As Integer
    y=a+b
    Return y
End Function
```

Ще одна відмінність функції від процедури полягає в способі її виклику: ім'я функції вказується в правій частині оператора присвоєння (наприклад, *z=Sum(a,b)*) або є частиною складного виразу.

1.6 Масиви

Масив являє собою набір змінних одного типу з одним іменем. Кожний елемент масиву має свій номер, який називається індексом. Індексація елементів масиву починається з нуля. Кількість вимірів масиву називається його рангом або розмірністю.

У *Visual Basic* існують масиви фіксованого розміру і динамічні масиви, в яких розмір може змінюватись в процесі виконання програми.

Оголошення масиву фіксованого розміру залежить від області його видимості і здійснюється таким чином:

– за допомогою *Public* оператора в секції *Declaration* (Об'ява) модуля – глобальний масив;

– за допомогою *Private* оператора в секції *Declaration* модуля – масив рівня модуля;

– за допомогою *Dim* або *Static* в тілі процедури – локальний масив;

При оголошенні одновимірного масиву після його імені в круглих дужках вказується верхня границя масиву. Приклади оголошень масиву:

Dim iMas1(10) As Integer – локальний масив із 11 елементів цілого типу;

Static iMas2%(7) – локальний масив із 8 елементів цілого типу;

Public sMas(6) As String – глобальний масив із 7 елементів типу *String*.

Масиви з кількістю вимірів більше одиниці, називаються багатовимірними. При оголошенні багатовимірного масиву після його імені в дужках через кому вказуються верхні границі індексів по кожному виміру (нижні границі починаються з нуля). Наприклад,

Dim dMas(3,5,4) As Double – локальний масив із 4·6·5 = 120 елементів дійсного типу.

Якщо заранні невідомо розмір масиву тоді можна створити динамічний масив:

Dim A() As Single
Dim A(,) As Integer.

Під час виконання програми можна змінити розмірність раніше оголошеного динамічного масиву за допомогою оператора *ReDim*. Якщо необхідно змінити розмір масиву без втрати даних, тоді необхідно скористатись оператором *ReDim* разом із ключовим словом *Preserve*. Варто пам'ятати, що у багатовимірному масиві без втрати даних можна змінити розмірність тільки останнього виміру. Наприклад, якщо масиви спочатку були описані як

Dim Mas1(9) As Integer
Public Mas2(3,3,3) As Double

то в подальшому можна перевизначити границі цих масивів таким чином:

ReDim Mas1(15)
ReDim Preserve Mas2(3,3,7).

Ініціалізацію масивів можна здійснити поелементно за допомогою оператора присвоєння. Однак зручніше зробити ініціалізацію одночасно з оголошенням масиву, наприклад:

Dim Names() As String ={"Віктор", "Тетяна", "Микола"}

Dim Counts(,) As Integer={{2,45},{12,7},{5,9}}

Для виконання стандартних операцій з масивами зручно користуватись класом *Array* із бібліотеки *NET Framework*. Найбільш корисними методами цього класу є такі методи:

– *Array.Sort()* – сортування елементів масиву;

– *Array.Reverse()* – сортування елементів масиву в зворотному порядку;

– *Array.Copy()* – копіювання масиву;

– *Array.Clear()* – очищення масиву;

– *Array.Find()t* – пошук в масиві заданого елементу.

Наприклад, для сортування масиву *Mas(5)* необхідно записати:

Array.Sort(Mas)

Дуже корисними є дві функції, які визначають границі масивів: функція *Lbound(ArrayName)* повертає нижню границю масиву з іменем *ArrayName*, а функція *Rbound(ArrayName)* повертає верхню границю цього масиву.

1.7 Організація введення та виведення даних

Ввести початкові дані у програми та вивести результати можна за допомогою різних керуючих компонентів, які будуть розглянуті в наступних темах. Однак використання візуальних компонентів не завжди доцільно, оскільки такі компоненти мають постійно знаходитись на формі. У випадку введення чи виведення разових даних краще скористатись спеціальними діалоговими вікнами, які з'являються тільки в разі потреби.

Таким діалоговим вікном для введення даних служить вікно введення *InputBox*. Виклик цього вікна реалізовано у формі звертання до функції

<змінна> =InputBox(<повідомлення>[,<заголовок>, X,Y,<help>])

Значення функції *InputBox()* передається змінній символьного типу. При натисненні кнопки *OK* ця змінна набуває значення введеного тексту у полі введення вікна. При виклику функції необхідно обов'язково вказати *<повідомлення>*, яке буде відображено в центрі діалогового вікна. Серед необов'язкових аргументів функції є *<заголовок>* вікна, координати *X* та *Y* лівого верхнього кута вікна та підказка *<help>* у полі введення. Можна ввести до 255 символів загальним обсягом до 1024 байт.

Наприклад, звертання

str=InputBox("Введіть пароль", "Аутентифікація")

зумовить виклик діалогового вікна, зображеного на рис. 1.1.

Для виведення даних служить вікно повідомлення *MessageBox*. Це вікно складається із заголовка, текста повідомлення, значка та однієї або

кількох кнопок. Можна викликати *MessageBox* як процедуру або як функцію.

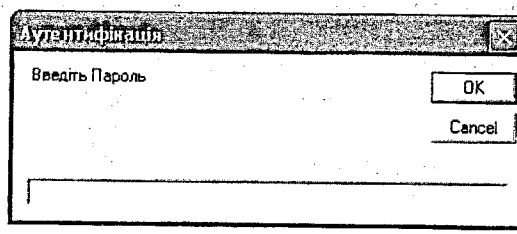

Рисунок 1.1 – Вікно *InputBox*

Формат процедури *MessageBox* має вигляд:

MsgBox(<повідомлення>[,<кнопки і значки>,<заголовок>])

Як і для функції обов'язковим для задання є лише перший аргумент *<повідомлення>*, який буде відображено як текст в центрі вікна повідомлення. В цей текст обсягом до 1024 байт для форматування можна вводити також символи переведення рядка *vbLf* та повернення каретки *vbCr*. За замовчуванням вікно повідомлення міститиме одну кнопку *OK* та заголовок, що збігається з назвою проекту. Можна задати свої кнопки і значки, в табл. 1.9 наведено набір деяких з них.

Наприклад, звертання

MsgBox("Text", MsgBoxStyle.YesNo, "Result")

зумовить виклик вікна повідомлення, зображеного на рис. 1.2.

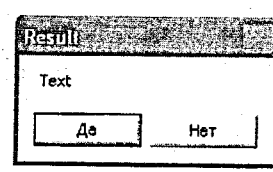

Рисунок 1.2 – Вікно повідомлення *MessageBox*

Таблиця 1.9 – Кнопки і значки в діалоговому вікні *MsgBox*

Значення аргументу	**Набір кнопок і значків в *MsgBox***
MsgBoxStyle.OkOnly	Кнопка *Ok*
MsgBoxStyle.Okancel	Кнопки *Ok, Cancel*
MsgBoxStyle.RetryCancel	Кнопки *Retry, Cancel*
MsgBoxStyle.YesNo	Кнопки *Yes, No*
MsgBoxStyle.YesNoCancel	Кнопки *Yes, No, Cancel*
AbortRetryIgnore	Кнопки *Abort, Retry, Ignore*
MsgBoxStyle.Information	Кнопка *Ok*, значок "*Information*"
MsgBoxStyle.Question	Кнопка *Ok*, значок "*?*"

Формат *MsgBox* як функції, такий же, як і процедури. Різниця полягає лише в тому, що функція повертає деяке значення, яке залежить від натиснутої кнопки (табл. 1.10). Ці значення можна використати в програмі, наприклад, для розгалуження подальших дій.

Для відображення вікна повідомлення можна також скористатися класом *MessageBox*, який є частиною простору імен *System.Windows.Forms*. Цей клас відображає вікно за допомогою методу, який використовує ті ж самі аргументи, що і функція *MsgBox*. Наприклад:

MessageBox.Show("Data", MsgBoxStyle.Ok, "Type")

Таблиця 1.10 – Значення, що повертає функція *MsgBox*

Значення аргументу	**Набір кнопок і значків в *MsgBox***
MsgBoxResult. Abort	Натиснення кнопки *Abort*
MsgBoxResult.Cancel	Натиснення кнопки *Cancel*
MsgBoxResult.Ignore	Натиснення кнопки *Ignore*
MsgBoxResult.Ok	Натиснення кнопки *Ok*
MsgBoxResult.Yes	Натиснення кнопки *Yes*
MsgBoxResult.No	Натиснення кнопки *No*
MsgBoxResult.Retry	Натиснення кнопки *Retry*

На завершення відзначимо, що вікна *InputBox* і *MessageBox* є модальними, тобто подальша роботи програми буде неможливою без закриття цих вікон в результаті натиснення відповідної кнопки.

1.8 Організація розгалужень у програмі

Для зміни порядку виконання операторів у програмі існують такі типи керуючих конструкцій:

If – якщо задана умова розгалуження може приймати два значення: *True/False*;

Select Case – якщо задана умова розгалуження є виразом, яке може приймати більше двох значень;

Try catch – використовується для коректної обробки виняткових ситуацій.

Розглянемо детальніше синтаксис умовних операторів на основі наведених керуючих конструкцій.

Умовний оператор *If ... Then* використовується в тому випадку, коли треба виконати один або декілька операторів при виконанні заданої умови, тобто, коли значення умови дорівнює *True*. Є дві різновидності цього оператора: однорядковий і багаторядковий. Однорядковий оператор має такий синтаксис:

If <умова> Then <оператори>

В одному рядку можна записати декілька операторів, розділивши їх символом двокрапки, наприклад

```
If X=0 Then A=A+5 : C=A+B
```

Якщо оператори записати по одному в окремому рядку, тоді використовується багаторядковий умовний оператор із синтаксисом:

```
If <умова> Then
   <оператори>
End If
```

Умовний оператор *If ... Then ...Else* дозволяє задати дії, які виконуються, коли значення умови дорівнює *True* (*<оператори1>*) і коли значення умови дорівнює *False* (*<оператори2>*).. Цей оператор має такий синтаксис:

```
If <умова> Then
   <оператори1>
Else
   <оператори2>
End If
```

Оператор *If* може перевірити тільки одну умову, якщо розгалуження необхідно виконати на основі декількох умов, тоді додаткові умови можна задати за допомогою оператора *ElseIf*:

```
If <умова1> Then
   <оператори1>
ElseIf <умова2>
   <оператори2>
End If
```

Нижче наведено приклад процедури з введення початкового значення цілого числа *x* та виведення результатів обчислення арифметичних виразів.

```
Imports System.Math
Module Module1
    Sub Main()
    Dim a, b, c, y As Double, x As Integer
    a = 3
    b = 12
    c = 7
    x = InputBox("Введите x")
    If x = 0 Then
        y = Sqrt(a + b)
    Else
        y = b ^ 2 / (a + c)
    End If
    MsgBox(CStr(y), MsgBoxStyle.Question, "Result")
End Sub
End Module
```

У *Visual Basic* не обмежується кількість розгалужень, створених операторами *If* та *ElseIf*. Однак при кількох однотипних умовах зручніше

користуватись перемикачем – оператором *Select Case*, який має такий синтаксис:

```
Select Case <тестовий вираз>
    Case <значення1>
        <оператори1>
    Case <значення2>
        <оператори2>
        . . .
    Case Else
        <операториN>
End Select
```

Якщо *<тестовий вираз>* приймає *<значення1>*, тоді виконуються *<оператори1>*, якщо *<значення2>* – тоді виконуються *<оператори2>* і т.д. У тому випадку, коли *<тестовий вираз>* не збігається із жодним із наведених значень, тоді виконуються *<операториN>* (їх необов'язково вказувати).

1.9 Організація циклів у програмі

Для багатократного повторення операторів у програмі передбачені такі оператори циклів.

Цикл *For ... Next* використовується в тому разі, коли кількість повторів заданного блоку операторів наперед відома. Синтаксис цього оператора:

```
For лічильник=<поч. значення> To <кін. значення> [Step <крок> ]
        <оператори1>
    [If <умова> Then Exit For]
        <оператори2>
Next
```

Аргумент *лічильник* є змінною числового типу (тип змінної можна оголосити безпосередньо в конструкції), якій спочатку присвоюється *<поч. значення>*. Після одного виконання тіла циклу (*<оператори1>* і *<оператори2>*) лічильник збільшується на значення *<крок>* (якщо значення кроку дорівнює 1, його можна не вказувати). Цикл завершується нормально, якщо значення лічильника досягає *<кін. значення>* або достроково, якщо в тілі циклу буде виконана передбачена умова в операторі

```
If <умова> Then Exit For
```

При достроковому завершенні *<оператори1>* виконаються в останній раз , а *<оператори2>* – ні.

Нижче наведено приклад процедури для знаходження суми елементів одновимірного масиву.

```
Sub Proc()
    Dim mas() As Integer = {9, 4, 2, 1, 5}
    Dim sum% = 0, i%
    For i% = 0 To UBound(mas)
            sum = sum + mas(i)
    Next i
    MsgBox(sum, MsgBoxStyle.OkOnly, "Dialog")
End Sub
```

Цикл *Do ... Loop* використовується в тому разі, коли кількість повторів заданного блоку операторів зарані невідома. Існує чотири різновиди такої конструкції:

```
1)  Do While <умова>
      <оператори>
    Loop
2)  Do
      <оператори>
    Loop While <умова>
3)  Do Until <умова>
      <оператори>
    Loop
4)  Do
      <оператори>
    Loop While <умова>
```

В перших двох конструкціях *<оператори>* виконуються до тих пір, поки значення *<умова>* дорівнює *True*. Якщо вказана *<умова>* від самого початку дорівнює *False*, тоді в першому випадку цикл не виконається жодного разу, а в другому випадку тільки один раз. Останні дві конструкції відрізняються від двох попередніх лише тим, що цикл буде виконуватись лише до тих пір, поки значення *<умова>* дорівнює *False*.

1.10 Оформлення програмного коду

Особливістю запису програмного коду, яка збереглася ще з перших версій мови *Basic* є правило запису одного оператора в новому рядку. Саме тому в цій мові немає спеціальних символів закінчення операторів.

У сучасному *Visual Basic* дозволяється записати в одному рядку кілька коротких операторів, однак їх необхідно розділити символом двокрапки, наприклад

```
A=7 : B=5 : C=A+B .
```

Часто виникає протилежна проблема, коли оператор має велику довжину. Такий оператор можна розбити на декілька рядків, записуючи в кінці кожної частини оператора пропуск та символ підкреслення " _ ".

Для того, щоб програмний код був зрозумілий не тільки автору програми, але й іншим користувачам рекомендується вводити коментарії в

кожній частині програми. Ознакою початку коментарію є символ апострофа, він діє лише до кінця рядка.

Гарним тоном в програмуванні є оголошення всіх змінних на початку програми, причому ім'я змінної повинно свідчити про її призначення, а префікс імені – про її тип. Ім'я змінної і константи може поєднувати кілька слів, кожне з яких бажано писати з великої літери.

Професійною ознакою програми є також використання обробників виняткових ситуацій, які повинні видавати зрозумілі повідомлення про настання типових помилок (відсутність потрібних файлів, введення некоректних даних тощо).

Порядок виконання роботи

1. Написати програму для обчислення заданого арифметичного виразу з використанням підпрограм та функцій.
2. Організувати введення початкових даних у програму за допомогою діалогового вікна *InputBox* і виведення результатів за допомогою вікна повідомлення *MessageBox*.
3. Використати в програмі клас *Array* для виконання операцій з масивом.
4. Використати в програмі різні оператори циклів.
5. Створити новий проект у середовищі *Visual Basic 2008*.
6. Ввести текст програми і налагодити її.

Контрольні запитання

1. Які існують типи даних у *Visual Basic 2005/2008*?
2. Які Ви знаєте методи для перетворення типів даних?
3. Як можна оголосити змінні у програмі?
4. Що означає область видимості для змінних?
5. Назвіть основні математичні операції.
6. Чим функції відрізняються від підпрограм?
7. Як передати аргументи у процедуру?
8. Як ввести початкові дані у программу та вивести результати обчислень?
9. Як створюються масиви фіксованого розміру і динамічні масиви?
10. Поясніть можливості класу *Array* для роботи із массивами.
11. Які існують оператори для програмування розгалужень у програмі?
12. Які існують оператори для програмування циклів у програмі?

ТЕМА № 2

Базові елементи інтерфейсу

Зміст теми. Знайомство з різними типами інтерфейсу: однодокументним, багатодокументним та типу провідника. Способи створення меню, панелі інструментів, рядка стану.

Теоретичні відомості

2.1 Загальні відомості про інтерфейс

При створенні будь-якої програми важливою задачею є створення дружнього інтерфейсу з користувачем. Наявність простого і наочного інтерфейсу суттєво полегшує перше знайомство з програмою та її наступне використання. В конкурентній боротьбі різних програмних продуктів завжди перемагає програма із зручним і гарним зовнішнім інтерфейсом.

У сучасних операційних системах використовуються сотні різних програмних пакетів і, звичайно, було б дуже важко їх використовувати, якби кожна програма мала свої оригінальні способи зовнішньої взаємодії. Тому важливим принципом при розробці інтерфейсу користувача є його стандартизація.

Стандартизація проявляється як у використанні готових елементів інтерфейсу (меню, панелі інструментів, діалогових вікон тощо), так і в їх призначенні: назвах, розташуванні. Завдяки цьому значно зменшуються терміни розробки самих програм і час освоєння користувачем нових продуктів. Задача розробника зводиться по суті до формування інтерфейсу із готових інтерфейсних компонентів і написання невеликого програмного коду, який повинен виконуватися при активізації відповідних елементів інтерфейсу.

В цьому розділі ми розглянемо розробку інтерфейсу користувача з використанням основних складових інтерфейсу будь-якої прикладної програми: меню, панелі інструментів, рядка стану .

2.2 Типи інтерфейсів

Зараз для програм, розроблюваних у середовищі *Windows* з допомогою *Visual Basic*, використовується три типи інтерфейсів:

– однодокументний (*Single-Document Interface, SDI*);

– багатодокументний (*Multiple-Document Interface, MDI*);

– інтерфейс типу провідника (*Explorer*).

SDI-інтерфейс надає можливість роботи тільки з одним документом в одному вікні. Прикладом такого інтерфейсу може бути *Блокнот* (*Notepad*). При роботі з декількома документами і різними даними в

такому інтерфейсі необхідно багаторазово запускати програму, що призводить до додаткових витрат оперативної пам'яті. До складу *SDI*-інтерфейсу, як правило, входить тільки головне меню і панель інструментів.

Головна відмінність *MDI*-інтерфейсу полягає в тому, що для одного інтерфейсу можна в одному вікні відкривати багато різних документів. *MDI*-програми зазвичай складаються з головного (батьківсього вікна), в межах якого можна відкривати багато дочірніх вікон. Загалом до складу *MDI*-інтерфейсу входять меню, панель інструментів, рядок стану, головне і дочірнє вікна, засоби керування дочірніми вікнами (рис. 2.1). Прикладом *MDI*-програми може бути *Microsoft Word*.

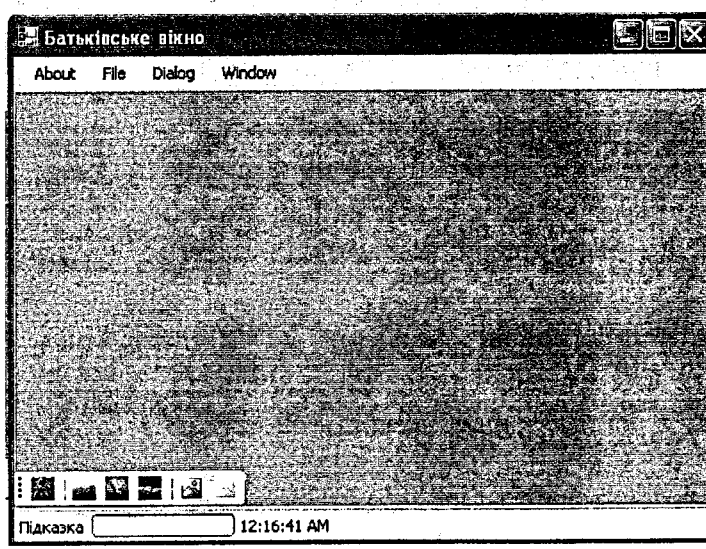

Рисунок 2.1 – Приклад батьківського вікна

Інтерфейс типу провідника розробляється для доступу до ієрархічних деревоподібних структур, тобто до таких структур, де присутня вкладеність. Прикладом вкладеності можуть бути папки і файли. Це аналогія *SDI*-інтерфейсу, розробленого спеціально для деревоподібних структур. Прикладом такого інтерфейсу є Провідник (*Explorer*) у *Windows*.

2.3 Батьківське і дочірні вікна *MDI*-інтерфейсу

Батьківське вікно *MDI*-інтерфейсу є контейнером для дочірніх вікон. При його мінімізації разом з ним мінімізуються і всі його дочірні вікна. У тому випадку, коли хоча б одне дочірнє вікно не вміщується у видиму частину батьківського вікна, в головному вікні з'являється смуга

прокрутки. Для створення батьківського вікна *MDI*-інтерфейсу необхідно присвоїти значення *True* властивості *IsMdiContainer* стандартної форми *Windows*.

Дочірні вікна можуть знаходитись тільки всередині батьківського і не можуть бути винесені за його межі. При розгортанні дочірні вікна займають весь простір батьківського вікна, а до його заголовку додається заголовок активного дочірнього вікна у квадратних дужках (рис. 2.2). При згортанні дочірнього вікна його піктограма відображається внизу батьківського вікна. Дочірні вікна зручно створювати в результаті виконання обробників подій головного меню батьківського вікна.

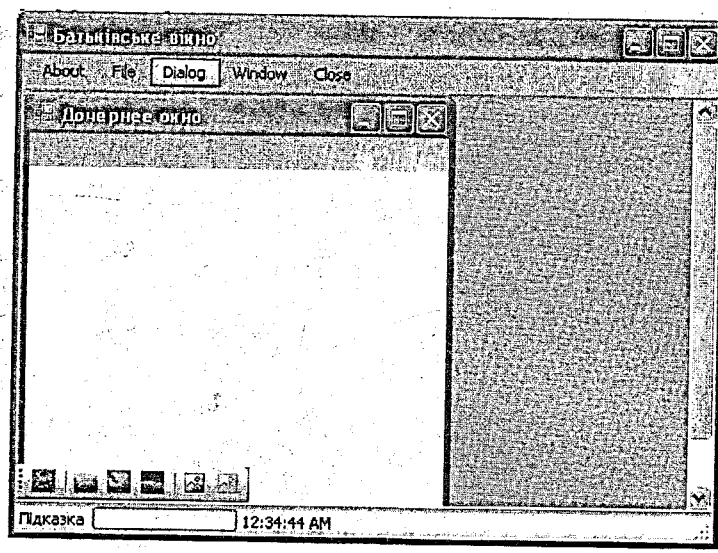

Рисунок 2.2 – Приклад батьківського і дочірнього вікна

Програмний код такого обробника подій може бути таким:

```
Dim ChildWindow  As New Form
ChildWindow.MdiParent=Me
ChildWindow.Show()
```

Розташуванням дочірніх вікон у батьківському вікні можна керувати за допомогою методу *LayoutMdi*, параметр якого може приймати одне із таких значень:

ArrangeIcons, Cascade – каскадне розташування;

TileHorizontal – розташування у вигляді горизонтальної мозаїки;

TileVertical – розташування у вигляді вертикальної мозаїки;

Для розташування дочірніх вікон, наприклад, у вигляді горизонтальної мозаїки потрібно дописати у наведений вище обробник події такий рядок: *Me.LayoutMdi(MdiLayout.TileHorizontal)*

2.4 Меню

Для швидкого доступу до усіх функцій програми є меню: головне меню і контекстне меню.

У *Visual Basic 2005/2008* для проектування головного меню використовується елемент керування *MenuStrip*. Процес створення меню складається з двух основних етапів:

– створення візуального рисунку меню;

– написання процедур обробки подій для кожного пункту меню.

Для створення візуального рисунка меню використовується спеціальний дизайнер меню, який вмикається після перенесення на форму *MenuStrip* (рис.2.3). При цьому об'єкт меню з'являється внизу форми на панелі „Область компонент", а в верхній частині форми з'являється тільки одне поле *"TypeHere"*. З цього першого поля і розпочинається створення всіх необхідних пунктів меню.

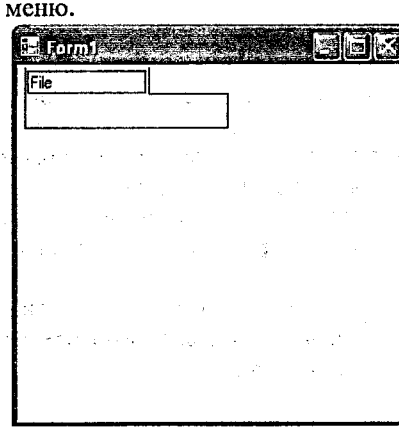

Рисунок 2.3 – Вікно дизайнера меню

Для того, щоб пункти меню виконували команди, для них необхідно написати відповідні процедури обробки подій.

Подія – це нове поняття, якого не було раніше в об'єктно-орієнтованому програмуванні. Подія визначає механізм, що пов'язує якусь дію (в даному випадку вибір пункту меню) з програмним кодом. Такий програмний код, реалізований за допомогою одного або кількох методів, називають обробником події.

Найпростішим обробником події на вибір пункту меню може бути поява вікна повідомлення *MessageBox*. В попередній темі детально розглядалися різні способи виклику такого вікна. Програмний код для виклику вікна повідомлення необхідно записати в середині автоматично згенерованих операторів процедур в редакторі коду *Code Editor*, який викликається подвійним клацанням миші на відповідному пункті меню.

Наприклад, якщо при виборі пункта меню „*About*" повинно з'явитися вікно повідомлення, в заголовку якого має бути назва пункту меню, а текст в середині вікна має містити прізвище розробника, (наприклад, „*Ковальчук*"), тоді в редакторі коду необхідно записати:

MsgBox("About ", MsgBoxStyle. OkOnly, "Ковальчук")

Виконувати команди меню можна також з допомогою клавіатури, наприклад, пункти меню "*Open*" можна відкрити, натиснувши на клавішу "*Alt*", а потім не відпускаючи натиснути також ще на одну клавішу – клавішу швидкого доступу. Такою клавішою може бути будь-яка клавіша (частіше перша буква назви пункту меню). Щоб добавити до пункту меню клавішу доступу, необхідно знову активізувати дизайнер меню і ввести перед потрібною буквою в імені меню символ „*амперсента*" (&). Потрібно звернути увагу, що не можна назначати одну і ту саму клавішу для двох і більше пунктів меню.

2.5 Контекстне меню

Контекстне меню – це меню, яке пов'язане з деякою подією (зазвичай це натиснення правої клавіші миші на об'єкті), яка може бути викликана в будь-якому місці програми. В початковому стані контекстне меню невидиме і візуалізується поруч із курсором миші після свого виклику. Контекстним таке меню називається тому, що воно з'являється поруч із вибраним об'єктом і його склад залежить від змісту (контексту) цього об'єкту).

Для створення контекстного меню використовується елемент керування *ContextMenuStrip* і процес його формування, як і для головного меню, здійснюється за два етапи.

2.6 Панель інструментів

Панель іструментів (*Toolbar*) – це сукупність візуальних елементів, які виконують функції, аналогічні меню. Головна відмінність від меню полягає в тому, що текстові назви замінюються рисунками. Панель інструментів дублює роботу меню, але вона працює дещо швидше і більш зручна в користуванні. Крім традиційних кнопок, *Visual Basic 2005/2008* надає більшу кількість нових елементів і додаткових властивостей.

Розробка панелі інструментів схожа на процес створення меню і також складається з двух основних етапів:

– візуальне створення панелі інструментів;

– написання процедур обробки подій для кожного елементу панелі інструментів.

Для створення панелі інструментів у вікні *Toolbox* є спеціальний ЕК *ToolStrip*. Візуальне проектування зводиться, головним чином, до вибору елементу панелі інструментів із списку можливих елементів: кнопки,

мітки, списки, розділювачі, текстові поля, індикатори виконання (рис. 2.4). Можна створити власний рисунок у вбудованому графічному редакторі, вибрати готову піктограму або підключити попередньо створені зображення в будь-якому графічному редакторі.

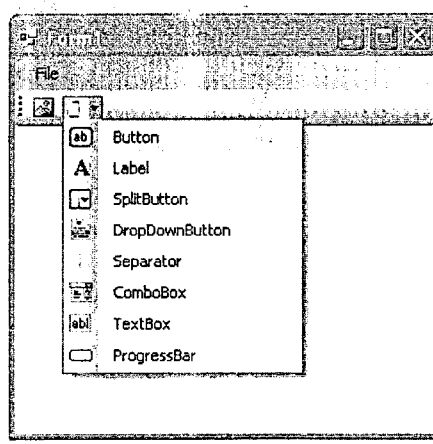

Рисунок 2.4 – Список елементів панелі інструментів

За замовчуванням панель інструментів знаходиться у верхній частині вікна програми. Щоб змінити її знаходження, необхідно використати властивість *Dock*, яка може приймати одне із значень *DockStyle*: *Bottom* (внизу), *Fill* (на всю форму), *Left* (зліва), *None* (в будь-якому місці), *Right* (справа), *Top* (зверху).

Для того, щоб можна було під час виконання програми змінювати положення елементу на панелі інструментів за допомогою миші та клавіші *<Alt>* необхідно властивості *AllowItemReorder* присвоїти значення *True*.

У властивостях панелі інструментів і її окремих складових є можливість для зміни розмірів, кольорової гами, шрифтів, різних підказок і таке інше.

2.7 Рядок стану

Рядок стану – це рядок у батьківському вікні, який можна розділити на декілька частин для відповідного виведення в них різної поточної довідкової інформації. Наприклад, можна виводити точні підказки про функції вибраних пунктів меню чи панелі інструментів або вивести інформацію про поточні дату та час.

Для настроювання рядка стану необхідно спочатку перенести на форму ЕК *StatusStrip* і потім задати необхідні значення властивостей. За замовчуванням рядок стану розміщується в нижній частині вікна. Список

можливих елементів рядка стану схожий на відповідний список панелі інструментів, але вдвічі менший (рис. 2.5). Для зміни його розташування використовується властивість *Dock*, яка може приймати ті ж значення, що і однойменна властивість панелі інструментів: *Bottom* (внизу), *Fill* (на всю форму), *Left* (зліва), *None* (в будь-якому місці), *Right* (справа), *Top* (зверху).

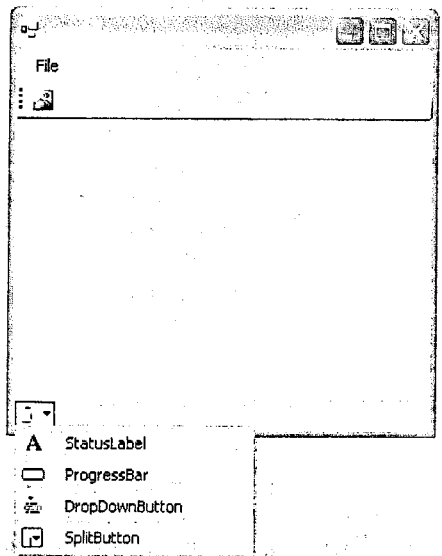

Рисунок 2.5 – Список елементів рядка стану

Для рядка стану також можна написати обробники різних подій. Наприклад, для виведення інформації про поточний час в рядку стану з іменем *status1* необхідно спочатку перенести на батьківську форму ЕК *Timer*, присвоїти йому імя *timer1*, далі в процедуру обробки події *timer1_Tick* цього ЕК записати програмний код

status1.Text = System.DateTime.Now.ToLongTimeString

Активізація таймера відбувається після додання в процедуру обробки *MdiForm1_Load* батьківського вікна з іменем *MdiForm1* рядка

timer1.Enabled = True

2.8 Інтерфейс типу провідника

Для створення інтерфейсу типу провідника використовуються ЕК *ListView* і *TreeView*, які можна знайти у вікні *ToolBox*.

Інтерфейс програми типу провідника складається з головного меню, панелі інструментів, рядка стану, ієрархічного списку елементів деревоподібної структури (файлів, папок, документів).

Порядок виконання роботи

Виконати послідовно завдання, які подані у таблицях 2.1 – 2.3.

Таблиця 2.1 – Створення головного меню для *SDI* – інтерфейсу

Завдання	Виконання
1. Перенести на форму Form1 ЕК MenuStrip	Клацнути мишею на ЕК *MenuStrip* закладки *Menus & Toolbars* вікна *Toolbox*
2. Створити рисунок меню	Клацнути мишею на полі "*Type Here*" і ввести назву для кожного пункту меню
3. Переглянути меню	Натиснути клавішу *F5*
4. Додати клавіші швидкого доступу до команд меню	Перейти в режим дизайнера. Клацнути мишею на полі "*Type Here*" для кожного пункту меню і ввести перед іменем меню символ „*амперсента*" (&)
5. Написати процедуру обробки подій для кожного пункту меню	Активізувати дизайнерське меню, клацнути мишею двічі на імені кожного пункту меню і в редакторі коду ввести команду виклику вікна повідомлення *MsgBox*
6. Перевірити роботу меню	Натиснути клавішу *F5*. Викликати по черзі кожний пункт меню за допомогою курсора миші та відповідної клавіші швидкого доступу

Таблиця 2.2. – Створення дочірніх вікон для *MDI* -інтерфейсу

Завдання	Виконання
1. Підготувати створення MDI-інтерфейсу у формі Form1	Відкрити вікно "*Properties*" форми *Form1* і задати для неї такі значення: *Name = MdiForm1* *IsMdiContainer = True* *Text = Батьківське вікно*
2. Створити пункт "Add Window" в головному меню	Виконати завдання із таблиці 2.1 тільки для пункту "*Add Window*" меню.
3. Запрограмувати створення та виклик дочірнього вікна через головне меню батьківського вікна	Написати таку процедуру обробки подій для пункту "*Add Window*" меню: *Dim ChildWindow As New Form* *ChildWindow.MdiParent=Me* *ChildWindow.Show()*
4. Створити нові дочірні вікна в батьківському вікні	Натиснути клавішу *F5* і потім по черзі викликати пункт "*Add Window*"
5. Закрити дочірні вікна через головне меню батьківського вікна	По черзі викликаючи пункт меню "*Close Window*", закрити дочірні вікна в зворотному порядку їх створення

Таблиця 2.3. – Створення панелі інструментів

Завдання	Виконання
1. Перенести на форму ЕК ToolStrip	Клацнути мишею на ЕК *ToolStrip* закладки *Menus & Toolbars* вікна *Toolbox*
2. Створити набір кнопок панелі інструментів	Клацнути мишею на списку *Add ToolStripButton* у лівому верхньому куті смуги панелі інструментів і вибрати ЕК *Button*
3.Вибрати зображення для кожної кнопки панелі інструментів	В контекстному меню кожної кнопки вибрати пункт "*Set Image...*". В відкритому діалоговому вікні натиснути кнопку "*Import...*" і вибрати графічний файл з необхідним зображенням
4. Створити підказку функції кожної кнопки	В контекстному меню кнопки вибрати команду "*Properties*" і задати текст підказки для властивості *ToolTipText*
5. Налаштувати додаткові властивості кнопок панелі інструментів	В контекстному меню кнопки вибрати команду "*Properties*" і задати нові значення для кольорів та інших параметрів
6. Розташувати панель інструментів в різних місцях батьківського вікна	Через контекстне меню панелі інструментів вибрати пункт "*Properties*". Для властивості *Dock* вибрати різні варіанти розташування
7. Написати процедуру обробки подій для кожної кнопки панелі інструментів	Клацнути мишею двічі на рисунку кожної кнопки і в редакторі коду ввести команди виклику вікна повідомлення *MsgBox*
8. Перевірити роботу панелі інструментів	Натиснути клавішу *F5*

Контрольні запитання

1. Назвіть основні типи інтерфейсів *у середовищі Windows.*
2. Як створити батьківське і дочірні вікна для *MDI*-інтерфейсу?
3. Поясніть послідовність розроблення головного меню програми.
4. Поясніть послідовність розроблення контекстного меню програми.
5. Поясніть послідовність розроблення панелі інструментів програми.
6. Яке призначення рядка стану?
7. Чим модальне вікно відрізняється від немодального вікна?
8. Розкажіть про інтерфейс програми типу провідника.

ТЕМА № 3

Основні елементи керування

Зміст теми. Знайомство з елементами керування для введення-виведення даних і вибору варіантів.

Теоретичні відомості

3.1 Елементи керування в середовищі Windows

Елементи керування (ЕК) – це компоненти, з яких будується інтерфейс програми. Від вміння правильного вибору ЕК та їх грамотного розташування на формах залежить зручність роботи з програмою.

ЕК є складовою частиною операційної системи *Windows*. В середовищі *Visual Studio 2008* для всіх мов програмування (*C++, C#, Visual Basic*) ЕК та принципи роботи з ними залишаються незмінними. Тому, засвоївши роботу з ними при вивченні мови *Visual Basic*, легко можна зрозуміти техніку візуального програмування не тільки з іншими мовами у *Visual Studio 2008*, а також і в інших середовищах візуальної розробки (*C++ Builder, Delphi*).

ЕК в більшості випадків розміщуються на формі, рідше – на інших ЕК (такі ЕК, які можуть містити інші ЕК, називають контейнерами). Спочатку всі ЕК знаходяться на панелі елементів керування (*Toolbox*) (рис. 3.1). Зазвичай ця панель в згорнутому вигляді вже знаходиться в головному вікні *Visual Studio*, її також можна викликати командою *View/Toolbox* в головному меню. *Toolbox* містить декілька вкладок, в кожній з яких об'єднані ЕК для виконання близьких за призначенням задач. При вивченні цієї теми нам знадобляться ЕК із вкладки *Common Controls*.

З позицій теорії програмування компонент є подальшим розвитком поняття клас. Якщо клас має лише дані та функції (методи), то компонент характеризується вже трьома параметрами – властивостями, методами і подіями.

Властивості компонентів розширюють поняття даних класу, крім основних характеристик компонента вони можуть мати і свої власні функції. На відміну від даних класу, властивість може приховувати деталі реалізації від користувача.

Кожний компонент може мати кілька десятків властивостей, які можна побачити у спеціальному вікні, що викликається командою *Properties* із контекстного меню компонента. В табл. 3.1 наведені деякі властивості, які зустрічаються в більшості ЕК. Як і належить даним, властивостям також мають бути присвоєні конкретні значення.

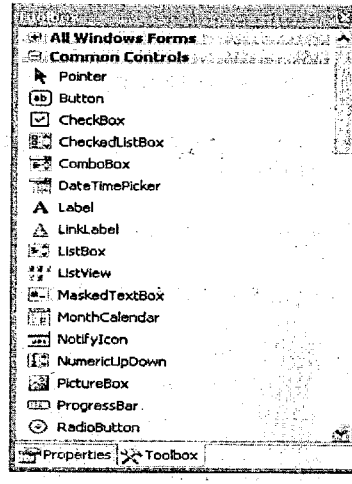

Рисунок 3.1 – Вікно *Toolbox*

Таблиця 3.1 – Загальні властивості елементів керування

Властивість	Опис
Name	Ім'я ЕК
Dock	Розташування ЕК на формі (зверху, знизу, зліва, справа, на всю форму, в будь-якому місці)
Location	Координати лівого верхнього кута ЕК
Size	Розмір ЕК, який включає ширину і висоту
Font	Шрифт для відображення тексту в ЕК
ForeColor	Колір тексту на ЕК
BackColor	Колір фону ЕК
Visible	Видимість ЕК на формі під час виконання
Enabled	Доступність використання ЕК

Для більшості властивостей такі значення вже присвоєно за замовчуванням, при необхідності їх можна замінити новими значеннями. Наприклад, щоб найуживанішому ЕК – кнопці *Button1* – дати нове ім'я *"Ok"*, необхідно присвоїти відповідне значення властивості *Name* :

Button1.Name = "Ok"

Як видно із наведеного запису, належність властивості *Name* до ЕК *Button1* задається точкою. Таке ж правило існує і для методів. Крім спільних властивостей, кожний ЕК має декілька індивідуальних властивостей, які і визначають поведінку цього ЕК. Алгоритм візуальної програми в основному залежить від аналізу значень саме цих індивідуальних властивостей ЕК.

Компонентні методи нічим не відрізняються від функцій-членів класу. Вони також задають якусь конкретну операцію з компонентом. Наприклад, для запису в ЕК *Listbox1* слова "*word*" необхідно використати метод *Add()*:

ListBox1.Items.Add("word")

На відміну від властивостей, методам не присвоюють значень. Останній запис показує ще одну характерну особливість використання методів: вони можуть зв'язуватись з ЕК не безпосередньо, а через властивості, в даному випадку через властивість *Items*.

З третім параметром компонентів – подією – ми вже познайомились при створенні меню. Нагадаємо, що подія визначає механізм, що пов'язує якусь дію (вибір пункту меню, натиснення клавіші клавіатури чи кнопки миші) з програмним кодом. Такий програмний код, реалізований за допомогою одного або кількох методів, називають обробником події. Для багатьох стандартних подій в операційній системі вже розроблені стандартні відповіді. Наприклад, при натисненні лівої клавіші миші Windows посилає повідомлення *WM_LBUTTONDOWN*, в результаті чого викликається відповідний обробник події.

Прикладний програміст не зобов'язаний писати обробники на всі можливі події, а записує лише ті, які пов'язані із заданим алгоритмом задачі. Таким чином, у візуальному програмуванні реалізовано стиль програмування, який можна назвати програмуванням подій. Візуальна програма – це набір обробників подій. Кожний обробник прив'язаний до якоїсь події, як правило, це вибір відповідного пункту меню, кнопки панелі інструментів чи іншої кнопки на формі.

Всі перераховані вище події у *Visual Studio* відносяться до події *OnClick*. Коли причиною виникнення такої події має бути натиснення ЕК *Button1* на формі, тоді автоматично формується заголовок процедури *Button2_Click()*. Програмісту залишається лише написати тіло цієї процедури. Після запуску на виконання візуальної програми цей обробник події буде знаходитись в стані очікування, активізувати його можна тільки після натиснення ЕК *Button1*.

3.2 Елементи керування для введення і виведення даних

3.2.1 Текстове поле

Зручним способом введення, редагування і виведення числових та символьних даних у Visual Studio може служити ЕК *TextBox* (Текстове поле) (рис. 3.2). Найважливішою індивідуальною властивістю цього ЕК є властивість *Text*. Найпростіше вводити текстовий рядок:

```
Dim str As String
str = TextBox1.Text
```

або його виводити:

```
str = "Рядок 1"
TextBox1.Text = str
```

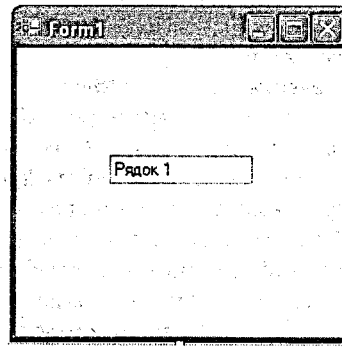

Рисунок 3.2 – ЕК *TextBox*

Можна заборонити введення даних у *TextBox,* якщо для властивості *ReadOnly* задати значення *True.*

За замовчуванням передбачається, що служить для введення одного рядка тексту, причому достатньо довгого (до 32767 символів). Але текст можна сформувати також у вигляді блоку з кількох рядків, якщо для властивості *MultiLine* задати значення *True.* Для великого блоку текстової інформації можна додати у *TextBox* смуги прокрутки по горизонталі або вертикалі за допомогою властивості *ScrollBars* (рис. 3.3).

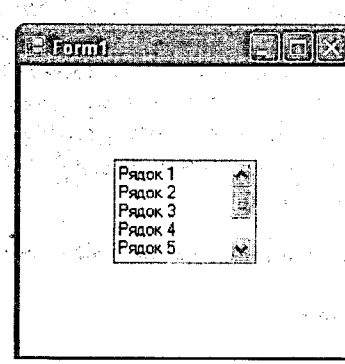

Рисунок 3.3 – ЕК *TextBox* із властивістю *MultiLine=True*

Якщо властивость *WrapWord* має значення *True,* тоді автоматично здійснюється переніс слів на новий рядок. В протилежному випадку перехід на новий рядок здійснюється при натисненні на клавішу *Enter.*

Передбачено кілька властивостей для керування текстом. Наприклад, можна вибрати із *TextBox* лише частину тексту, а саме того, який буде виділено курсором миші:

str = TextBox1.SelectedText .

TextBox дозволяє організувати перевірку введених даних, якщо властивість *CausesValidation* має значення *True*. Наприклад, для перевірки введення числових даних необхідно ще використати функцію *IsNumeric:*

```
If Not IsNumeric(TextBox1.Text) Then
MsgBox("Неправильний формат")
```

3.2.2 Текстове поле з маскою

Ще у *Visual Studio 2005* з'явився новий ЕК *MaskedTextBox*, (Текстове поле з маскою) який схожий на *TextBox*, однак дозволяє задавати маску для даних, що вводяться.

Наприклад, якщо в такому ЕК передбачається тільки введення дати у форматі ДД/ММ/РРРР, тоді значення властивості *Mask* повинно мати вигляд: 00/00/0000.

Нулі у масці означають, що мають бути введені тільки цифри. Якби дозволялося введення тільки текстових даних, тоді маскою служив би символ *L*.

Варто також відмітити, що *MaskedTextBox* не може бути багаторядковим і не підтримує переніс слів.

3.2.3 Мітка

ЕК *Label* (Мітка) зручна для розміщення тексту безпосередньо на формі. Текст мітки задається властивістю *Text* і може бути зміненим тільки програмно. Наприклад:

Label.Next="Приклад тексту"

Додатково можна задати властивості, які визначають параметри тексту: шрифт (*Font*), вирівнювання тексту (*TextAlign*) та ін.

3.3 Елементи вибору варіантів

3.3.1 Прапорець

Для введення у форму даних, які можуть мати лише один із двох можливих значень, призначений ЕК *CheckBox* (Прапорець) (рис. 3.4). Одне з цих значень приймається істинним, і тоді користувач повинен встановити прапорець. В протилежному випадку поле прапорця залишається пустим, що свідчить про негативну відповідь на поставлене запитання.

Далі програмно аналізується стан прапорця, і виконання алгоритму програми розгалужується відповідним чином. Приклад програмної перевірки того чи встановлений прапорець:

```
If (CheckBox1.Checked =True) Then
    MsgBox ("Yes")
Else
    MsgBox ("No")
End If
```

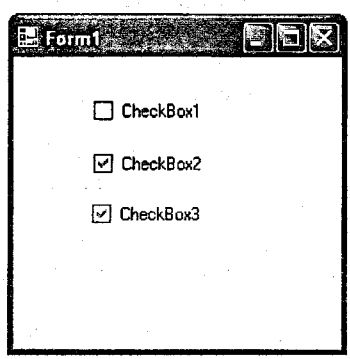

Рисунок 3.4 – ЕК *CheckBox*

Можливий також ще один стан прапорця, коли він невизначений. Цей стан дозволяється використати для розгалуження алгоритму на три варіанти вибору, якщо встановити властивість *TreeState=True*. В цьому випадку для вибору варіантів можна скористатись властивістю *CheckState*, яка може мати три значення: *Checked* (встановлено), *Unchecked* (скинуто) і *Indeterminate* (невизначено):

```
If CheckBox1.CheckState = CheckState.Checked Then
    MsgBox("Variant 1")
End If
If CheckBox1.CheckState = CheckState.Unchecked Then
    MsgBox("Variant 2")
End If
If CheckBox1.CheckState = CheckState.Indeterminate Then
    MsgBox("Variant 3")
End If
```

Властивість *CheckState* дозволяє не тільки перевірити, але також програмно встановити необхідний стан прапорця.

3.3.2 Перемикач

ЕК *RadioButton* (Перемикач) використовуються в тих випадках, коли необхідно вибрати тільки один варіант із кількох можливих (рис. 3.5).

Тому на формі має бути група перемикачів і сама система слідкує за тим, щоб вибраним був тільки один перемикач. Якщо користувач вибирає курсором миші новий перемикач, тоді він стає вибраним, а раніше вибраний автоматично стає невибраним.

Для аналізу стану перемикача зручно користуватись його властивістю *Checked*, яка може приймати два значення: вибрано (*True*) та невибрано (*False*). Приклад програмної перевірки того чи був вибраний перемикач *RadioButton2* із групи перемикачів:

```
If (RadioButton2.Checked =True) Then
        MsgBox ("Yes")
Else
        MsgBox ("No")
End If
```

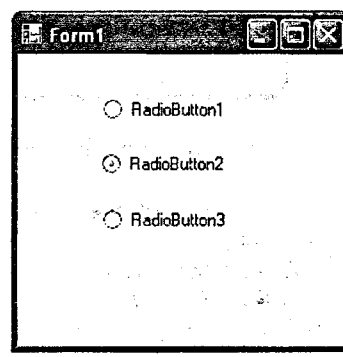

Рисунок 3.5 – ЕК *RadioButton*

Всі перемикачі на формі автоматично утворюють тільки одну групу, і тому тільки один перемикач може бути вибраним. Якщо виникає потреба в організації двох або більше груп з можливістю вибору одного перемикача в кожній групі, тоді на форму потрібно перенести ЕК *GroupBox* (Група). Цей ЕК стане контейнером для необхідної кількості груп перемикачів.

3.3.3 Список

ЕК *ListBox* (Список), розміщений на формі, являє собою список, із якого користувач може вибрати одне із наявних значень (рис. 3.6). Значення у списку можуть розміщуватись в одну або декілька колонок в залежності від значення властивості *MultiColumn*. Якщо всі значення списку на поміщаються у виділену для них область на формі, тоді автоматично з'являються смуги прокрутки, їх також можна залишати постійно при заданні значення *True* властивості *ScrollAlwaysVisible*.

Найважливішою властивістю для списку є властивість *Items*, оскільки вона забезпечує використання методів для реалізації всіх основних операцій зі списком. Цією властивістю зручно користуватись як на етапі конструювання форми, так і при програмування роботи зі списком.

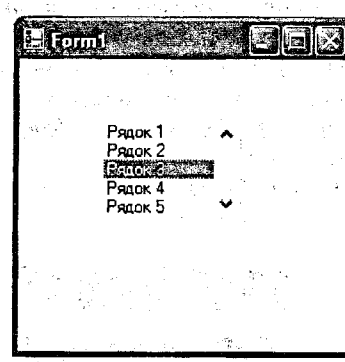

Рисунок 3.6 – ЕК *ListBox*

Розглянемо приклади програмування основних операцій зі списком.

а) заповненя списку:

Listbox1.Items.Add("Київ")
Listbox1.Items.Add("Лондон")
Listbox1.Items.Add("Париж")

б) вилучення елемента списку за його значенням:

Listbox1.Items.Remove("Лондон")

в) вилучення елемента списку за його номером:

Listbox1.Items.RemoveAt(1)

г) вилучення вибраного елемента списку:

Listbox1.Items.Remove(ListBox1.SelectedIndex)

д) вставка нового елемента в задану позицію списку:

Listbox1.Items.Insert(2,"Москва")

е) очищення списку:

Listbox1.Items.Clear()

Додавати в список можна відразу кілька елементів, якщо скористатись таким способом:

Dim str() As String={"Київ", "Лондон", "Париж"}
Listbox1.Items.AddRange(str)

Дані в список необов'язково вводити в алфавітному порядку, їх можна відсортувати, якщо записати:

ListBox1.Sorted = True

Отримати доступ до елементу списку можна за індексом, який визначає його положення в списку:

Dim st As String= Listbox1.Items(3)

або вибравши його курсором миші під час виконання програми:

Dim str As String = ListBox1.SelectedItem

Дозволяється вибрати одночасно також декілька елементів списку, якщо властивості *SelectionMode* присвоїти значення *MultiSimple* або *MultiExtended* (різниця між ними у способі виділення елементів списку).

Якщо необхідно знайти елемент списку можна використати метод *FindString()*, синтаксис якого такий:

Listbox1.FindString(str As String, n As Integer)

де *n* – позиція, з якого буде здійснюватись пошук (цей параметр можна пропустити, якщо пошук починати від початку списку);

str – значення елементу списку (можна задати і частину значення елементу, яка однозначно його ідентифікує).

Метод *FindString()* повертає індекс елементу у списку або -1 при відсутності у списку заданого елементу.

Підрахувати кількість *m* елементів списку можна таким чином:

Dim m As Integer = Listbox1.Items.Count

3.3.4 Комбінований список

ЕК *ComboBox* (Комбінований список) поєднує в собі можливості *ListBox* і *TextBox*: дозволяє вибрати необхідний елемент зі списку або набрати його з клавіатури (рис. 3.7). Перевагою цього ЕК є також економний спосіб зображення на формі.

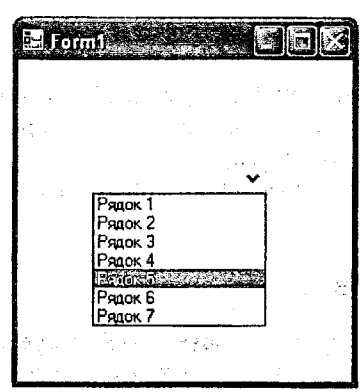

Рисунок 3.7 – ЕК *ComboBox*

Зовнішній вигляд *ComboBox* визначається його властивістю *DropDownStyle*, яка може приймати такі значення:

Simple – просте поле зі списком (на формі відображається список і текстове поле, доступне для редагування);

DropDown – поле зі списком, що розкривається (на формі відображається текстове поле зі стрілкою, поле доступне для редагування);

DropDownList – список, що розкривається (на формі відображається текстове поле зі стрілкою, але поле недоступне для редагування).

Всі інші операції з елементами списку аналогічні, як у ЕК *ListBox*.

3.3.5 Список з прапорцями

ЕК *CheckedListBox* (Список з прапорцями) є ще одним прикладом поєднання можливостей двох елементів керування: в даному випадку

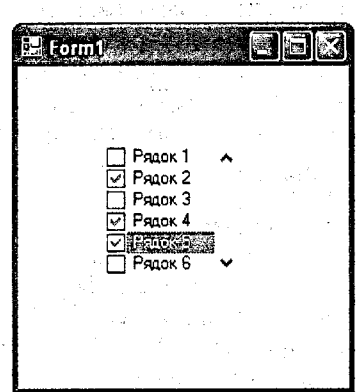

Рисунок 3.8 – *CheckedListBox*

ListBox і *CheckBox* (рис. 3.8). На формі цей ЕК має вигляд списку, кожному елементу якого поставлено у відповідність прапорець. Таким чином зручно виділяти окремі елементи списку.

ЕК *CheckedListBox* має всі можливості *ListBox*, додаючи при цьому декілька нових.

3.3.6 Числове поле

ЕК *NumericUpDown* (Числове поле) є спеціалізованим елементом керування, яке забезпечує введення тільки цілих або дробових чисел. На формі цей ЕК являє собою текстове поле і дві кнопки з протилежно направленими стрілками (рис. 3.9). Кожне натиснення кнопки зі стрілкою доверху збільшує, а кожне натиснення кнопки зі стрілкою донизу зменшує в текстовому полі число на задану величину.

Для визначення початкового значення *n* списку використовується властивість *Value*. Для визначення максимального і мінімального значення текстового поля використовуються властивості *Minimum* та *Maximum*. За замовчуванням вони приймають відповідно значення 0 і 100.

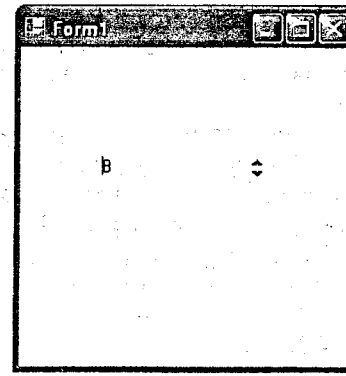

Рисунок 3.9 – *NumericUpDown*

За допомогою властивості *Increment* задається крок, з яким буде збільшуватись або зменшуватись числове значення в текстовому полі при натисненні відповідної кнопки. За замовчуванням цей крок дорівнює 1.

Приклад задання початкових значень для ЕК *NumericUpDown:*

NumericUpDown1.Value = 10
NumericUpDown1.Maximum = 50
NumericUpDown1.Minimum = -50
NumericUpDown1.Increment = 2

Щоб ЕК відображав шістнадцяткові значення замість десяткових, необхідно задати значення *True* для властивості *Hexadecimal.*

3.4 Робота з кількома формами

Для програм, які використовують інтенсивний діалог з користувачем, однієї форми замало. *Visual Basic 2008* надає можливість працювати з багатьма формами різного типу. Щоб додати у проект нову форму, необхідно виконати команду *Add Windows Form* із головного меню *Project.* В результаті на екрані з'явиться діалогове вікно *Add New Item.* І хоча крім форм у цьому вікні містяться всі можливі елементи, які можна додати до проекту, легко можна знайти і можливі типи додаткових форм. Найпростішою формою є перша за списком *Windows Form.* Зручною є форма *Dialog*, яка вже містить готові кнопки *Ok* та *Cancel.* Після вибору потрібної форми і натиснення клавіші *Add* вона автоматично ввійде до складу проекту. Кожна нова форма розглядається як об'єкт класу *System.Windows.Forms.Form.* Перша форма програми називається *Form1*, а всі наступні форми будуть мати імена *Form2*, *Form3* і так далі (якщо, звичайно, вибирались форми шаблону *Windows Form).* Для інших типів форм нумерація теж йде в порядку зростання. Всі форми можуть відображатись одночасно, а можуть завантажуватися і вивантажуватися

при необхідності. При відображенні кількох форм можна легко перемикатись між ними, а також керувати порядком їх використання. Для відображення форми використовується метод *Show()*, а для приховування форми – метод *Hide()*.

Розглянемо програми для роботи із трьома формами. Нехай вибір для відображення другої чи третьої форми здійснюється в залежності від стану прапорця на першій формі. Помістимо на другу та третю форму текстове поле, куди буде записуватись інформація після виклику відповідної форми. Помістимо також на першу форму кнопку *Ok*, програма-обробник для якої буде матиме такий вигляд.

```
If CheckBox1.Checked Then
    Me.Hide()
    Form2.Show()
    Form2.TextBox1.Text = "Form2 is opened"
End If
If CheckBox2.Checked Then
    Me.Hide()
    Form3.Show()
    Form3.TextBox1.Text = "Form3 is opened"
End If
```

Для повернення знову до першої форми можна помістити на другу і третю форми кнопку *Cancel*, програма-обробник для якої буде матиме такий вигляд.

```
Me.Hide()
Form1.Show()
```

Дамо два важливих коментарі до наведених програм. По-перше, для звертання до поточної форми (форми, з якою йде робота в даний момент часу) необхідно використовувати ім'я *Me*, а не її стандартне ім'я. Перед іменем ЕК, який знаходиться не на поточній формі, необхідно вказати стандартне ім'я цієї форми. Форма, якій при її відображенні на екрані має бути приділена основна увага, називається діалоговим вікном (їх ще також часто називають модальними формами). Діалогове вікно не можна залишити і перейти до іншої форми, не натиснувши кнопки *Ok* чи *Cancel* або не закривши це вікно іншим способом. Щоб відкрити існуючу форму як діалогове вікно необхідно для її відкриття використати метод *ShowDialog()*. Також не можна одночасно відкрити декілька діалогових вікон. Якби, наприклад, в попередній програмі замість методу *Show()* використати метод *ShowDialog()*, тоді неможливо було б відкрити одночасно обидві форми.

На завершення відзначимо, що додаткові форми можна отримати не тільки за допомогою діалогового вікна *Add New Item*. Оскільки форми є об'єктами класу *System.Windows.Forms.Form*, їх можна створювати і відображати безпосередньо за допомогою коду програми. Для роботи з додатковими формами можна також використовувати об'єкти *My.Forms*.

Порядок виконання роботи

1. Перенести на форму ЕК *TextBox* (Текстове поле) і перевірити програмно операції введення і виведення текстових даних.
2. Перенести на форму ЕК *MaskedTextBox* (Текстове поле з маскою) і сформувати маску для введення заданих даних.
3. Написати програму, в якій вибір варіантів для обчислення заданого арифметичного виразу здійснюється за допомогою ЕК *CheckBox* (Прапорець).
4. Написати програму, в якій вибір варіантів для обчислення заданого арифметичного виразу здійснюється за допомогою ЕК *RadioButton* (Перемикач).
5. Написати програму для виконання різних операцій зі списком за допомогою ЕК *ListBox* (Список).
6. Написати програму для виконання різних операцій зі списком за допомогою ЕК *ComboBox* (Комбінований список).
7. Провести дослідження можливостей ЕК *NumericUpDown* (Числове поле).
8. Створити три форми і організувати програмне перемикання між ними по циклу (1→2→3→1→ ...) за допомогою відповідних кнопок "*Ok*" та "*Cancel*".

Контрольні запитання

1. Охарактеризуйте компонент з позицій теорії програмування.
2. Поясніть призначення властивостей, подій і методів компонентів.
3. Назвіть загальні властивості компонентів.
4. Розкажіть про призначення ЕК *TextBox*.
5. Як можна перевірити правильність введених даних за допомогою ЕК *TextBox*?
6. Який компонент дає можливість задавати маску при введенні даних?
7. Чим відрізняються між собою ЕК *CheckBox* та ЕК *RadioButton*?
8. Поясніть основні операції зі списком *ListBox*.
9. Які додаткові можливості для роботи зі списками надають ЕК *ComboBox* та ЕК *CheckedListBox*?
10. Як можна ввести у програму дробові числа?
11. Які використовуються методи для роботи з кількома формами?
12. Як можна програмно створити додаткові форми?

ТЕМА № 4

Робота з графікою

Зміст теми. Знайомство із классами *GDI+* для побудови ліній, стандартних геометричних фігур, растрових зображень, виведення тексту.

Теоретичні відомості

4.1 Класи для роботи з графікою

У *Visual Basic 2005/2008* для роботи із графікою використовується система класів *GDI+*. В цю систему входить більше 60 класів, які використовуються для роботи із векторною і растровою графікою а також для для виведення тексту.

Покращений графічний інтерфейс (*Graphics Device Interface, GDI+*) дозволяє створювати лінії, стандартні та довільні геометричні фігури і растрові зображення на формі або спеціальних компонентах. Інтерфейс *GDI+* повністю замінив застарілий *GDI* і є основним програмним засобом створення графіки в програмах *Windows*.

Класи *GDI+* організовані в ієрархічному порядку і доступ до них здійснюється через різні простори імен. Наприклад, доступ до базових класів *GDI+* здійснюється через простір імен *System.Drawing*, до класів векторної графіки – через простір імен *System.Drawing.Drawing2D*, до класів растрової графіки – через простір імен *System.Drawing.Imaging* і т.д. Кожний простір імен складається з набору класів для виконання відповідних задач графіки. З метою скорочення програмного коду доступ до вказаних просторів імен можна також здійснити, якщо на початку програми вказати на них через оператор *Imports*, наприклад:

Imports System.Drawing

4.2 Графічна система координат

В більшості випадків графічні зображення виводяться безпосередньо на форму, тому спочатку потрібно познайомитись із координатною системою форми.

Двовимірна система координат складається із рядків та стовпців апаратно-незалежних графічних елементів – пікселів. Рядки пікселів розташовані вздовж осі *x* (горизонтальна вісь), а стовпці пікселів розташовані вздовж осі *y* (вертикальна вісь). Координати лівого верхнього кута форми завжди рівні (0,0).

У векторній та растровій графіці точка відповідає одному пікселу. Для кожної точки можна задати пару координат (*x, y*) та її колір. Із окремих точок утворюються лінії та геометричні фігури.

4.3 Структури простору імен *System.Drawing*

Тепер детально познайомимся зі структурами простору імен, які дозволяють задавати координати, розміри, колір об'єктів (табл. 4.1).

Таблиця 4.1. – Структури простору імен *System.Drawing*

Структура	Опис структури
Color	Дозволяє задати кольори *ARGB*-кольори об'єктів, де *A* – основна складова, *R,G,B* – червона, зелена та синя складові кольору
Point	Задає координати точки на площині (цілого типу)
PointF	Задає координати точки на площині (дійсного типу)
Rectangle	Задає розташування і розмір прямокутника (цілого типу)
RectangleF	Задає розташування і розмір прямокутника (дійсного типу)
Size	Задає висоту та ширину об'єкта (цілого типу)
SizeF	Задає висоту та ширину об'єкта (дійсного типу)

Кожна структура має свої властивості та методи. Наприклад, структура *Point* має властивості *X, Y*, які повертають відповідно цілочисельну *x*-координату та цілочисельну *y*-координату точки. За допомогою методу *Offset(dx As Integer, dy As Integer)* структура *Point* дозволяє змістити точку на величину *dx* по осі *x* та на величину *dy* по осі *y*. Наступний програмний код створює точку, а потім переміщує її.

```
Dim point1 As New Point(135,180)
point1.Offset(25,50)
```

Структура *Color* надає велику кількість кольорів, які можна викликати вказавши назву кольору, наприклад:

```
Color.Green
```

Існує також можливість формування і своїх власних кольорів. Для цього зручно використати структуру, яка задає колір у форматі *RGB*, де *R,G,B* – червона (*Red*), зелена (*Green*) і синя (*Blue*) складові кольору. Складові кольору задаються в межах від 0 до 255: 0 означає її відсутність, а 255 – максимальне значення. Наявність кількох складових формує заданий колір:

ARGB(255,255, 0, 0) – непрозорий яскраво–червоний колір;
ARGB(0,128, 128, 0) – прозорий жовтий колір;
ARGB(255,255, 255, 255) – білий колір;
ARGB(255,0, 0, 0) – чорний колір

Приклад створення у програмі власного кольору:

```
Dim MyColor As Color = Color.FromArgb(0, 255, 0)
```

Існує також можливість задання прозорості кольору за допомогою альфа-компоненти – четвертої складової у форматі *ARGB*-кольору.

4.4 Створення ліній

Простір імен *System.Drawing* містить велику кількість класів для створення в програмах стандартної графіки та спеціальних ефектів. Розглянемо детально клас *Graphics*, який надає методи та властивості для рисування ліній різних стилів.

Спочатку необхідно створити об'єкт *Graphics*:

Dim G As Graphics = CreateGraphics()

В цьому випадку всі подальші рисунки будуть створені на головній формі *Form1*. Якщо ж для графічних фігур має бути використана інша поверхня (наприклад, *Form2*), тоді її необхідно вказати:

Dim G As Graphics
G =Form2. CreateGraphics()

Далі має бути один із графічних інструментів. Для рисування ліній і контурних фігур використовується об'єкт класу *Pen* (Перо). Можна використати системне перо із набору *Pens* або створити власне перо. Системне перо надає лише пера обмеженого набору кольорів з одиничною товщиною лінії. Для власного пера можна задати будь-який *RGB*-колір і товщину лінії, однак потрібно спочатку оголосити його. Наприклад, перо червоного кольору з товщиною лінії 3 пікселі буде створено таким оператором:

Dim MyPen As New Pen(Color.Red,3)

Для створення прямих ліній використовується метод *DrawLine* класу *Graphics*, який дозволяє нарисувати відрізок лінії по двох точках. Можна задати пару координат початкової *x1%, y1%*, та кінцевої *x2%, y2%* точок:

DrawLine(pen As Pen, x1%, y1%, x2%, y2%),

або початкову *pt1* та кінцеву *pt2* точки як елементів структури *Point*:

DrawLine(pen As Pen, pt1 As Point, pt2 As Point)

За замовчуванням всі нарисовані лінії будуть суцільними із прямокутними краями. Однак, можна задати власний стиль лінії, а також стиль оформлення її країв. Для задання стилю лінії використовується властивість *DashStyle* простору імен *System.Drawing.Drawing2* (табл.4.2). Для створення власного виду лінії використовується властивість *DashPattern* класу *Pen*, яка використовує масив дійсних чисел, що позначають довжини почергово розташованих відрізків і пропусків. Наприклад, лінія *Custom* із табл. 4.2 задається таким програмним кодом:

Dim pen As New Pen(Color.Black, 2)
Dim dp As Single()={1,1,3,1,2,1,1,1,3,1,2}
Pen.DashPattern()=dp

Для вказування відстані *n* від початку лінії, з якої розпочинається штриховка, використовується властивість *DashOffset* класу *Pen:*

DashOffset=n

Оформлення країв ліній здійснюється за допомогою властивості *EndCap*, яка задає завершення лінії, і властивості *StartCap*, яка розпочинає

лінію. Наприклад, якщо початок лінії прямокутний, а кінець закінчується стрілкою, тоді необхідно записати таким чином:

StartCap=Square
EndCap=ArrowAnchor

Таблиця 4.2 – Стилі лінії

Макрос	Тип лінії пера
Solid	Суцільна лінія
Dash	Пунктирна лінія (рівномірні відрізки)
DashDot	Штрихпунктирна лінія
DashDotDot	Штрихпунктирна лінія
Dot	Точкова лінія
Custom	Задається властивістю *DashPattern*

Оформлення пунктирних ліній здійснюється за допомогою властивості *DashCap*. Всі вказані властивості оформлення країв ліній належать простору імен *System.Drawing.Drawing2*.

4.5 Створення базових контурних фігур

Окрім ліній клас *Graphics*, надає методи та властивості для рисування прямокутників, полігонів, еліпсів та інших геометричних фігур.

Створити прямокутник можна кількома способами. По-перше, прямокутник буде нарисовано за допомогою методу *DrawRectangle*, якщо заранні визначити для нього перо *Pen* і вказати координати *x* і *y* лівого верхнього кута , а також його ширину *width* та висоту *height:*

```
DrawRectangle(pen As Pen, x As Integer, y As Integer, _
                                    Width As Integer, height As Integer)
```

По-друге, прямокутник можна нарисувати, задавши його параметри через структуру *Rectangle:*

```
Dim rect As New Rectangle(x As Integer, y As Integer, _
                                    width As Integer, height As Integer)
DrawRectangle(pen As Pen, rect As Rectangle)
```

Якщо необхідно нарисувати декілька прямокутників, тоді можна скористатись методом *DrawRectangles*, який використовує масив прямокутників *rects()*, що утворюють послідовність:

```
DrawRectangles(pen As Pen, rects() As Rectangle)
```

Для рисування еліпса необхідно задати параметри прямокутника, в який вписується еліпс. Тому програмно окремий еліпс, подібно прямокутнику, також можна задати двома способами:

DrawEllipse(pen As Pen, x As Integer, y As Integer, _
Width As Integer, height As Integer x

або

DrawEllipse(pen As Pen, rect As Rectangle)

Варто відмітити, що координати і розміри прямокутника та еліпса можуть бути також і дійсними числами типу *Single*.

Створити трикутник, ромб або будь-який інший багатокутник можна, звичайно, нарисувавши його по лініях за допомогою методу *DrawLine*. Однак для подібних фігур передбачено спеціальний метод *DrawPolygon*, що має такий синтаксис:

DrawPolygon(pen As Pen, points() As Point),

де *points()* – масив точок структури *Point* або *PointF*, який задає вершини багатокутника.

Наступний фрагмент програми показує рисування прямокутника системним червоним пером, еліпса власним зеленим пером і трикутника власним синім пером.

```
Dim G As Graphics
G = Me.CreateGraphics
G.DrawRectangle(Pens.Red, 200, 220, 140, 50)
Dim MyColor As Color = Color.FromArgb(0, 255, 0)
Dim Pen2 As New Pen(MyColor, 2)
G.DrawEllipse(Pen2, 220, 50, 100, 150)
Dim Pen3 As New Pen(Color.Blue, 3)
Dim points As Point() = {New Point(20, 120), New Point(66, 300), _
        New Point(154, 300)}
G.DrawPolygon(Pen3, points)
```

Результат виконання програми показано на рис. 4.1.

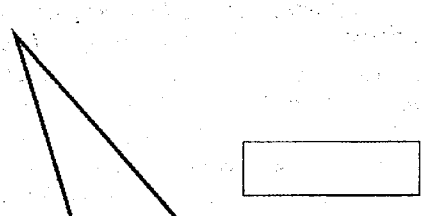

Рисунок 4.1 – Приклад базових контурних фігур

4.6 Створення складних контурних фігур

Розглянемо спочатку створення спеціальних кривих – сплайнів. Найбільш відомими представниками сплайнів є криві Без'є, які являють собою гладкі криві, що визначаються чотирма точками (початковою, кінцевою та двома контрольними). Крива Без'є обов'язково проходить через початкову і кінцеву точку та недалеко від контрольних точок. Ці контрольні точки задають форму кривої, тому вони мають розташовуватись між двома крайніми точками, але не на одній лінії з ними.

Криві Без'є будуються за допомогою методів *DrawBezier* та *DrawBeziers*, які мають такий синтаксис:

```
DrawBezier(pen As Pen, point1 As Point, point2 As Point, _
                       point3 As Point, point4 As Point)
DrawBezier(pen As Pen, x1 As Single, y1 As Single, _
                       x2 As Single, y2 As Single, _
                       x3 As Single, y3 As Single, _
                       x4 As Single, y4 As Single)
DrawBeziers(pen As Pen, points() As Point)
```

де *x1,y1,x2,y2,x3,y3,x4,y4* – координати точок структури *Point* або *PointF*;

point1, point2 , point13, point4 - точки структури *Point* або *PointF;*

points() – масив точок структури *Point* або *PointF.*

Розглянемо приклад програми для формування масиву кривих Без'є, результат виконання якої показано на рис 4.2.

```
Dim G As System.Drawing.Graphics
G = Me.CreateGraphics
Dim i, j As Integer
Dim points As Point() = {New Point(10, 70), _
                         New Point(30, 170), New Point(80, 170), _
                         New Point(100, 20)}
For i = 0 To 20
    For j = 0 To 3
       points(j).Offset(5, 2)
    Next j
       G.DrawBeziers(Pens.BlueViolet, points)
Next i
```

Крива Без'є завжди є незамкнутою фігурою. Можна також рисувати і замкнуті сплайни не вказуючи початкової точки. Для цього використовується метод *DrawClosedCurve*, в якому задається масив із трьох точок *points()*:

```
DrawClosedCurve(pen As Pen, points() As Point)
```

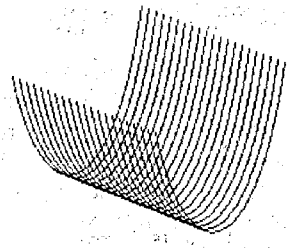

Рисунок 4.2 – Криві Без'є

Найскладнішим об'єктом із усіх раніше розглянутих є шлях, який може включати в себе лінії, фігури та інші елементи. Для побудови даного об'єкта використовується метод *DrawPath*:

DrawClosedCurve(pen As Pen, path As GraphicsPath),

де *path* – об'єкт, що задає параметри шляху.

Задати параметри шляху можна за допомогою класу *GraphicsPath* простору імен *System.Drawing.Drawing2*. Цей клас створюється конструктором, в якому вказуються масив *points()* всіх точок шляху, масив типів *types()* кожної точки із масиву *points()* (вибирається з табл. 4.3) та необов'язковий параметр *fillMode* заповнення шляху:

Dim GraphicsPath As New GraphicsPath(points() As Point, _
types() As Byte, fillMode As FillMode)

Таблиця 4.3 – Типи шляху

Параметр масиву *types()*	**Пояснення**
Bezier	Задає точку кривої Без'є
Bezier3	Задає точку кубічної кривої Без'є
CloseSubPath	Вказує кінцеву точку шляху
DashMode	Задає пропуск
Line	Задає лінію
PathMarker	Задає маркер шляху
PathTypeMask	Задає маску шляху
Start	Вказує початкову точку шляху

Шлях можна створювати різними способами: додаючи різні геометричні фігури і лінії, за допомогою задання точок шляху або об'єднуючи обидва варіанти. При створенні шляху по точках необхідно врахувати, що тип точки задає вид лінії від попередньої точки до заданої. Якщо такою лінією є крива Без'є, тоді необхідно задати три точки (або чотири точки в тому випадку, коли крива Без'є розпочинає шлях). Наступна програма будує шлях по точках для фігури на рис. 4.3.

```
Dim G As Graphics
G = Me.CreateGraphics
Dim Pen1 As New Pen(Color.Red, 1)
Dim points As PointF()={New PointF(20, 120), New PointF(66, 300), _
                        New PointF(154, 300), New PointF(200, 120), _
                        New PointF(20, 120)}
Dim types() As Byte = {Drawing2D.PathPointType.Start, _
                       Drawing2D.PathPointType.Bezier, _
                       Drawing2D.PathPointType.Bezier, _
                       Drawing2D.PathPointType.Bezier, _
                       Drawing2D.PathPointType.Line}
Dim Path As New Drawing2D.GraphicsPath(points, types)
G.DrawPath(Pen1, Path)
```

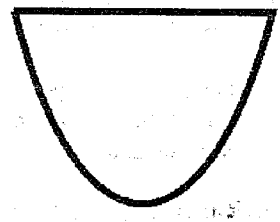

Рисунок 4.3 – Фігура, створена за допомогою побудови шляху

4.7 Зафарбовування фігур

Для зафарбовування фігур використовуються класи *SolidBrush* і *TextureBrush* простору імен *System.Drawing* та класи *HatchBrush, LinearGradientBrush* і *PathGradientBrush* простору імен *System.Drawing. Drawing2* (табл. 4.4).

Таблиця 4.4 – Класи для зафарбовування фігур

Клас	Пояснення
SolidBrush	Суцільне зафарбовування фігур
TextureBrush	Зафарбовування фігур на основі заданного зображення
HatchBrush	Зафарбовування фігур різними видами штриховки
LinearGradientBrush	Зафарбовування фігур на основі градієнтного розподілу кольорової гами вздовж прямої лінії
PathGradientBrush	Зафарбовування фігур на основі градієнтного розподілу кольорової гами вздовж довільної лінії

Всі вказані вище класи є похідними від базового класу *Brush*. Операція зафарбовування передбачає використання спеціального графічного інструмента – пензля. Кожний спосіб зафарбовування здійснюється своїм пензлем.

Для суцільного зафарбовування створюється пензель конструктором класу *SolidBrush*. Це найпростіший пензель, оскільки для його створення необхідно задати лише колір *color*:

Dim brush As New SolidBush(color as Color)

Розглянемо тепер способи зафарбовування геометричних фігур.

Для зафарбовування прямокутника і сукупності прямокутників використовуються відповідно метод *FillRectangle* і метод *FillRectangles* класу *Graphics* простору імен *System.Drawing*. Ці методи мають такий синтаксис:

FillRectangle(brush As Brush, x As Integer, y As Integer, _
width As Integer, height As Integer)
FillRectangle(brush As Brush, rect As Rectangle)
FillRectangles(brush As Brush, rects() As Rectangle)

Призначення параметрів *x, y, width, height, rect* та *rects()* у наведених методах такі ж самі, як і для методів *DrawRectangle* та *DrawRectangles*.

Зафарбовувати еліпс можна двома способами за допомогою методу *FillEllipse* класу *Graphics:*

FillEllipse(brush As Brush, x As Integer, y As Integer, _
width As Integer, height As Integer)
FillEllipse(brush As Brush, rect As Rectangle)

Зафарбовувати багатокутник також можна двома способами за допомогою методу *FillPolygon* класу *Graphics:*

FillPolygon(brush As Brush, points() As Point)
FillPolygon(brush As Brush, points() As Point, _
Fillmode As FillMode)

де *points()* – масив точок структури *Point* або *PointF*, який задає вершини багатокутника.

fillmode – параметр, що визначає спосіб заповнення фігури.

Для зафарбовування замкнутої геометричної фігури, побудованої методом шляху, використовується метод

FillPath(brush As Brush, path As GraphicsPath)

Складнішим способом зафарбовування є штрихове зафарбовування, оскільки вимагає задання трьох параметрів: виду штриховки *hatchStyle*, кольору штриховки *foreColor* та кольору фону *backColor*:

Dim brush As New HatchBrush(hatchStyle As HatchStyle, _
foreColor As Color, backColor As Color)

Якщо останній параметр не вказати, тоді як фон буде використовуватись чорний колір. Деякі види штриховки наведені в табл. 4.5.

Таблиця 4.5 – Види штриховки

Параметр *foreColor*	Пояснення
BackwardDiagonal	діагональна штриховка
Cross	штриховка хрест-навхрест
Divot	штриховка точками

Розглянемо програмну реалізацію видів штриховки, які наведені у табл. 4.5.

```
Dim G As Graphics
G = Me.CreateGraphics
Dim Brush1 As New _
Drawing2D.HatchBrush(Drawing2D.HatchStyle.BackwardDiagonal, _
        Color.Red, Color.BlanchedAlmond)
G.FillEllipse(Brush1, 20, 30, 100, 50)
Dim Brush2 As New _
        Drawing2D.HatchBrush(Drawing2D.HatchStyle.Cross, _
        Color.Red, Color.BlanchedAlmond)
G.FillEllipse(Brush2, 20, 100, 100, 50)
Dim Brush3 As New _
        Drawing2D.HatchBrush(Drawing2D.HatchStyle.Divot, _
        Color.Red, Color.BlanchedAlmond)
G.FillEllipse(Brush3, 20, 170, 100, 50)
```

Результат роботи програми показано на рис. 4.4.

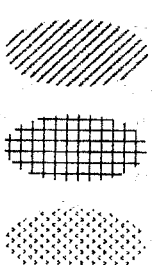

Рисунок 4.4 – Приклади штриховки

4.8. Виведення тексту

Інтерфейс *GDI+* дозволяє розміщувати текст на формі та інших компонентах. При виведення тексту необхідно в першу чергу визначити

стиль, тип і розмір шрифту. Для задання цих параметрів призначається клас *Font* простору імен *System.Drawing*.

Для створення об'єктів цього класу використовується велика кількість конструкторів. Розглянемо деякі з них.

New(prototype As Font, newStyle As FontStyle)
New(family As String, Size As Single, Style As FontStyle)
New(family As String, Size As Single, unit As GraphicsUnit)

де *prototype* – шрифт, на основі якого створюється новий;
newStyle – стиль нового шрифту, який може приймати значення із *FontStyle: Bold, Italic, Regular, Strikeout, Underline;*
family – сімейство шрифтів, на основі якого створюється новий шрифт;
Size – розмір шрифту;
newStyle – стиль шрифту, який може приймати значення із *FontStyle* (за замовчуванням задається звичайний шрифт, тобто *Regular*);
unit – одиниця виміру, яка використовується при заданні шрифту класу *Graphics* простору імен *System.Drawing*. Цей метод має такий синтаксис:

DrawString(str As Srting, font As Font, brush As Brush, _
point As PointF, format As StringFormat)

або

DrawString(str As Srting, font As Font, brush As Brush, _
x As Single, y As Single, format As StringFormat)

де *str* – текст, що виводиться;
font – об'єкт *Font*, який задає стиль і розмір шрифту;
brush – об'єкт *Brush*, який задає колір і текстуру тексту;
point – точка, яка задає лівий верхній кут початку тексту;
x,y – координати точки, яка задає лівий верхній кут початку тексту;
format – додаткові параметри тексту.

Додаткові параметри тексту можна настроїти за допомогою класу *StringFormat* простору імен *System.Drawing*. Цей клас має ряд властивостей, які визначають розташування тексту на формі. Зокрема, властивість *Alignment* задає спосіб вирівнювання тексту і може приймати такі значення: *Center* (по центру), *Near* (по правому краю), *Far* (по лівому краю). За замовчуванням текст виводиться горизонтально, зліва направо. Як приклад, розглянемо виведення на форму рядка тексту "*Visual Basic*".

```
Dim G As Graphics
G = Me.CreateGraphics
Dim Pen1 As New Pen(Color.Red, 1)
Dim Brush1 As New SolidBrush(Color.Green)
Dim Font As New Font("Arial", 12, FontStyle.Bold)
Dim s As String = "Visual Basic"
```

```
Dim StringFormat As New StringFormat
StringFormat.Alignment() = StringAlignment.Near
G.DrawString(s, Font, Brush1, New PointF(30, 150), StringFormat)
```

4.9 Виведення растрових зображень

Перед виведенням растрового зображення на форму необхідно спочатку створити об'єкт *Image1* одним із таких способів.

Спосіб 1. Отримати зображення із файла з іменем *<filename>* використовуючи метод *FromeFile*:

```
Dim Image1 As Image = Image.FromeFile(filename)
```

Спосіб 2. Отримати зображення із файла з іменем *<filename>* використовуючи констуктор класу *Bitmap*:

```
Dim Image1 As New Bitmap(filename)
```

Спосіб 3. Створити спочатку зображення в памяті, а потім вивести його на екран:

```
Dim Image1 As New Bitmap(x, y)
Dim G As Graphics = Graphics.FromImage(Image1)
```

Далі зображення необхідно вивести на форму використовуючи метод *DrawImage* класу *Graphics*. Зображення може бути виведено в прямокутник або в паралелограм. Існує декілька варіантів задання розташування цієї фігури. Наприклад, можна задати точку *point*, яка визначає лівий верхній кут такого прямокутника:

```
DrawImage(image As Image, point As Point),
```

або задати параметри прямокутника через структуру *Rectangle:*

```
Dim rect As New Rectangle(x As Integer, y As Integer, _
                          width As Integer, height As Integer)
DrawImage(image As Image, rect As Rectangle)
```

При виведенні зображення на форму можна змінювати його початкові розміри. Закінчений фрагмент програми для виведення зображення із файла на форму у прямокутник із одночасним зменшенням вдвічі геометричних розмірів зображення (рис 4.5) буде таким:

Рисунок 4.5 – Виведення растрового зображення на форму

```
Dim G As System.Drawing.Graphics
G = Form2.CreateGraphics
Dim myImage As Image = Image.FromFile("c:\Zakat.jpg")
Dim w As Single = myImage.Width, h As Single = myImage.Height
Dim rect As New Rectangle(100, 150, w / 2, h / 2)
G.DrawImage(myImage, rect)
```

Порядок виконання роботи

1. Для заданого варіанта нарисувати контурні фігури з використанням об'єктів класу *Pen*.
2. Зафарбувати замкнуті контурні фігури різними видами штриховки з використанням об'єктів класів, які є нащадками базового класу *Brush*.
3. Нарисувати довільну графічну фігуру як шлях, що складається з різних стандартних елементів.
4. Вивести на форму текст із заданими параметрами.
5. Вивести на форму статичне растрове зображення різними способами.
6. Вивести на форму динамічне растрове зображення, яке може рухатись по формі в заданих напрямках.
7. Створити проект із двома формами, в першій з яких в діалоговому режимі здійснюється вибір геометричних фігур, що мають бути нарисовані на другій формі.

Контрольні запитання

1. Розкажіть про організацію класів графічної бібліотеки *GDI+*.
2. Поясніть графічну систему координат у *Visual Basic 2005/2008*.
3. Як задаються кольори у векторній графіці?
4. Які типи ліній можна створити за допомогою класу *Graphics*?
5. Які контурні фігури можна створити за допомогою класу *Graphics?*
6. За допомогою яких методів можна нарисувати криві Без'є ?
7. Які можливості для векторної графіки надає шлях?
8. Які існують способи зафарбовування фігур?
9. Як зафарбовувати замкнуту геометричну фігуру, побудовану методом шляху?
10. Які параметри можна задати при виведенні тексту?
11. Як отримати зображення із файла?
12. Як можна вивести на форму растрове зображення?

ТЕМА № 5

Робота з файлами і потоками введення-виведення

Зміст теми. Основні операції з текстовими файлами за допомогою класів *FileSystem*, *File*, *FileInfo*, запис і зчитування даних з файлів за допомогою класів потоків, знайомство з діалоговими вікнами для пошуку файлів, робота з каталогами за допомогою класів *Directory* і *DirectoryInfo*, використання об'єкта *My* для роботи з файлами і каталогами.

Теоретичні відомості

5.1 Файли і потоки введення-виведення у *Visual Basic*

Більшість серйозних програм передбачає запис результатів своєї роботи у файли (*files*), а також читання початкових даних із файла. Програмний пакет *Visual Basic 2005/2008* передбачає всі необхідні можливості для роботи з файловою системою в середовищі *Windows*. При виконанні операцій введення-виведення використовуються також і нові принципи обробки даних – робота з потоками (*streams*) та сховищами даних. Потік введення-виведення являє собою дані, які переміщаються з одного місця в інше місце. Сховище даних – це місце, в якому дані зберігаються тривалий час.

Спочатку необхідно відмітити, що для роботи з файлами і потоками введення-виведення бібліотека класів *NET* використовує переважно класи із простору імен *System.IO*. Тому на початку кожної програми, що розглядається в цій темі має бути записано: *Imports System.IO*

З позицій обробки розрізняють три види файлів: текстові, типізовані та нетипізовані. Текстові файли складаються із рядків довільної довжини, які відділяються один від одного двома спеціальними символами: повернення каретки і переведення рядка. Типізовані файли складаються із набору записів однакової довжини. Нетипізовані файли – це файли, які не мають чітко вираженої логічної структури.

5.2 Основні операції з текстовими файлами за допомогою класів

5.2.1 Операції з файлами за допомогою класу *FileSystem*

Клас *FileSystem* дозволяє виконувати всі основні операції з текстовими файлами: створення файла, запис і зчитування текстових рядків, копіювання файла, встановлення атрибутів файла, отримання довідкової інформації тощо. В табл. 5.1 наведені основні методи цього класу. Цей клас зручний тим, що у програмах не потрібно створювати об'єкти цього класу і можна навіть не вказувати ім'я класу.

Таблиця 5.1 – Основні методи класу *FileSystem*

Методи	Призначення
FileOpen()	Відкриття файла
FileClose()	Закриття файла
FileCopy()	Копіювання файла
Rename()	Перейменування файла
EOF()	Перевірка досягнення кінця файла
Input()	Введення з файла списка значень
LineInput()	Введення із файла одного рядка
Print()	Запис у файл списка значень
PrintLine()	Запис у файл одного рядка
FileLen()	Визначення довжини файла
FileDateTime()	Визначення дати і часу створення файла

Перш ніж працювати з файлом, його необхідно відкрити, тобто зв'язати з якимось цілим числом – файловим номером і вказати режим роботи. Надалі файловий номер буде заміщати у програмі фізичне ім'я файла на диску. Відкриття файла здійснюється за допомогою методу

FileOpen(filenumber, filename, mode)

де *filenumber* – файловий номер;
filename –фізичне імя файла на диску;
mode – режим роботи, який може мати значення *OpenMode.Input* (для читання), *OpenMode.Output* (для запису), *OpenMode.Append* (для додання в кінець файла).

Після закінчення роботи з файлом його необхідно закрити за допомогою методу

FileClose(filenumber).

Якщо файл відкрито в режимі запису, тоді можна використати методи *Print()* або *PrintLine()* для запису рядка із змінної *var* у файл з номером *filenumber*:

Print(filenumber, var),
PrintLine(filenumber, var).

Метод *PrintLine()* відрізняється від методу *Print()* тим, що вставляє після значення *var* пару символів – повернення каретки і переведення рядка. Приклад програми зчитування текстових рядків з компонента *ListBox1* і запис їх у файл.

```
Dim i As Integer
Dim filename As String
filename = "c:\File9.txt"
FileOpen(1, filename, OpenMode.Output)
For i = 0 To ListBox1.Items.Count - 1
        PrintLine(1, ListBox1.Items(i))
```

```
Next
TextBox1.Text = filename
        FileClose(1)
```

Приклад програми зчитування текстових рядків з файла і відображення їх на формі у компоненті *ListBox1*:

```
Dim str, filename As String
filename = "c:\\File3.txt"
ListBox1.Items.Clear()
FileSystem.FileOpen(1, filename, OpenMode.Input)
While Not EOF(1)
   str = LineInput(1)
   ListBox1.Items.Add(str)
End While
TextBox1.Text = filename
MsgBox(FileSystem.FileDateTime(filename))
        FileClose(1)
```

Метод *EOF(filenumber)* повертає *True*, якщо досягнуто кінець файла з номером *filenumber*, і *False* в протилежному випадку.

5.2.2 Операції з файлами за допомогою класів *File* та *FileInfo*

В таблиці 5.2 наведені основні властивості класу *FileInfo* (клас *File* не має властивостей), а в таблиці 5.3 – основні методи обох класів. Як видно із наведених таблиць ці класи схожі за своїми можливостями, хоча вирішують однакові задачі по-різному.

Перша відмінність полягає в тому, що що клас *File* надає статичні методи для роботи з файлами, а клас *FileInfo* – динамічні.

Таблиця 5.2 – Основні властивості класу *FileInfo*

Властивості класу *FileInfo*	**Призначення**
Name	Ім'я файла
FullName	Повне ім'я файла
Attributes	Атрибути файла
CreationTime	Дата і час створення файла
LastAccessTime	Дата і час останнього доступу до файла
LastWriteTime	Дата і час останнього запису у файл
Length	Довжина файла в байтах
Directory	Ім'я батьківського каталогу
DirectoryName	Повне ім'я батьківського каталогу
Exists	Підтвердження існування файла (*True/False*)

Таблиця 5.3 – Основні методи класів *File* та *FileInfo*

Методи класу *File*	Методи класу *FileInfo*	Призначення
Create()	*Create()*	Створення файла
CreateText()	*CreateText()*	Створення текстового файла
Exists()		Перевірка існування файла
Open()	*Open()*	Відкриття файла
OpenRead()	*OpenRead()*	Відкриття файла для читання
OpenText()	*OpenText()*	Відкриття файла для читання тексту
OpenWrite()	*OpenWrite()*	Відкриття файла для запису
ReadAllText		Читання всього файла
WriteAllText		Запис всього тексту у файл
Copy()	*CopyTo()*	Копіювання файла
Delete()	*Delete()*	Вилучення файла
Move()	*MoveTo()*	Переміщення файла
GetAttributes()		Отримання атрибутів файла
GetCreationTime()		Отримання часу створення файла
GetLastAccessTime()		Отримання часу останнього доступу до файла
GetLastWriteTime()		Отримання часу останнього запису у файл

Нагадаємо, що статичними є ті методи класу, які можуть виконуватись без попереднього створення об'єктів класу.

По-друге, встановити та отримати інформацію про різні характеристики файла можна або за допомогою властивостей класу *FileInfo*, або за допомогою методів класу *File*.

Основні операції з цілісними файлами (створення, копіювання, переміщення, вилучення) дозволяють виконувати обидва класи, однак існують ряд методів, які належать тільки одному із цих класів.

Для відкриття файла використовується метод *Open()* класу *File*:

Open(path As String, mode, access, share),

де *path* – фізичне ім'я файла на диску;

mode – режим роботи, який може мати значення *Append* (для додання в кінець файла), *Create, CreateNew* (створення нового файла), *Open* (відкриття файла), *OpenOrCreate* (відкриття або створення файла);

access – тип доступу до файла, який може мати значення *Read* (читання), *Write* (запис), *ReadWrite* (читання і запис);

share – тип дозволу для доступу до файла іншим процесам, який може мати значення *None* (немає доступу), *Read* (читання), *ReadWrite* (читання і запис); *Write* (запис).

Приклад програми введення текстових рядків з заданого файла і відображення їх на формі у компоненті *TextBox1* за допомогою методів класу *File:*

```
Dim filename, AllText As String
    filename = "c:\File3.txt"
    If File.Exists(filename) Then
        File.Open(filename, FileMode.Open, FileAccess.Read,
                FileShare.Read)
        AllText = File.ReadAllText(filename)
        TextBox1.Text = AllText
    Else
            MsgBox("File not exists")
    End If
```

Спочатку метод *ReadAllText()* копіює весь вміст текстового файла у рядкову змінну *AllText,* а потім дані передаються у *TextBox1.* Це займає менше часу, ніж читати файл по окремих рядках за допомогою методу *LineInput()* класу *FileSystem.*

Нагадаємо, що для відображення багаторядкового тексту у *TextBox1* необхідно для цього компонента встановити властивість *MultiLine=True.*

5.3 Операції з файлами за допомогою потоків введення-виведення

5.3.1 Основні потоки введення-виведення бібліотеки класів *NET*

Введення-виведення даних в середовищі *NET* основано на двох взаємопов'язаних поняттях: потоках введення-виведення (*stream*) і сховищах даних. Потік введення-виведення є послідовністю даних, а сховище є те місце, в якому дані можуть зберігатись: файлі, області оперативної памяті, адресі в Інтернеті.

Для роботи з потоками бібліотека класів *NET* використовує такі класи:

- *Stream;*
- *FileStream;*
- *NetworkStream;*
- *MemoryStream;*
- *BufferedStream;*
- *CryptoStream.*

Клас *Stream* – це абстрактний клас, від якого успадковані всі інші класи-потоки. Клас *FileStream* описує потік для читання та запису файлів на диску, а також інших об'єктів, для яких операційна система надає дескриптори файлів, наприклад, для клавіатури і дисплея. Клас *NetworkStream* надсилає та отримує дані іншому процесу, який

знаходиться за заданою мережевою адресою, що задається іменем хоста та номером порту. Потік класу *MemoryStream* можна подати як потік, у якого сховищем даних виступає область оперативної памяті, а дані передаються між процесами. Клас *BufferedStream* використовує при передаванні даних додаткову буферізацію, а клас *CryptoStream* – потокове шифрування.

Клас *Stream* та всі згадані нащадки цього класу працюють з нетипізованими даними. Такий підхід зручний лише при переміщенні великих масивів даних з одного місця в інше.

Якщо необхідно працювати з даними всередині потоку, тоді варто звернутись до спеціальних класів, які читають та записують типізовані дані. Їх зручно згрупувати за такими парами:

– *BinaryReader і BinaryWriter;*
– *TextReader і TextWriter;*
– *StreamReader і StreamWriter;*
– *StringReader і StringWriter.*

В типізованих файлах за допомогою методу *Seek()* можна встановити покажчик в будь-яку позицію всередині файла і далі виконувати операції читання і запису з цієї позиції.

5.3.2 Класи *BinaryReader* і *BinaryWriter*

Класи *BinaryReader* і *BinaryWriter* надають можливість записувати і зчитувати задану кількість байтів даних. Цей процес відбувається без втрат. Іншими словами, дані, які записані за допомогою об'єкта класу *BinaryWriter*, а потім прочитані об'єктом класу *BinaryReader* матимуть той же самий вигляд (при умові, що при записі і читанні використовується однаковий тип даних). Особливістю цих файлів є те, що вони зберігаються у двійковому форматі і тому непридатні для перегляду у текстових редакторах. В табл. 5.4 наведені методи класу *BinaryWriter*, а в табл. 5.5 – деякі основні методи класу *BinaryReader*.

Таблиця 5.4 – Методи класу *BinaryWriter*

Властивості	Призначення
Close()	Закрити потік
Flush()	Записати дані з буфера в сховище даних потоку
Seek()	Встановити позицію в потоці
Write()	Записати дані в потік

Наведемо приклад програми створення бінарного файла за допомогою класу *File* і запис у нього трьох змінних різного типу (*Int32, Double, String*) за допомогою відповідних методів класу *BinaryWriter*.

Таблиця 5.5 – Основні методи класу *BinaryReader*

Методи	Призначення
ReadBoolean()	Читання даних типу *Boolean* із потоку
ReadChar()	Читання символа із потоку
ReadString()	Читання рядка із потоку
ReadByte()	Читання байта із потоку
ReadInt16()	Читання 2-байтового знакового цілого числа із потоку
ReadInt32()	Читання 4-байтового знакового цілого числа із потоку
ReadInt64()	Читання 8-байтового знакового цілого числа із потоку
ReadSingle()	Читання даних типу *Single* із потоку
ReadDouble()	Читання даних типу *Double* із потоку
ReadDecimal()	Читання даних типу *Decimal* із потоку

```
Dim BinaryWriter As New BinaryWriter(File.Create("E:\\File1.bin"))
Dim a As Integer = 345
Dim b As Double = 75.82
Dim str As String = "word"
BinaryWriter.Write(a)
BinaryWriter.Write(b)
BinaryWriter.Write(str)
BinaryWriter.Flush()
BinaryWriter.Close()
```

Приклад програми зчитування з заданого файла трьох змінних різного типу (*Int32, Double, String*) за допомогою відповідних методів класу *BinaryReader* та відображення їх на формі у компонентах *TextBox1, TextBox2* і *TextBox3:*

```
Dim BinaryReader As New BinaryReaderFile.OpenRead("E:\\File1.bin"))
Dim a As Integer, b As Double, str As String
a = BinaryReader.ReadInt32
TextBox1.Text = a
b = BinaryReader.ReadDouble
TextBox2.Text = b
str = BinaryReader.ReadString
TextBox3.Text = str
BinaryReader.Close()
```

5.3.3 Класи *StreamReader* і *StreamWriter*

Розглянуті вище класи *BinaryReader* і *BinaryWriter* зручні для роботи з файлами, які містять довільні дані. Для роботи із текстом

призначені інші класи – *TextReader* і *TextWriter*. Однак ці класи не мають можливості працювати з потоками та сховищами. Тому на практиці більш корисними є класи *StreamReader* і *StreamWriter*, які можуть працювати з текстовими даними та потоками і сховищами.

В табл. 5.6 наведені деякі основні методи класу *StreamWriter*, а в табл. 5.7 – деякі основні методи класу *StreamReader*.

Таблиця 5.6 – Основні методи класу *StreamReader*

Методи	Призначення
Write()	Запис даних у відповідний об'єкт потоку
WriteLine()	Запис текстового рядка у відповідний об'єкт потоку
Flush()	Записати дані з буфера в сховище даних потоку
Close()	Закриття відповідного об'єкта і потоку

Таблиця 5.7 – Основні методи класу *StreamReader*

Методи	Призначення
Read()	Читання символів із відповідного об'єкта потоку
ReadLine()	Читання рядка символів із відповідного об'єкта потоку
ReadToEnd()	Читання символів із поточної позиції і до кінця відповідного об'єкта потоку
Peek()	Повернення наступного символа без зміни позиції
Close()	Закриття відповідного об'єкта і потоку

Розглянемо детальніше можливості цих класів для роботи із блоками тексту. Завдяки методу *Seek()* встановити покажчик в будь-яку позицію всередині файла для подальшого запису і зчитування частин рядків або окремих текстових рядків всередині файла. Формат методу *Seek()* такий:

Seek (offset As Long, origin As SeekOrigin) As long,

де *offset* – зміщення покажчика відносно позиції, заданої параметром *origin*;

origin – параметр, що визначає позицію, в якій може знаходитись покажчик: *SeekOrigin.Begin* (на початку файла), *SeekOrigin.Current* (в поточній позиції), *SeekOrigin.End* (в кінці файла).

Приклад програми зчитування заданого файла після перших 100 символів за допомогою методу *ReadToEnd()* класу *StreamReader* та відображення тексту на формі у компоненті *RichTextBox1*:

```
Dim StreamReader As New StreamReader("E:\\File33.txt")
StreamReader.BaseStream.Seek(100, SeekOrigin. Begin)
RichTextBox1.Text = StreamReader.ReadToEnd()
StreamReader.Close()
```

5.4 Діалогові вікна для пошуку файлів

Діалогове вікно відкриття файла призначено для пошуку файлів, які використовуються у програмі. Для створення цього вікна використовується ЕК *OpenFileDialog* (рис. 5.1). Зауважимо, що цей ЕК не відкриває файл, він лише надає можливість вибору необхідного файла. Деякі із властивостей ЕК *OpenFileDialog* наведені у табл. 5.8.

Таблиця 5.8 – Основні властивості ЕК *OpenFileDialog*

Властивості	Призначення
CheckFileExists	Перевірка існування файлів
CheckPathExists	Перевірка існування шляху до вибраного файла
FileName	Повне імя вибраного файла
Filter	Фільтр для задання типу відображуваних файлів
MultiSelect	Можливість вибору кількох файлів при значенні *True*
InitialDitectory	Задання початкового каталогу при виклику діалогового вікна

Приклад програми зчитування текстових файлів з вибором імені файла за допомогою ЕК *OpenFileDialog1:*

```
Dim AllText As String
OpenFileDialog1.Filter = "Text files(*.txt)|*.txt"
OpenFileDialog1.ShowDialog()
File.Open(OpenFileDialog1.FileName, FileMode.Open, _
        FileAccess.Read, FileShare.Read)
AllText = File.ReadAllText(OpenFileDialog1.FileName)
RichTextBox1.Text = AllText
```

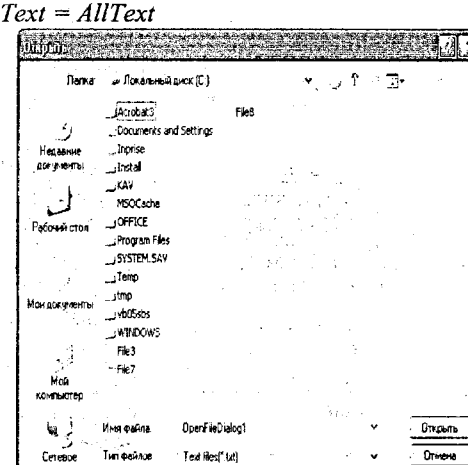

Рисунок 5.1 – Вікно ЕК *OpenFileDialog*

Для пошуку файлів, в яких мають бути збережені дані із програми використовується діалогове вікно збереження файла. Для створення цього вікна використовується ЕК *SaveFileDialog*. Даний елемент керування має аналогічні, як і в ЕК *OpenFileDialog*, властивості, за винятком того, що для відкриття файла задається вимога існування файла, а для вікна збереження файла ні (тобто різні за замовчуванням значення властивості *CheckFileExists*).

5.5 Робота з каталогами

Для операцій з каталогами (*directories*) передбачено два класи: *Directory* і *DirectoryInfo*. Ці класи схожі за своїми можливостями, хоча вирішують однакові задачі по-різному. Відмінності між ними дуже схожі на взаємні відмінності між класами *File* та *FileInfo*: клас *Directory* надає статичні методи для роботи з каталогами, а клас *DirectoryInfo* – динамічні. В табл. 5.8 наведені основні властивості класів *Directory* і *DirectoryInfo*.

Таблиця 5.8 – Основні методи класів *Directory* та *DirectoryInfo*

Методи класу *Directory*	**Методи і властивості класу *DirectoryInfo***	**Призначення**
CreateDirectory()	*Create()*	Створення каталогу
немає	*CreateSubdirectory()*	Створення підкаталогу
Delete()	*Delete()*	Видалення каталогу
GetDirectories()	*GetDirectories()*	Отримання списку підкаталогів вказаного каталогу
GetFiles()	*GetFiles()*	Отримання списку файлів вказаного каталогу
GetFileSystemEntries()	*GetFileSystemInfos()*	Отримання списку файлів і підкаталогів вказаного каталогу
Move()	*MoveTo()*	Переміщення каталогу
Exists()	*Exists* (власт.)	Перевірка на існування
немає	*Name* (власт.)	Отримання імені каталогу
GetParent()	*Parent* (власт.)	Отримання імені батьківського каталогу
GetDirectoryRoot()	*Root* (власт.)	Отримання імені кореневого каталогу
GetCurrentDirectory()	немає	Отримання імені поточного каталогу
SetCurrentDirectory()	немає	Задання імені поточного каталогу

Приклад програми створення каталогу *"C:\MyTemp"* та підкаталогу *"Temp1"* за допомогою класу *DirectoryInfo:*

```
Dim dirName As String = ("C:\\MyTemp")
Dim subdirName As String = ("Temp1")
Dim dirInfo As New DirectoryInfo(dirName)
If Not dirInfo.Exists Then
    dirInfo.Create()
    dirInfo.CreateSubdirectory(subdirName)
End If
```

Приклад програми видалення каталогу *"C:\MyTemp"* та всіх його підкаталогів і файлів за допомогою класу *DirectoryInfo:*

```
Dim dirName As String = ("C:\Temp")
Dim dirInfo As New DirectoryInfo(dirName)
If dirInfo.Exists Then
   dirInfo.Delete(True)
End If
```

Якщо у методі *Delete()* не вказувати параметр *True*, тоді видаляється лише пустий каталог.

Якщо необхідно перемістити каталог разом з усім його вмістом з одного місця в інше, тоді можна скористатися методом *MoveTo()* класу *DirectoryInfo* або методом *Move()* класу *Directory*. Ці методи мають такий формат:

```
Move(oldDirName As String, newDirName As String),
MoveTo( newDirName As String),
```

де *oldDirName* – повне ім'я каталогу, який необхідно перемістити,
newDirName – нове ім'я каталогу та повний шлях до нього.

Ці методи можна використати і для перейменування каталогу, такий приклад наведено далі:

```
Dim dirName1 As String = ("C:\\Temp")
Dim dirName2 As String = ("C:\\Visual")
Dim dirInfo As New DirectoryInfo(dirName1)
   If dirInfo.Exists Then
      dirInfo.MoveTo(dirName2)
End If
```

Розглянемо задачу виведення списку всіх файлів кореневого каталогу диска С у ЕК *ListBox1* двома способами. Відповідна програма з використанням класу *Directory* має такий вигляд:

```
Dim dirName As String = ("C:\\")
   If Directory.Exists(dirName) Then
      For Each str As String In Directory.GetFiles(dirName)
         ListBox1.Items.Add(str)
      Next str
   End If
```

Аналогічна програма з використанням класу *DirectoryInfo* має такий вигляд:

```
Dim dirName As String = ("C:\\")
Dim dirInfo As New DirectoryInfo(dirName)
Dim iles() As FileInfo = dirInfo.GetFiles("*.*")
If dirInfo.Exists Then
  For i As Long = 0 To files.GetUpperBound(0)
    ListBox1.Items.Add(files(i))
  Next i
End If
```

Як видно із останньої програми, клас *DirectoryInfo* дозволяє задати шаблон типів файлів, тобто можна вибрати лише бажані файли.

5.6 Використання об'єкта *My* для роботи з файлами і каталогами

Головна перевага об'єкта *My* – швидкість і простота доступу, до бібліотеки *.NET Framework* при виконанні багатьох сервісних задач. По суті, *My* – це сукупність великої кількості різних об'єктів, які ієрархічно об'єднані. Для роботи з файлами та каталогами існує об'єкт *My.Computer.FileSystem*, ряд методів якого наведені у табл. 5.9.

Таблиця 5.9 – Основні методи об'єкта *My.Computer.FileSystem*

Методи	Призначення
CopyDirectory()	Копіювання каталогу
CopyFile()	Копіювання файла
CreateDirectory()	Створення каталогу
DeleteDirectory()	Видалення каталогу
DeleteFile()	Видалення файла
DirectoryExists()	Перевірка існування каталогу
FileExists()	Перевірка існування файла
FindInFiles()	Повернення імен файлів, в яких знайдено заданий текст
GetDirectories()	Повернення імен підкаталогів заданого каталогу
GetFiles()	Повернення імен файлів заданого каталогу
GetParentPath()	Повернення повного імені батьківського каталогу
MoveDirectory()	Переміщення одного каталога в інший
MoveFile()	Переміщення або перейменування файла
RenameDirectory()	Перейменування каталогу
RenameFile()	Перейменування файла
ReadAllBytes()	Читання даних бінарного файла
ReadAllText()	Читання даних текстового файла
WriteAllBytes()	Запис даних у бінарний файл
WriteAllText()	Запис даних у текстовий файл

Приклад програми зчитування вмісту вибраного текстового файла за допомогою методу *ReadAllText()* об'єкта *My.Computer.FileSystem* та відображення його на формі у ЕК *RichTextBox1:*

```
Dim AllText$ = ""
OpenFileDialog1.Filter = "Text files(*.TXT)|*.TXT"
OpenFileDialog1.ShowDialog()
    AllText = _
    My.Computer.FileSystem.ReadAllText(OpenFileDialog1.FileName)
RichTextBox1.Text = AllText
```

Порядок виконання роботи

1. Створити програмно новий файл за допомогою класу *FileSystem* і записати в нього дані.
2. Прочитати програмно дані з файла за допомогою класу *FileSystem* і відобразити їх у компонентах *TextBox1* і *ComboBox1* на формі.
3. Здійснити програмно за допомогою класу *File* копіювання та переміщення окремих файлів.
4. Визначити програмно за допомогою класу *FileInfo* характеристики файла (імя, атрибути, дату створення, час створення та ін.).
5. Виконати програмно за допомогою класів *BinaryReader* і *BinaryWriter* операції запису у файл та зчитування із файла двійкових даних.
6. Виконати програмно за допомогою класів *StreamReader* і *StreamWriter* операції запису у файл та зчитування із файла текстових даних.
7. Виконати програмно операції з файлами з вибором імен файлів за допомогою компонентів *OpenFileDialog* та *SaveFileDialog.*
8. Виконати програмно основні операції з каталогами за допомогою класів *Directory* і *DirectoryInfo.*

Контрольні запитання

1. Які можливості для роботи з файлами надає клас *FileSystem*?
2. В чому полягає відмінність між класами *File* та *FileInfo* ?
3. Як прочитати текстові дані з файла і записати їх у файл за допомогою класів *File* та *FileInfo* ?
5. Що являють собою потоки введення-виведення і сховища у *Visual Basic 2005/2008* ?
4. Які можливості для роботи з потоками введення-виведення надають класи *BinaryReader* і *BinaryWriter* ?
5. Які можливості для роботи з потоками введення-виведення надають класи *StreamReader* і *StreamWriter* ?
6. Як програмується пошук файлів за допомогою діалогових вікон.

ТЕМА № 6

Робота з базами даних

Зміст теми. Знайомство із з новою технологією роботи з даними *ADO.NET,* основними об'єктами *ADO.NET,* програмуванням баз даних, відображенням даних на формі та формуванням *SQL*-запитів за допомогою майстра.

Теоретичні відомості

6.1 Нова технологія роботи з даними у Visual Studio

Завжди, коли виникає потреба у великих обсягах даних використовуються бази даних (*databases*). Під базою даних (БД), як правило, розуміють комп'ютеризованону систему збереження записів, основна залежності від способу подання даних розрізняють декілька типів БД. Найбільш поширеними є реляційні БД , в яких дані зберігаються у вигляді спеціальних двовимірних таблиць – реляційних таблиць. В останні роки отримують розповсюдження нереляційні БД, які можуть містити графіку, електронну пошту та іншу різноманітну інформацію.

З метою забезпечення дружнього інтерфейсу з БД створені спеціальні технології доступу до даних. Найголовніша задача таких технологій – забезпечення уніфікованого способу роботи з даними незалежно від конкретного формату БД.

Попередні версії *Visual Basic* підтримували такі технології роботи з даними:

– *DAO (Data Access Objects),*
– *RDO (Remote Data Objects),*
– *ADO (ActiveX Data Objects),*
– *ODBC (Open DataBase Connectivity).*

У *Visual Studio.NET 2002* вперше була представлена нова технологія роботи з даними – *ADO.NET* (*ActiveX Data Object.NET*) версії 1.1, яка замінила всі інші технології. У *Visual Studio 2005* використовується технологія *ADO.NET* версії 2.0.

Visual Studio 2005/2008 не призначена для створення нових БД – вона призначена для відображення, аналізу та керування інформацією із існуючих БД. Як і інші пакети візуального програмування, *Visual Studio 2005/2008* дозволяє працювати з різними форматами БД, причому ця різноманітність форматів ще більше зросла.

Універсалізм роботи з даними проявляється також і в тому, що *ADO.NET* призначений також для використання в середовищі Інтернет, що означає, що використовується єдиний метод доступу до локальних, клієнт-серверних і розміщених на сайтах Інтернету джерел даних. *ADO.NET*

підтримує не тільки реляційні БД, але також і нереляційні БД, що містять аудіо-, відео- і інші типи даних.

Внутрішнім форматом даних в *ADO.NET* є мова *XML* (*Extensible Markup Language*) – стандарт відомої організації *Word Wide Web Consortium* для опису даних. Це нова мова розмітки текстових даних, яка дозволяє передавати файли по протоколу *HTTP* через Інтернет. Дані у форматі *XML* передаються в текстовому вигляді, тому брандмауер їх вільно пропустить (чого не можна сказати про дані, що передаються з допомогою об'єктів *ADO*, оскільки в основі їх лежить технологія *COM*).

6.2 Подання даних в *ADO.NET*

В *ADO.NET* використовується від'єднані набори даних, над якими і проходят всі перетворення. З'єднання з реальною базою даних виконується тільки для приймання чи передавання даних.

Основним об'єктом для зберігання автономного набору даних є *DataSet*. Подання даних в об'єкті *DataSet* реалізовано у вигляді таблиць – об'єктів *DataTable*. В *DataTable* зберігаються стовпці, рядки, ключі, обмеження цілісності, інформація про зв'язки між таблицями. Можна сказати, що об'єкт *DataSet* являє собою спрощену реляційну БД із вбудованою підтримкою мови *XML*, яка зберігає розміщенний в пам'яті миттєвий знімок частин реальної БД, з якою працює програма.

Після завантаження даних в об'єкт *DataSet* з'єднання з джерелом може бути розірвано. Далі програма виконує обробку даних, після чого знову встановлює з'єднання з джерелом, і модифіковані дані знову передаються джерелу.

Зв'язок між БД і об'єктами *DataSet* виконується з допомогою провайдера (*provider*) – спеціального набору об'єктів. Види провайдерів, які підтримуються у *Visual Basic 2005/2008* наведені у табл. 6.1.

Таблиця 6.1 – Провайдери даних в *ADO.NET*

Провайдер даних	**Простір імен**	**Опис**
SQL Server	*System.Data.SqlClient*	Встановлення зв'язку з БД *MS SQL Server* (7.0 і вище)
OLE DB	*System.Data.OleDB*	Встановлення зв'язку з БД будь-якого формату, для яких є провайдер *OLE DB* (*MS Access* та інші)
Oracle	*System.Data.OracleClient*	Встановлення зв'язку з БД *Oracle*
ODBC	*System.Data.Odbc*	Встановлення зв'язку через драйвери *ODBC*

Об'єкти *SQL Server* мають префікс *Sql* (наприклад, *SqlConnection*), об'єкти *OleDB* – префікс *OleDb* (*OleDbConnection*) і т.д. Незважаючи на схожі базові принципи роботи, існують деякі відмінності у практичній реалізації провайдерів, тому не можна перейти від одного провайдера до іншого простим перейменуванням об'єктів, необхідно вносити зміни у код програми.

Для розробників БД на основі програмних продуктів *Microsoft* використовується два провайдери: *SQL Server* і *OLE DB*. З форматом даних *XML* працює лише перший провайдер, завдяки чому забезпечується максимальна швидкість обміну даними. Провайдер *OLE DB* менш ефективний з позицій продуктивності роботи, однак його перевагою є великий вибір типів форматів БД.

Більшість джерел даних підтримують обмежену кількість з'єднань, наприклад, БД *Access* можуть забезпечити одночасну роботу не більше ніж з 50 користувачами. Тому завдання програміста-розробника полягає в мінімізації терміну зв'язку з БД, оскільки з'єднання вимагає значних системних ресурсів.

6.3 Об'єктна модель *ADO.NET*

Розглянемо детальніше об'єкти, що входять до складу *ADO.NET*. На рис. 6.1 показана спрощена архітектура *ADO.NET*.

Почнемо з провайдера даних. Основними складовими провайдера даних є об'єкти *Connection, Command, DataAdapter* і *DataReader*.

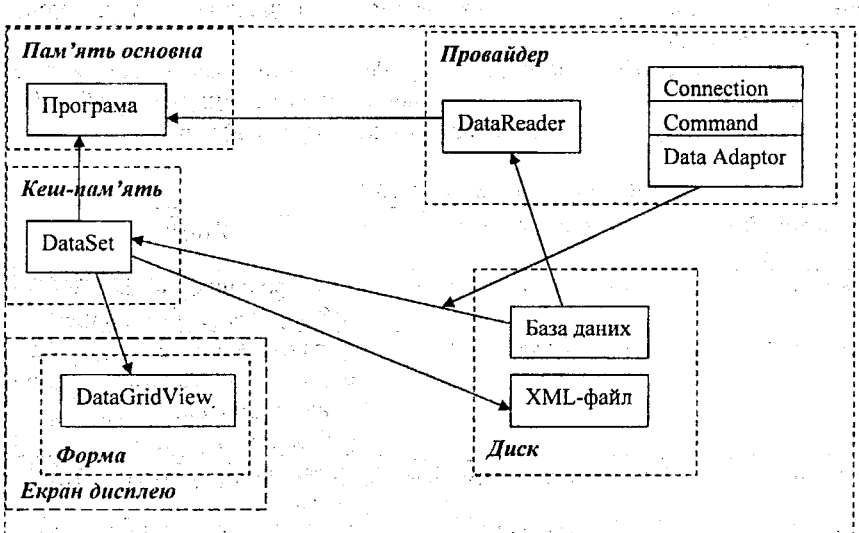

Рисунок 6.1 – Архітектура *ADO.NET*

Об'єкт *Connection* забезпечує безпосереднє з'єднання з джерелом даних. Об'єкт *Command* дозволяє керувати даними джерела, а також виконувати процедури, що зберігаються.

Об'єкт *DataAdapter* використовується для передавання даних між джерелом даних і об'єктом *DataSet* з допомогою *SQL*-команд. Для виконання окремих операцій з даними використовуються також 4 властивості: *SelectCommand, InsertCommand, UpdateCommand, DeleteCommand* відповідно для вибірки, додання, зміни та вилучення даних.

Об'єкт *DataReader* являє собою однонаправлений потік даних від джерела тільки на читання без повернення до уже прочитанних рядків. Об'єкт *DataReader* може замінити об'єкт *DataSet* в тих випадках, коли необхідно забезпечити спрощене передавання даних від джерела, наприклад, коли необхідний швидкий однократний доступ до даних без їх зміни. Це дозволяє підвищити швидкодію програми, оскільки не потрібно виділяти пам'ять для всієї таблиці на весь період функціонування програми, однак при цьому зберігається відкрите з'єднання з БД.

Об'єкт *DataSet* зберігає в собі набір таблиць з додатковою інформацією про їх структуру і зв'язками між ними. Цей об'єкт може працювати як з провайдерами даних, так і з даними в файлах формату *XML*. Об'єкт *DataSet* включає дві колекції:

– *DataTableCollection*, яка складається з таблиць, кожній із яких відповідає об'єкт *DataTable;*

– *DataRelationCollection*, яка містить інформацію про взаємозв'язки між таблицями.

Об'єкт *DataTable* складається з:

– колекції *DataColumnCollection*, що містить стовпці таблиці;

– колекції *DataRowCollection*, що містить рядки таблиці;

– колекції *ConstriantCollection*, що містить сукупність обмежень цілістності даних для стовпців.

У *Visual Basic 2005/2008* існує тільки один компонент, спеціально призначений для відображення таблиці із БД на формі – *DataGridView*. З його допомогою можна відображати будь-яку інформацію: текст, цифри, дати, масиви. Цей компонент дозволяє також редагувати дані в таблиці.

Щоб зв'язати об'єкт *DataGridView* з джерелом даних необхідно в його властивості *DataSource* вказати конкретний об'єкт *DataSet*. Якщо об'єкт *DataSet* містить кілька таблиць, тоді на екрані будуть відображені імена всіх таблиць, і користувачу залишиться лише вибрати конкретну таблицю.

Організувати всі зв'язки між розглянутими об'єктами можна як програмно, так і за допомогою майстра *Data Source Configuration Wizard*.

6.4 Створення запитів до БД за допомогою мови *SQL*

Найпоширенішим способом отримання інформації із реляційних БД є створення до них *SQL*-запитів (*SQL-queries*).

SQL (*Structured Query Language*) – скорочена назва структурованої мови запитів, яка дозволяє окрім читання даних ще і створювати, модифікувати і захищати БД. Незалежність від конкретних форматів окремих СУБД зробили цю мову стандартною мовою локальних і мережевих БД.

Нагадаємо основні терміни реляційних БД. Реляційна таблиця складається із набору полів (стовпців), одне або декілька з яких утворюють ключ. Ключові поля – це такі поля, сукупна інформація в яких не повинна повторюватись, оскільки пошук в таблиці відбувається саме за ключем. Для пошуку можуть задаватись додаткові критерії відбору на основі конкретних значень полів. Рядки в реляційній таблиці іноді називають ще записами або кортежами. Порядок розташування записів в БД довільний, але під час формування запиту відібрані записи можуть бути відсортовані визначеним способом.

Реляційна БД може складатись з кількох реляційних таблиць. Ці реляційні таблиці утворюються під час проектування БД методом нормалізації, тобто послідовного переходу від однієї чи кількох великих таблиць до менших таблиць. В правильно побудованій реляційній БД окремі таблиці об'єднуються тільки за спільними ключовими полями, тобто між кожною парою таблиць існує такий зв'язок по однойменних ключових полях, що дає можливість від однієї таблиці побудувати шлях до будь-якої іншої таблиці БД.

Розглянемо детально лише один оператор мови *SQL*, на основі якого і створюються всі можливі запити до БД – оператор *SELECT*. Синтаксис цього оператора в скороченій формі такий:

SELECT [ALL / DISTINCT] [список полів]
FROM [список таблиць БД]
WHERE [критерії відбору]
ORDER BY [поле, за яким здійснюється сортування]
HAVING [поле, за яким здійснюється сортування]

де

ALL – означає, що відбираються всі рядки, навіть однакові;

DISTINCT – означає, що відбираються тільки унікальні рядки;

Результатом виконання *SQL*-запиту по реляційній таблиці буде також таблиця – або частиною початкової таблиці або таблицею з новими даними, отриманими із початкових даних.

Наведемо кілька простих прикладів побудови *SQL*-запитів для реляційної БД "Замовник" (рис. 6.1).

Замовник: таблиця

Організація	Фірма	Адреса	Кількість
ВНТУ	Апекс	Хмельницьке шосе, 95	8
ВНМУ	Арго	Пирогова, 34	22
КНУ	Квазар-Мікро	Шевченка, 124	64
НАУ	Ліана	Комарова, 1	12
ХПІ	Ліана	Космонавтів, 8	12

Рисунок 6.1 – Приклад реляційної таблиці

Для отримання всієї таблиці, достатньо в *SQL*-запиті вказати тільки її назву:

*SELECT **
FROM Замовник

В більшості випадків цікава деяка вибіркова інформація, наприклад, назви організацій, які замовили кількість товарів, що перевищує 10, причому дані мають бути відсортованими за кількістю товару. Тоді *SQL*-запит буде складнішим:

SELECT Організація, Кількість
FROM Замовник
WHERE Кількість > 10
ORDER BY Кількість

Результат виконання цього *SQL*-запиту показано на рис. 6.2.

Організація	Кількість
ХПІ	12
НАУ	12
ВНМУ	22
КНУ	64

Рисунок 6.2 – Результат *SQL*-запиту

В *SQL*-запитах можна використовувати так звані агрегативні функції: *MAX* (максимальна величина), *MIN* (мінімальна величина), *SUM* (сума), *AVG* (середнє арифметичне), *COUNT* (кількість елементів) та інші.

Наприклад, для знаходження загальної кількості замовлених товарів необхідно записати:

SELECT SUM(Кількість) AS [Загальна кількість]
FROM Замовник

Результат виконання цього *SQL*-запиту показано на рис. 6.3.

У *Visual Basic 2005/2008* формувати *SQL*-запити можна двома способами: візуально за допомогою майстра *Data Source Configuration Wizard* або програмно.

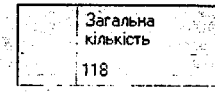

Рисунок 6.3 – Результат *SQL*-запиту

6.5 Програмування БД на основі провайдера *OLE DB*

Спочатку необхідно записати оператори *Imports* для відкриття доступу до тих класів бібліотеки *.NET Framework*, які пов'язані із об'єктами *ADO.NET* та провайдером *OLE DB*:

```
Imports System.Windows.Forms
Imports System.Data
Imports System.Data.OleDb
```

Далі оголосимо власні об'єкти, що знадобляться для подальшої роботи:

```
Dim myConnection As New OleDbConnection
Dim myCommand As New OleDbCommand
Dim mydataAdapter As New OleDbDataAdapter
Dim myDataSet As New DataSet
```

Для встановлення зв'язку з БД спочатку необхідно сформувати рядок підключення *StringConnection*, в якому вказується провайдер (*Provider*), шлях до БД (*Initial Catalog*), ім'я файла БД (*Data Source*), пароль (*Password*) та інші дані. Обов'язковими є лише назва провайдера і адреса БД. Наприклад, для підключення до БД з адресою *E:\Base\Tovar.mdb* формату *Access* необхідно вказати таке:

```
"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=E:\Base\Tovar.mdb"
```

Далі необхідно сформувати *SQL*-запит до БД у вигляді текстового рядка *CommandString*. Наприклад, для отримання всієї таблиці *Замовник* із БД *Tovar.mdb* *SQL*-запит матиме такий вигляд:

```
Select * From Замовник
```

Після визначення параметрів рядка з'єднання можна викликати метод *Open()* об'єкта *myConnection:*

```
myConnection.Open()
```

Для об'єкта *myCommand* необхідно вказати конкретні значення двом властивостям: властивості *Connection* присвоїти значення об'єкта *myConnection*, а властивості *CommandText* – значення текстового рядка *CommandString* із *SQL*-запитом:

```
myCommand.Connection = myConnection
myCommand.CommandText = CommandString
```

Тепер пов'яжемо об'єкт *mydataAdapter* з об'єктом *myCommand:*

```
mydataAdapter.SelectCommand = myCommand
```

і заповнимо об'єкт *myDataSet* даними із таблиці *Замовник:*

mydataAdapter.Fill(myDataSet, "Замовник")

Відобразити таблицю зручно за допомогою ЕК *DataGridView*, який має бути заздалегідь підготовленим на формі. Для цього компонента назва таблиці присвоюється властивості *DataSource:*

DataGridView1.DataSource= myDataSet.Tables("Замовник").DefaultView

Повний текст програми для підключення до БД за допомогою провайдера *OLE DB* має такий вигляд:

```
Imports System.Windows.Forms
Imports System.Data
Imports System.Data.OleDb
Dim myConnection As New OleDbConnection
Dim myCommand As New OleDbCommand
Dim mydataAdapter As New OleDbDataAdapter
Dim myDataSet As New DataSet
Dim myConnectionString As String =
            Provider=Microsoft.Jet.OLEDB.4.0; _
            Data Source=E:\Base\Tovar.mdb"
Dim CommandString As String = "Select * From Замовник"
myConnection.ConnectionString = myConnectionString
myConnection.Open()
myCommand.Connection = myConnection
myCommand.CommandText = CommandString
mydataAdapter.SelectCommand = myCommand
mydataAdapter.Fill(myDataSet, "Замовник")
Me.DataGridView1.DataSource =
            myDataSet.Tables("Замовник").DefaultView
```

Результат роботи програми показано на рис. 6.4.

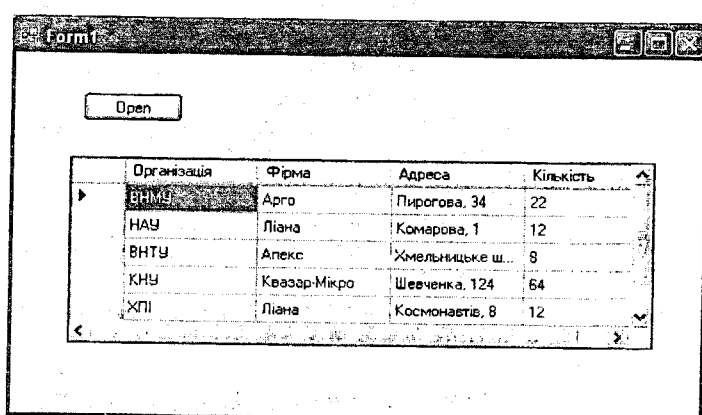

Рисунок 6.4 – Відображеня таблиці БД на формі

Порядок виконання роботи

Виконати послідовно завдання, які подані у таблицях 6.2 – 6.4.

Таблиця 6.2 – Встановлення зв'язку з БД

Завдання	Виконання
1. Створити новий проект *Visual Basic Windows Application* з іменем *MyBase*	Див. порядок виконання роботи з теми 2.
2. Викликати майстра для підключення до джерела даних	В головному меню "*Data*" вибрати пункт меню "*Add New Data Source*" (рис 6.5)
3. Вказати тип джерела даних – БД	У вікні "*Data Source Configuration Wizard*" вибрати "*DataBase*" і натиснути кнопку "*Next*" (рис 6.6)
4. Вибрати тип з'єднання з БД	У вікні "*Data Source Configuration Wizard*" вибрати базу даних і натиснути кнопку "*New Connection*" (рис 6.7)
5. Вибрати тип БД (*Access*), її розташування на диску та відповідний провайдер даних	а) у вікні "*Add Connection*" (рис 6.8) підтвердити джерело даних "*Microsoft Access Database File*" (а при необхідності його зміни натиснути кнопку "*Change...*"); б) у вікні "*Add Connection*" (рис 6.8) натиснути кнопку "*Browse*" і вибрати файл БД ("*DataBase file name*") в каталозі диску; в) при необхідності вказати нове ім'я користувача ("*User name*") і пароль доступу до БД ("*Password*"); г) натиснути кнопку "*Test Connection*" для підтвердження зв'язку з БД. Натиснути кнопку "*Ok*" для закриття вікна "*Add Connection*"
6. Підтвердити з'єднання з БД 7. Створити у робочому каталозі копію вибраної БД 8. Зберегти рядок з'єднання в конфігураційному файлі 9. Вибрати підмножину об'єктів БД, які будуть використані в подальшій роботі	У вікні "*Data Source Configuration Wizard*" натиснути кнопку "*Next*" (рис 6.9). У вікні повідомлення натиснути кнопку "*Да*" ("*Yes*") (рис 6.10) У вікні "*Data Source Configuration Wizard*" натиснути кнопку "*Next*" (рис 6.11) У вікні "*Data Source Configuration Wizard*" вибрати таблиці "*Tables*" і натиснути кнопку "*Finish*" (рис 6.12)

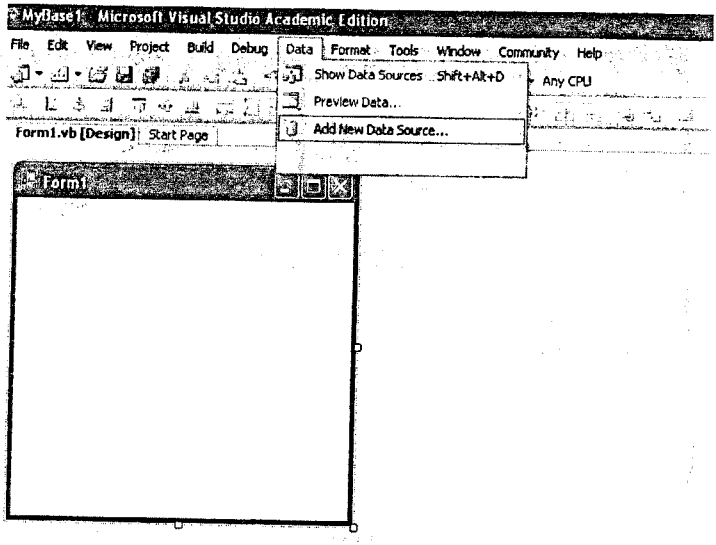

Рисунок 6.5 – Виклик майстра ADO.NET для з'єднання з БД

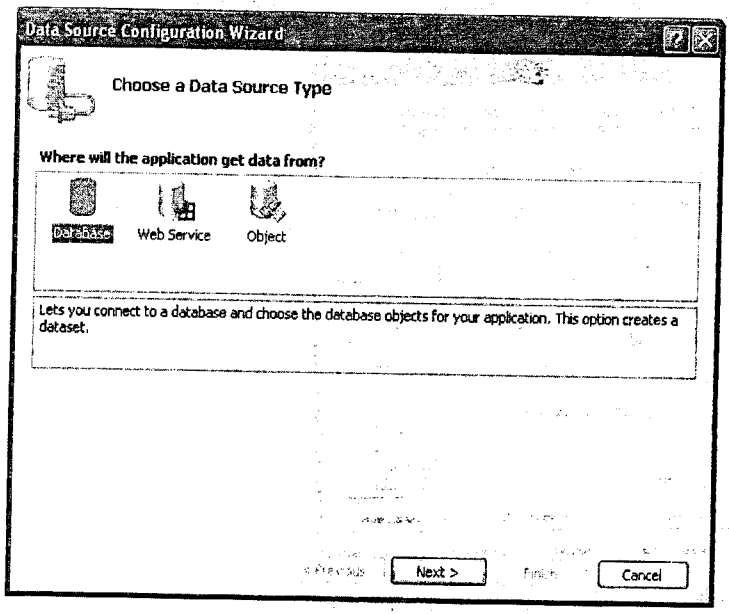

Рисунок 6.6 – Вибір типу джерела даних

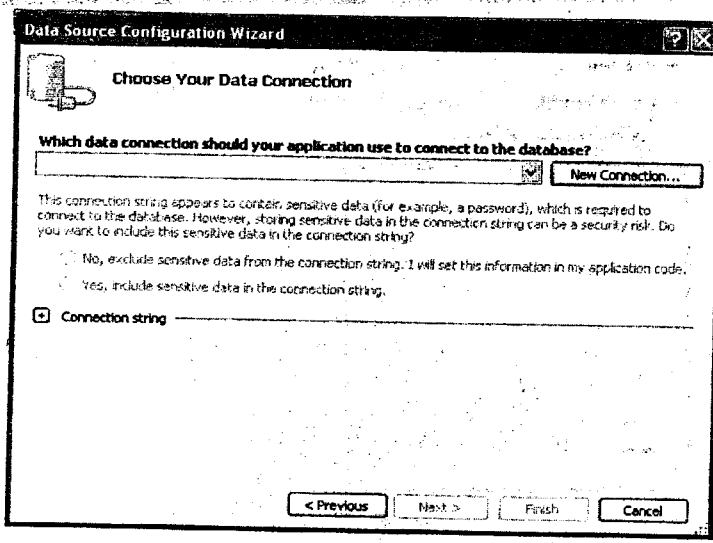

Рисунок 6.7 – Вибір типу з'єднання з БД

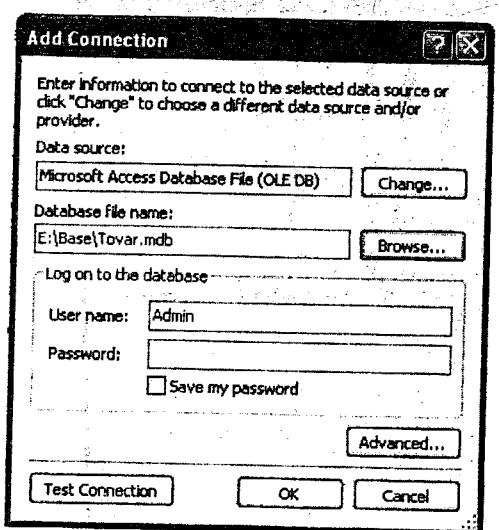

Рисунок 6.8 – Вибір розташування БД

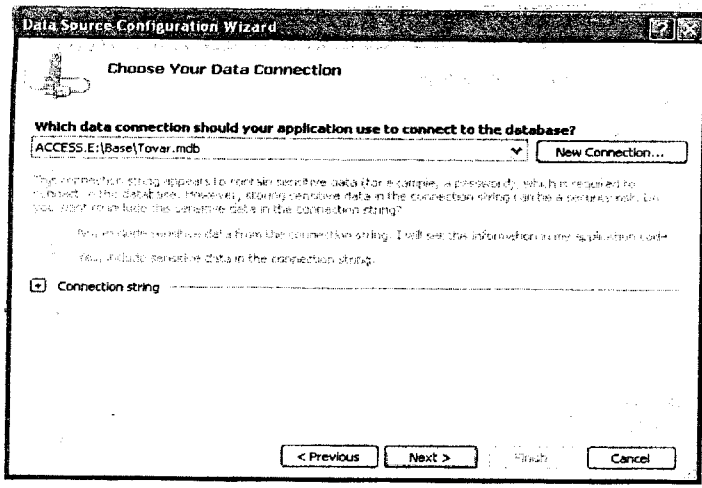

Рисунок 6.9 – Підтвердження з'єднання з БД

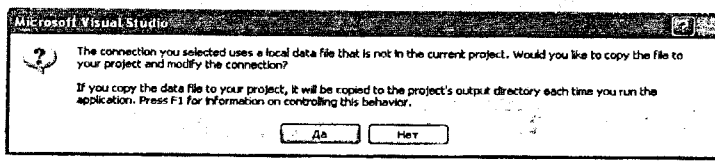

Рисунок 6.10 – Підтвердження створення копії БД

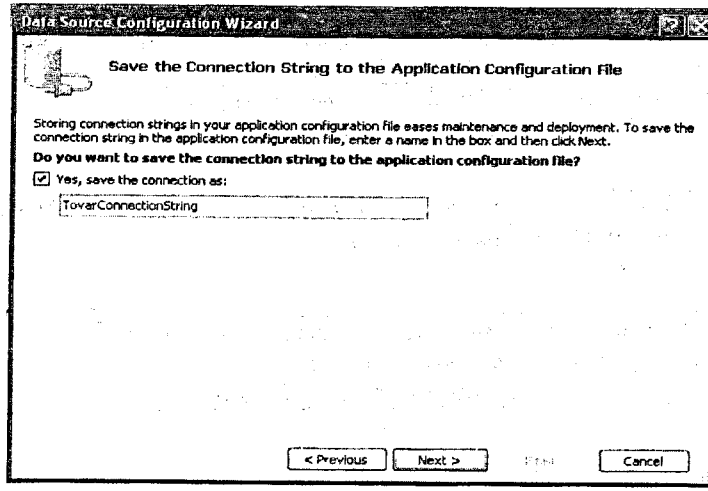

Рисунок 6.11 – Збереження рядка з'єднання з БД

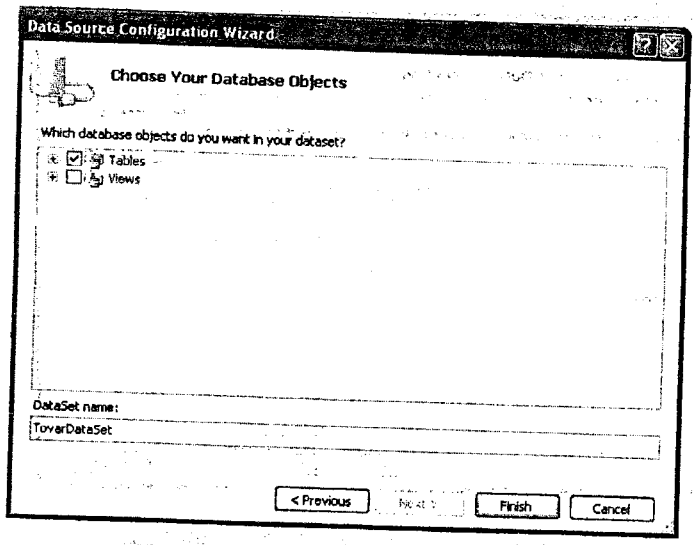

Рисунок 6.12 – Вибір об'єктів БД

Таблиця 6.3 – Візуальне подання БД на формі

Завдання	Виконання
1. Викликати майстра для візуального подання БД на формі	В головному меню "*Data*" вибрати пункт меню "*Show Data Sources*" (рис 6.13)
2. Вибрати таблицю БД для подання	Вибрати у вікні "*Data Sources*" вибрати таблицю "*Замовник*" (рис 6.14)
3. Створити на формі шаблон таблиці БД для подання	Перенести на форму із вікна "*Data Sources*" таблицю "*Замовник*" (рис 6.15)
4. Отримати на формі у компоненті *DataGridView* візуальне подання таблиці БД	Натиснути клавішу F5 (результат виконання на рис. 6.16)
5. Перевірити можливості переміщення по рядках таблиці	Використати у верхній частині форми стрілки навігатора "*ЗамовникBindingNavigator*"

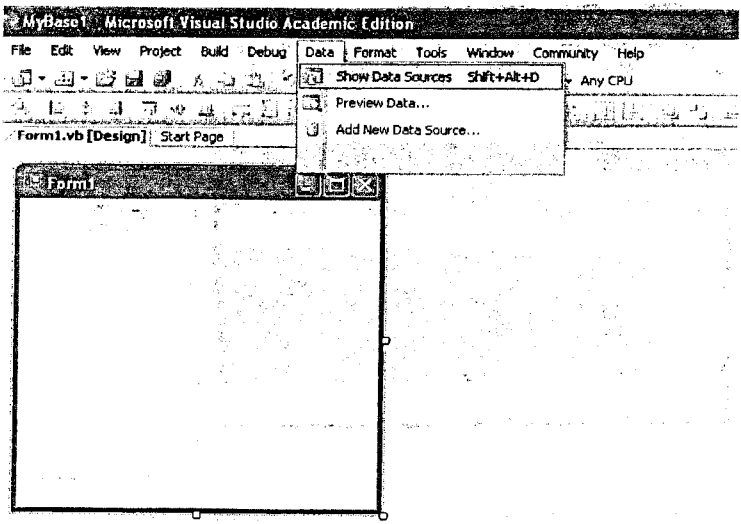

Рисунок 6.13 – Виклик майстра ADO.NET для подання БД

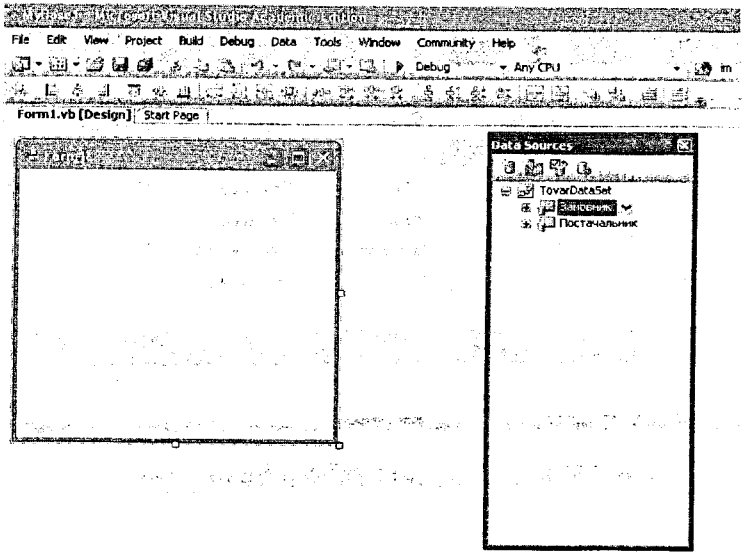

Рисунок 6.14 – Вибір таблиці БД для подання

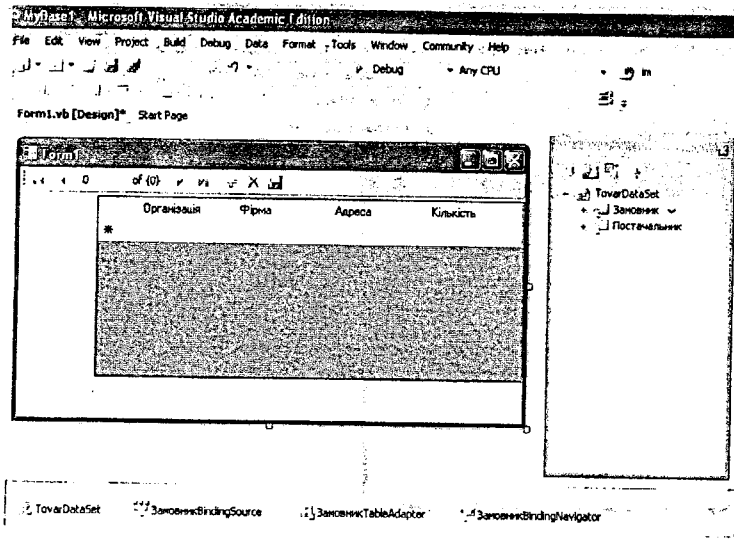

Рисунок 6.15 – Шаблон таблиці БД у компоненті *DataGridView*

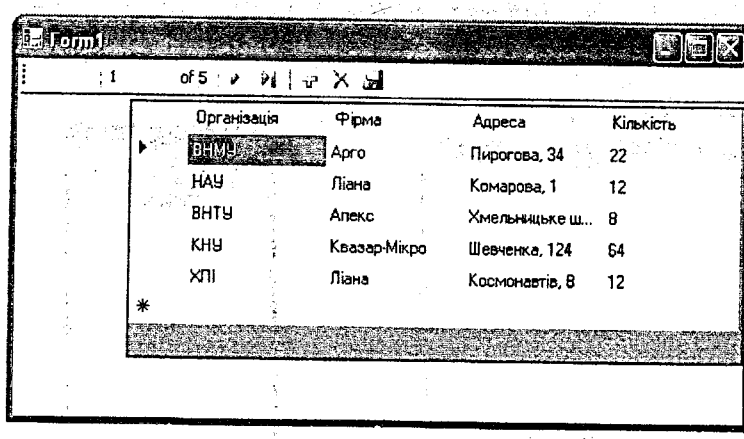

Рисунок 6.15 – Подання таблиці БД на формі

Таблиця 6.4 – Створення *SQL*-запитів за допомогою *Query Builder*

Завдання	Виконання
1. Викликати майстра для візуального створення *SQL*-запитів	В головному меню "*Data*" вибрати пункт меню "*Add Query...*" (рис 6.16).
2. Дати ім'я *SQL*-запиту	Вказати у полі "*New query name*" вікна "*Search Createria Builder*" (рис 6.17) ім'я "*Запит 1*" (за замовчуванням – ім'я *FillBy*) і натиснути кнопку "*Замовник*"
3. Створити *SQL*-запит	Всередині вікна майстра *Query Builder* записати *SQL*-запит (рис 6.18) і натиснути кнопку "*Execute Query*" .
4. Отримати результат виконання *SQL*-запиту	Результат виконання *SQL*-запиту на рис. 6.19.

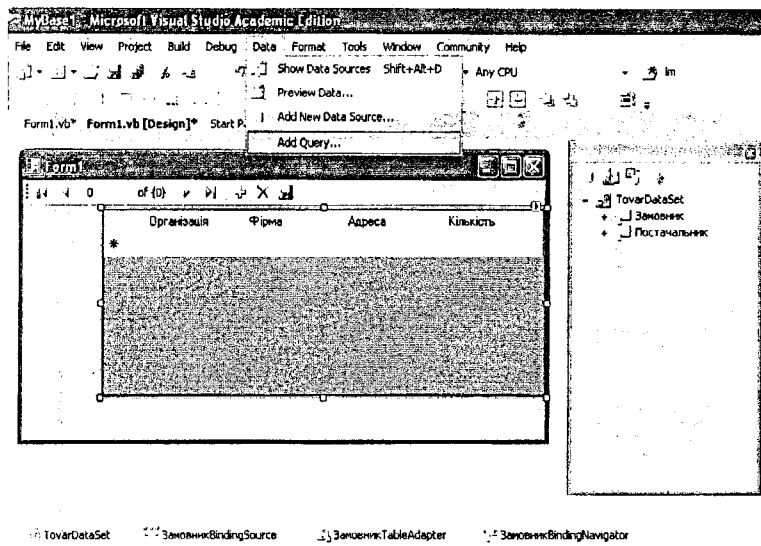

Рисунок 6.16 – Виклик майстра для створення *SQL*-запитів

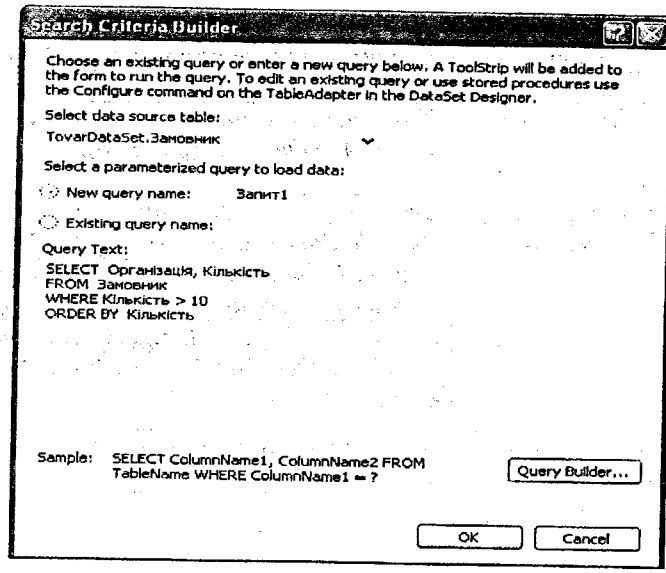

Рисунок 6.17 – Виклик майстра для створення *SQL*-запитів

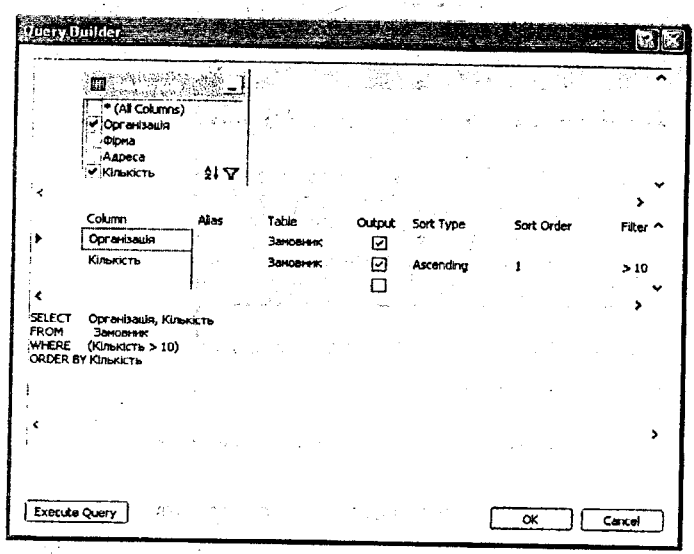

Рисунок 6.18 – Створення *SQL*-запиту

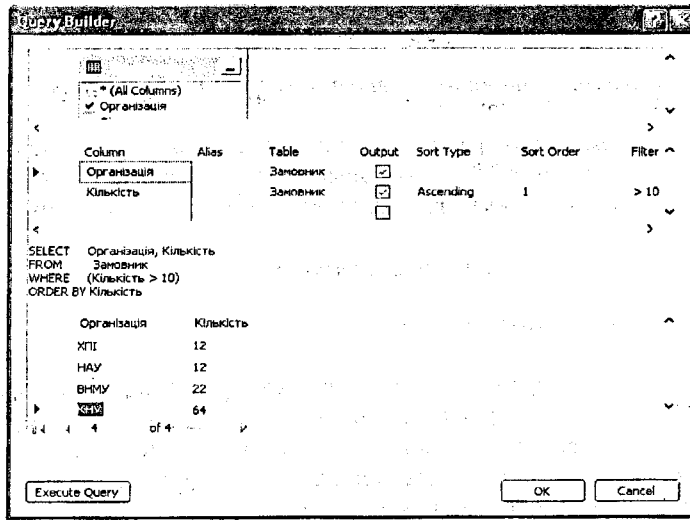

Рисунок 6.19 – Результат виконання *SQL*-запиту

Контрольні запитання

1. Розкажіть про основні типи баз даних.
2. Чим відрізняється нова технологія для роботи із БД *ADO.NET* від попередніх технологій?
3. Поясніть основні відмінності реляційних таблиць від звичайних таблиць.
4. Яка мова є внутрішнім форматом даних у *ADO.NET*?
5. Які Ви знаєте провайдери даних у *ADO.NET*?
6. Поясніть взаємодію об'єктів у об'єктній моделі *ADO.NET.*
7. В чому полягає різниця між об'єктами *DataSet* і *DataReader*?
8. Які колекції містить об'єкт *DataSet*?
9. Як відобразити дані із БД на формі? Які для цього використовуються компоненти?
10. Як створюються *SQL*-запити до БД у *Visual Basic 2005/2008* ?
11. Поясніть основні етапи програмування БД на основі провайдера *OLE DB* .
12. Поясніть основні етапи створення *SQL*-запитів за допомогою *Query Builder.*

ТЕМА № 7

Багатопотокове програмування мовою Visual Basic

Зміст теми. Створення додаткових потоків виконання, стани потоків, зміна пріоритетів потоків, механізми синхронізації потоків, пули потоків, використання делегатів для передачі даних між потоками.

Теоретичні відомості

7.1 Поняття багатопотоковості

В сучасних операційних системах можна запускати на виконання багато програм і створюється враження, начебто вони одночасно виконуються. Однак єдиний процесор не в змозі одночасно виконувати різні програми, він лише дуже швидко перемикається між ними. В термінології операційних систем це означає, що існує багато процесів (*processes*), тобто програм в стадії виконання, і в межах одного процесу виконується один або кілька потоків (*threads*). Тому такий режим роботи називають багатопотоковістю.

В момент запуску програми на виконання операційна система автоматично створює в межах процесу головний потік, а надалі можуть з'явитись додаткові потоки, створені в результаті виконання багатопотокової програми.

Написання прикладних багатопотокових програм вимагає високої кваліфікації програміста. Звичайно, необов'язково всі програми робити багатопотоковими. Тому варто звернути увагу на те, в яких випадках багатопотоковість є ефективною.

По-перше, багатопотокові програми більш оперативно реагують на дії користувача, оскільки інтерфейс користувача залишається постійно активним в той час, коли інші задачі, які вимагають інтенсивної роботи процесора, виконуються в інших потоках. Багатопотоковість також є ефективною при створенні програм, в яких користувач може додавати нові потоки для збільшення загальної продуктивності роботи.

Варто також відзначити, що багатопотоковість підтримується на рівні самої платформи *.NET*, тому принципи реалізації багатопотоковості є однаковими в *Visual Basic*, *C#* та *C++*.

Оскільки в подальшому ми на будемо згадувати про потоки введення-виведення (про них йшла мова в темі 5), тому потоки виконання далі будемо іменувати просто потоками.

7.2 Створення нового потоку виконання

Для роботи з потоками виконання у середовищі *.NET* використовується клас *System.Threading.Thread*. Для створення нового потоку у прикладній програмі необхідно спочатку створити об'єкт класу *Thread*. Конструктор цього класу приймає один параметр – об'єкт (екземпляр) делегата *ThreadStart*. Делегат вказує на той метод, що буде виконуватись як новий потік. Запускає новий потік метод *Start()* об'єкта класу *Thread*. Розглянемо приклад двопотокової програми в консольному режимі.

```
Imports System.Threading
Module Module1
 Sub ThreadFunc()
      Console.WriteLine("Hello from thread {0}!", _
      Thread.CurrentThread.GetHashCode())
End Sub
 Sub Main()
      Dim MyThread As New Thread(AddressOf ThreadFunc)
      Console.WriteLine("Main thread {0}", _
                        Thread.CurrentThread.GetHashCode())
      Console.WriteLine("Going new thread...")
      MyThread.Start()
      MyThread.Join()
      Console.WriteLine("new thread ending")
      Console.ReadLine()
End Sub
```

В цій програмі після створення головного потоку в момент запуску програми створюється об'єкт *MyThread* класу *Thread*. Метод *Start()* цього об'єкта створює додатковий потік, тобто починає виконуватися метод *ThreadFunc*, на який вказує делегат із конструктора класу *Thread*. Метод *Join()* блокує виконання головного потоку, поки не завершиться додатковий потік. На рис. 7.1 показано результат виконання програми.

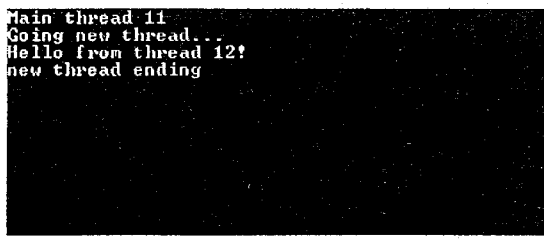

Рисунок 7.1 – Результат виконання програми 1 (варіант 1)

Якщо в програмі буде відсутній метод *Join()*, тоді може статися такий варіант виконання програми, коли спочатку виконається головний потік програми, а вже потім виконається додатковий потік, тобто рядок *"Hello from thread ..."* з'явиться останнім (рис. 7.2).

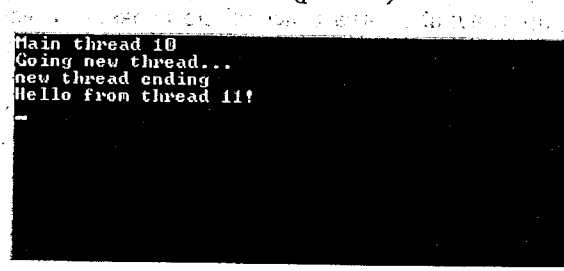

Рисунок 7.2 – Результат виконання програми 1(варіант 2)

Часто зручним є аналіз потоків по іменах, оскільки зарані невідомо, який номер присвоїть йому операційна система. Для цього можна використати властивість *Name:*

```
Dim MyThread As New Thread(AddressOf ThreadFunc)
MyThread.Name="Потік 1"
MyThread.Start()
```

7.3 Пріоритети потоків

Всі потоки при створенні отримують однаковий пріоритет (*priority*), тобто мають однакові права на використання центрального процесора. Однак часто виникає потреба в тому, щоб дати окремим потокам деяку перевагу. Наприклад, якщо одночасно з редагуванням одного документа відбувається друк іншого документа, то потоку редагування варто підвищити пріоритет для більш швидкої реакції на запити користувача.

Для задання величини пріоритету використовується властивість *Priority*, яка може мати 5 значень:

- *Lowest* (найнижчий);
- *BelowNormal* (нижче нормального);
- *Normal* (нормальний);
- *AboveNormal* (вище нормального);
- *Highest* (найвищий).

При створенні всі потоки мають однаковий пріоритет, але це значення може бути змінено. Величина пріоритету впливає на черговість виконання і завершення потоків. Розглянемо програму в *Windows Form*, в якій створюються два додаткових потоки.

```
Imports System.Drawing
Imports System.Windows.Forms
```

```
Imports System.Threading
Public Class Form1
Public G As System.Drawing.Graphics
Dim x As Integer = 30
Dim y As Integer = 90
Public font1 As New Font("Arial", 12, FontStyle.Bold)
Public thread1 = New Thread(AddressOf ThreadFunc1)
Public thread2 = New Thread(AddressOf ThreadFunc2)
Private Sub StartToolStripMenuItem_Click(...)
G = Me.CreateGraphics
thread1.Priority = ThreadPriority.Normal
thread2.Priority = ThreadPriority.Normal
G.DrawString("Привіт від головного потоку", font1, Brushes.Blue, x, y)
y = y + 20
thread1.Start()
thread2.Start()
End Sub
Sub ThreadFunc1()
G = Me.CreateGraphics
G.DrawString("Привіт від першого потоку", font1, Brushes.Blue, x, y)
y = y + 20
End Sub
Sub ThreadFunc2()
G = Me.CreateGraphics
G.DrawString("Привіт від другого потоку", font1, Brushes.Blue, x, y)
y = y + 20
End Sub
End Class
```

Оскільки обидва потоки мають однакові пріоритети, тому першим виконається той потік, який першим запускається, тобто потік 1 (рис. 7.3).

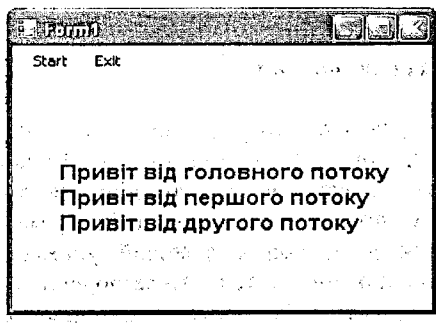

Рисунок 7.3 – Результат виконання програми 2 (варіант 1)

Однак, якщо потоку 1 надати найнижчий пріоритет, потоку 2 – найвищий:

thread1.Priority = ThreadPriority.Lowest
thread2.Priority = ThreadPriority.Highest

то першим завершиться потік 2 (рис. 7.4).

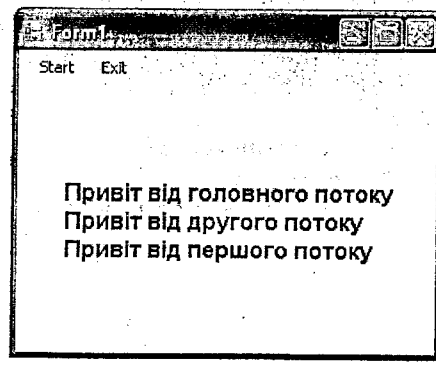

Рисунок 7.4 – Результат виконання програми 2 (варіант 2)

Така ситуація, коли пріоритети визначають черговість виконання характерна тільки для рівноцінних потоків. В загальному випадку пріоритети частіше впливають на кількість процесорного часу, що надається потоку. Тобто потік с більшою кількістю обчислень і вищим пріоритетом може завершитись пізніше потоку з малою кількістю обчислень та низьким пріоритетом.

Варто відзначити, що можна відновити початковий порядок виконання потоків, якщо першому потоку надати можливість до свого завершення заблокувати виконання інших потоків:

MyThread1.Join();

Метод *Join*() завжди гарантує завершення першим для того потоку, який його викликає.

7.4 Основні та фонові потоки

За замовчуванням потоки створюються як основні, тобто програма не буде завершена поки хоча б один із потоків буде виконуватися. *Visual Basic* підтримує також і фонові потоки, які відрізняються від основних лише тим, що вони не впливають на завершення програми. Основний потік може виконуватись безконечно, тоді як фоновий зупиняється відразу після закінчення останнього основного потоку. Перетворення основного потоку у фоновий може стати останнім шансом для завершення програми, оскільки невмираючий основний потік не дасть змоги програмі

завершитись. Необхідно пам'ятати, що основною причиною появи програм, які не можуть завершитись – це такі "забуті" основні потоки.

Фонові потоки часто створюються для приймання даних, однак це можна зробити лише тоді, коли в інших потоках присутній код, здатний обробити отримані дані.

Статус потоку перемикається з основного на фоновий за допомогою властивості *IsBackground*, як показано в такому прикладі.

```
Public thread1 = New Thread(AddressOf ThreadFunc1)
thread1. IsBackground = true
thread1.Start()
```

7.5 Стани потоків

Кожний потік користувача починає своє існування зі стану *Unstarted*. Після виклику методу *Start()* він отримує в своє розпорядження центральний процесор, тобто входить в стан *Running*. Після завершення потоку він переходить в стан *Stopped*.

Якщо потік необхідно тимчасово зупинити на задану кількість *n* мілісекунд, тоді викликається метод *Sleep(n)*, в результаті чого потік переходить в стан *WaitSleepJoin*. Після завершення часу очікування потік знову повернеться в стан *Running*.

В стан *WaitSleepJoin* потік також перейде в результаті блокування (детальніше про це в наступних розділах) або після виклику методів *Join()* чи *Monitor.Wait()*. Потік може вийти з цього стану, якщо інший потік викличе метод *Interrupt()*. Варто відмітити, що при виклику методу *Interrupt()* не вказується конкретний адресат. Це означає, що його може отримати зовсім інший потік, який може бути саме в цей момент часу заблокований, наприклад, чекаючи доступу до якогось спільного ресурсу. В результаті буде розблокований не той потік, на який розраховував програміст.

Блокований потік може бути також звільнений іншим потоком за допомогою методу *Abort()*. Однак в цьому випадку потік перейде в стан *AbortRequested*. Якщо відразу за викликом методу *Abort()* буде викликано метод *ResetAbort()*, тоді потік повернеться в активний стан, тобто в *Running*. В протилежному випадку потік аварійно завершиться і перейде в стан *Stopped*. Існує також велика відмінність між методами *Interrupt()* та *Abort()*, якщо їх викликати для незаблокованого потоку. Якщо *Interrupt()* нічого не виконує, поки потік не дійде до наступного блокування, то метод *Abort()* починає діяти негайно і здійснює аварійне завершення незаблокованого потоку.

Розглянемо приклад програми в *Windows Forms*, в якій для обчислення арифметичного виразу $z=a*b+c/d$ створюється два додаткових потоки: потік *thread1* для обчислення $z1=a*b$ і потік *thread2* для обчислення $z2=c/d$.

```
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms
Imports System.Threading
Public Class Form1
    Dim a, b, c, d, z, z1, z2 As Integer
    Dim x As Integer = 30 :  Dim y As Integer = 90
    Public G As System.Drawing.Graphics
    Public font1 As New Font("Arial", 12, FontStyle.Bold)
    Public thread1 = New Thread(AddressOf ThreadFunc1)
    Public thread2 = New Thread(AddressOf ThreadFunc2)
    Private Sub StartToolStripMenuItem_Click(...)
    G = Me.CreateGraphics
        a = 10 : b = 7 : c = 15 : d = 3
    thread1.Priority = ThreadPriority.Normal
    thread2.Priority = ThreadPriority.Highest
    Thread.CurrentThread.Sleep(3000)
    G.DrawString("*thread1 State=" + thread1.ThreadState.ToString, _
                font1, Brushes.Blue, x, y)
            y = y + 20
  thread1.Start()
 thread2.Start()
thread2.join()
  Thread.CurrentThread.Sleep(3000)
   G.DrawString("****thread1 State=" + thread1.ThreadState.ToString,

                font1, Brushes.Blue, x, y)
      y = y + 20    :   z = z1 + z2
G.DrawString("z=" + z.ToString() + " Time:" +
     DateTime.Now.ToString(), _
                font1, Brushes.Blue, x, y)
      y = y + 20
End Sub
Sub ThreadFunc1()
    G = Me.CreateGraphics
      z1 = a * b
   G.DrawString("**thread1 State=" + thread1.ThreadState.ToString, _
                font1, Brushes.Blue, x, y)
      y = y + 20
   G.DrawString("Thread1" + " Time:" + DateTime.Now.ToString(), _
     font1, Brushes.Blue, x, y)
          y = y + 20
   Thread.Sleep(1000)
End Sub
```

```
Sub ThreadFunc2()
    G = Me.CreateGraphics
      z2 = c / d
    G.DrawString("thread2 State=" + thread2.ThreadState.ToString, _
      font1, Brushes.Blue, x, y)
          y = y + 20
    G.DrawString("***thread1 State=" + thread1.ThreadState.ToString,_
      font1, Brushes.Blue, x, y)
          y = y + 20
    G.DrawString("Thread2" + " Time:" + DateTime.Now.ToString(), _
      font1, Brushes.Blue, x, y)
          y = y + 20
    Thread.Sleep(1000)
End Sub
End Class
```

Проаналізуємо детально послідовність станів потоку *thread1*. Відразу після створення потік переходить в стан *Unstarted*. Далі почергово запускаються потоки *thread1* та *thread2* і вони переходять в стан *Running*. Хоча потік *thread2* запускається пізніше, але він має найвищий пріоритет, і, що саме головне, викликає метод *Join()*. В результаті робота потоку *thread1* блокується і він переходить в стан *WaitSleepJoin*. Після завершення потоку *thread2* потік *thread1* знову повертається в стан *Running*, а потім завершується, переходячи в стан *Stopped*. На рис. 7.5 показано результат виконання програми.

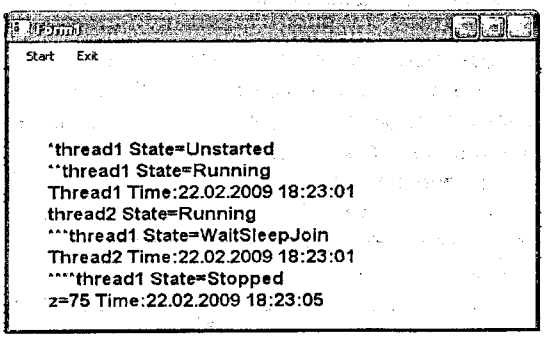

Рисунок 7.5– Результат виконання програми 3

7.6 Синхронізація паралельних потоків

Мультизадачна операційна система дозволяє керувати багатьма одночасно процесами і потоками одночасно причому як системними, так і прикладними. Природно, виникає запитання про те, яким чином в

комп'ютері з одним центральним процесором (можна ще уточнити – одноядерним) може одночасно виконуватись багато програм. Адже в будь-який момент часу процесор може працювати лише з одним процесом (потоком).

Суть мультипрограмного режиму роботи полягає в тому, що всі потоки процесів отримують в своє розпорядження центральний процесор тільки в маленькі проміжки часу (кванти часу), а в інший час знаходяться в різних типах станів готовності і очікування (*WaitSleepJoin, Suspended* та ін.). Процесор по черзі обслуговує всі потоки, що і створює ілюзію одночасної роботи.

Будемо називати потоки паралельними, якщо вони знаходяться в активному (*Running*) стані, чи в одному із станів готовності або очікування. Паралельні потоки можуть бути незалежними або взаємодіючими.

Незалежні потоки не впливають на результати роботи один одного, оскільки в них не перетинаються файли початкових даних та області оперативної пам'яті, де зберігаються проміжні та кінцеві результати роботи. Вони можуть тільки бути причиною затримки один одного, тому що використовують один процесор.

Взаємодіючі потоки спільно використовують також і деякі загальні об'єкти (файли, області пам'яті), і результати виконання одного потоку можуть залежати від ходу виконання іншого потоку. При виконанні взаємодіючих паралельних потоків можуть виникати гонки або взаємне блокування. Гонки з'являються, коли два чи більше потоків потребують доступу до одного фрагмента пам'яті, і тільки один із них може виконати цю операцію безпечно. Прикладом взаємного блокування може служити ситуація, коли з двох потоків кожний чекає звільнення ресурсу іншим потоком. Обидва чекають один одного, і жодний із них не може продовжити своє виконання, поки не буде виконано те, чого він очікує, в результаті потоки залишаються в стані довічного очікування.

Для вирішення таких проблем і служать різні механізми синхронізації, які дозволяють взаємодіючим потокам коректно працювати із спільними ресурсами та уникати можливості виникнення тупикових ситуацій: блокування пам'яті, системи переривань, семафорів, моніторів, асинхронних делегатів та ін.

Відразу варто звернути увагу на те, що при розгляді задач синхронізації виключається центральний процесор, оскільки для нього ця задача вирішується диспетчером задач операційної системи за заранні визначеними алгоритмами і користувач не може їх змінювати. В подальшому будуть розглядатись лише ті спільно використовувані ресурси, до яких заборонено одночасний доступ кількох потоків. Такі ресурси, до яких в кожний момент часу може мати доступ тільки один потік, будемо називати критичними.

Якщо кілька потоків хочуть користуватись критичним ресурсом в режимі розподілу часу, їм необхідно синхронізувати свої дії таким чином,

щоб ресурс завжди знаходився в розпорядженні не більше ніж в одного потоку. Якщо один потік користується в даний момент часу критичним ресурсом, то всі інші потоки повинні в цей час очікувати його звільнення. Можливі також і такі критичні ресурси, які можуть знаходитися в одночасному розпорядженні кількох потоків, але кількість цих потоків також наперед обмежена.

7.7 Використання механізму блокувань в задачах синхронізації

Блокування пам'яті, тобто тимчасове призупинення, належить до найпростіших способів синхронізації. Суть його полягає в забороні одночасного виконання двох і більше потоків, які звертаються до спільної комірки пам'яті. Як правило, будь-яке поле (змінна), доступне кільком потокам, повинно читатись і записуватись з блокуванням пам'яті. Блокування само по собі дуже швидке і потребує лише кількох десятків наносекунд, якщо власне блокування не відбувається. Якщо ж блокування необхідне, тоді подальше перемикання задач займає вже мікросекунди або мілісекунди на перепланування потоків.

Розглянемо приклад програми, в якій створюються два додаткових потоки. В наступній програмі показано використання об'єкта класу *Lock*.

```
Imports System.Threading
Public Class Form1
Public m As Integer = 0
Public d As Boolean
Private locker As Object = New Object()

Public thread1 = New Thread(AddressOf ThreadFunc1)
Public thread2 = New Thread(AddressOf ThreadFunc2)

Private Sub StartToolStripMenuItem_Click(0
        thread1.Name = "First thread"
        thread2.Name = "Second thread"
        thread1.Start()
        thread2.Start()
 End Sub
 Sub ThreadFunc1()
        Thread.Sleep(1000)
        PrintFunc1()
End Sub
 Sub ThreadFunc2()
        Thread.Sleep(1000)
        PrintFunc1()
End Sub
```

```
Sub PrintFunc1()
    SyncLock (locker)
        Do Until (d)
        m = m + 1
        d = True
        Loop
    End SyncLock
    MsgBox(Thread.CurrentThread.Name & " m=" & m.ToString)
End Sub
End Class
```

Кожний з потоків звертається до спільної функції *PrintFunc1()*, в якій здійснюється інкремент змінної *m*. Обмеження полягає в тому, що операція інкремента має бути виконана тільки один раз, незалежно від кількості звертань до функції *PrintFunc1()*. Таким чином, той потік, який першим викличе цю функцію, і виконає операцію інкремента, а виклик від іншого потоку має бути заблоковано. Для виконання цієї задачі створюється об'єкт *Lock*. Об'єкти класу *Lock* необов'язково створювати явно, для них в мові C# вже передбачено ключове слово *lock*. Коли два потоки одночасно борються за першочерговий доступ (в нашому випадку за право володіння об'єктом *locker*), один потік переходить в стан *WaitSleepJoin* (блокується), поки блокування не звільняється. В даному випадку це гарантує те, що тільки один потік зможе виконати критичну область коду (інкремент змінної). Код, який захищений таким чином від невизначеності багатопотокового програмування, називається потокобезпечним.

Безпечний інкремент змінної в потоці є настільки розповсюдженою задачею, що існує спеціальний клас для виконання цієї задачі – *InterLocked*. В таблиці 7.1 перераховані його загальнодоступні статичні методи.

Таблиця 7.1. – Методи класу *InterLocked*

Метод	Опис
CompareExchange()	Порівняння двох значень
Increment()	Зменшення змінної на одиницю (інкремент)
Exchange()	Обмін значеннями двох змінних
Decrement()	Збільшення змінної на одиницю (декремент)

На відміну від класу *Lock* клас *InterLocked* не забороняє іншим потокам змінювати спільну змінну, він лише забезпечує коректну почергову операцію (інкремента чи декремента) від різних потоків.

Варто зазначити, що при неправильному використанні у блокування можуть бути і негативні наслідки – взаємоблокування, гонки блокувань, зменшення загального ефекту паралелізму. Остання ситуація може мати

місце, коли дуже багато програмного коду міститься в конструкції *lock*, змушуючи інші потоки простоювати, поки один потік виконується.

7.8 Використання класу *Monitor* в задачах синхронізації

Розглянуте раніше ключове слово *lock* – це насправді скорочений спосіб використання класу *Monitor*. Однак цей клас не обмежується тільки задачею блокування доступу до спільного ресурсу, його можливості значно більші. В табл. 7.2 перераховані його загальнодоступні статичні методи.

Таблиця 7.2. – Методи класу *Monitor*

Метод	Опис
Enter()	Початок блокування об'єкта. Якщо об'єкт вже заблоковано іншим потоком, то виконання поточного потоку призупиняється до звільнення об'єкта
Exit()	Звільнення блокування об'єкта
Pulse()	Інформування наступного потоку, який призупинено, про можливість продовження роботи
PulseAll()	Інформування всіх призупинених потоків про можливість продовження роботи
TryEnter()	Намагання заблокувати переданий об'єкт
Wait()	Зняття всіх блокувань і призупинення поточного потоку до тих пір, поки інший потік не викличе метод *Pulse()*

При виклику методу *Enter()* робота з об'єктом обмежується лише одним потоком, інші призупиняються і очікують. На відміну від простого блокування за допомогою об'єктів класу *Lock* в даному випадку жоден призупинений процес не ігнорується. Потік, що утримує блокування, може викликати метод *Pulse()* і вказати в ньому конкретний потік, який може першим переміщатися в чергу готових до виконання. За допомогою методу *PulseAll()* можна всі призупинені потоки перевести в чергу готових до виконання. Як тільки блокування знімається (після виклику методу *Enter()* або *Wait()*), перший потік із черги готових отримує дозвіл працювати з об'єктом. Таким чином всі потоки, які зробили заявку на роботу із визначеним об'єктом, отримають таку можливість, але не одночасно.

В наступній програмі показано спосіб використання класу *Monitor* .

```
Imports System.Threading
Public Class Form1
Public m As Integer = 0
Public d As Boolean = True
Private locker As Object = New Object()
```

```
Public thread1 = New Thread(AddressOf ThreadFunc1)
Public thread2 = New Thread(AddressOf ThreadFunc2)

Private Sub StartToolStripMenuItem_Click()
        thread1.Name = "First thread"
        thread2.Name = "Second thread"
        thread1.Start()
        thread2.Start()
End Sub
Sub ThreadFunc1()
      PrintFunc1()
End Sub
Sub ThreadFunc2()
      PrintFunc1()
End Sub

Sub  PrintFunc1()
       Monitor.Enter(locker)
            Try
                  If (d) Then m = m + 1
      MsgBox(Thread.CurrentThread.Name & " m=" &  m.ToString)
                  Thread.Sleep(1000)
                  d = True
            Finally
                  Monitor.Exit(locker)
            End Try
End Sub
End Class
```

Ця програма подібна до раніше розглянутої: кожний з двох потоків звертається до спільної функції *PrintFunc1()*, в якій здійснюється інкремент змінної *m*. Потоки по черзі отримують доступ до операції інкремента, в результаті за значенням змінної *m* можна визначити загальну кількість потоків. Нагадаємо, що за допомогою об'єкта класу *Lock* можна було визначити лише факт звертання одного із потоків.

7.9 Асинхронні делегати

Раніше вже розглядався один із варіантів передавання даних в потік. Часто виникає інша потреба – повернення результатів роботи після завершення потоку. Зручним способом виконання цієї задачі є використання асинхронних делегатів (*delegates*), які дозволяють передавати в обох напрямках будь-яку кількість параметрів з одночасним контролем їх типу.

Делегат – це такий клас, об'єкт якого посилається на метод. Делегат може посилатись або на один метод (якщо він є нащадком класу *System.Delegate*), або на багато методів (якщо він є нащадком класу *System.MulticastDelegate*). В подальшому будуть використовуватись делегати-нащадки останнього класу, в якому передбачені методи: *Invoke()*, *BeginInvoke()* та *EndInvoke()*. Метод *Invoke()* виконує синхронний виклик делегата, а два останні методи – асинхронні виклики.

Розглянемо програму, в якій для обчислення заданого арифметичного виразу використовується функція *Summa()*. В цю функцію необхідно передати два початкових параметри і отримати результат обчислень.

```
class Program
  {
    static float a, b;
    private delegate float MyDelegate(float a, float b);
    //Метод для вычисления заданного выражения
    private static float Summa(float a,float b)
    {
      float y;
      Console.WriteLine("Calculation of expression in thread {0}!",
            Thread.CurrentThread.GetHashCode());
      y=a+b;
     Thread.Sleep(3000);
     return y;
    }
    static void Main(string[] args)
    {
       a = 10; b = 7;
       MyDelegate del = new MyDelegate(Summa);
       IAsyncResult endOp1 = del.BeginInvoke(a,b, null, null);
       Thread.Sleep(2000);
       //Завершить асинхронный вызов
       Console.WriteLine("Waiting...");
       float result1 = del.EndInvoke(endOp1);
       Console.WriteLine("Sum : {0}",result1);
       Console.ReadLine();
    }
  }
```

Спочатку створюється окремий потік, який здійснює виклик функції *Summa()* для обчислення заданого виразу. Метод *BeginInvoke()* починає асинхронний виклик делегата *del*. Метод *EndInvoke()* завершує асинхронний виклик делегата *del*, повертаючи його значення. Якщо на момент виклику методу *EndInvoke()* делегат ще не завершив свою роботу,

тоді викликаючий потік буде заблоковано до закінчення роботи потоку, який виконує асинхронний виклик.

7.10 Пули потоків

Якщо в програмі багато потоків, тоді досить важко організувати їх оптимальну взаємодію, багато часу буде витрачатись на очікування потоків в різних чергах. Для таких випадків в C# передбачено спеціальний механізм – пул (*pool*) потоків, для організації якого передбачено і спеціальний клас *ThreadPool*.

Пул – це така сукупність потоків, для яких передбачено централізоване керування. Немає потреби створювати кожний окремий потік за тією процедурою, що була описана в попередніх розділах, пул створюється весь відразу. Більше того, навіть не потрібно попередньо оголошувати про його створення, пул створюється автоматично при першому звертанні до нього. Для того, щоб поставити один потік в пул, просто викликається метод *ThreadPool.QueueUserWorkItem*, якому передається відповідний делегат.

Розглянемо текст програми, в якій створюється пул із 10 потоків, які по черзі створюються і завершуються.

```
class Program
{
  static object workerLocker = new object();
  static int number = 10;
  public static void Main()
  {
    for (int i = 0; i < number; i++)
      ThreadPool.QueueUserWorkItem(Function, i);
    Console.WriteLine("We expect the end of work of the threads...");
    lock (workerLocker)
      while (number > 0)
        Monitor.Wait(workerLocker);
    Console.WriteLine("Ready!");
    Console.ReadLine();
  }
  public static void Function(object instance)
  {
    Console.WriteLine("Begin:  " + instance);
    Thread.Sleep(1000);
    Console.WriteLine("End: " + instance);
    lock (workerLocker)
    {   number--;      Monitor.Pulse(workerLocker);   }
  }
}
```

Пул потоків розрахований на максимально ефективну роботу окремих потоків. Пул підтримує паралельну роботу такої кількості потоків, яка максимально можлива в даній системі (за замовчуванням 25). Якщо кількість потоків буде перевищувати цю цифру, тоді всі додаткові потоки автоматично організовуються в чергу очікування. Всі потоки пулу є фоновими, і вони знищуються автоматично після завершення основного потоку програми.

Порядок виконання роботи

1. Створити у *Visual Studio* новий проект типу "*Console Application*".
2. Згідно з індивідуальним завданням написати багатопотокову програму і перевірити її роботу.
3. Створити у *Visual Studio* новий проект типу "*Windows Forms Application*".
4. Згідно з індивідуальним завданням написати багатопотокову програму і перевірити її роботу.
5. У кожній потоковій функції організувати виведення на екран дисплея інформацію про стан всіх інших потоків.
6. Провести дослідження ходу виконання програми змінюючи пріоритети додаткових потоків.
7. Додати до попередньої програми функцію з використання об'єкта класу *Lock* і провести дослідження такого механізму синхронізації.
8. Додати до попередньої програми функцію з використання класу *Monitor* і провести дослідження такого механізму синхронізації.
9. Згідно з індивідуальним завданням написати багатопотокову програму для розпаралелювання циклів з використанням пулу потоків.

Контрольні запитання

1. В чому переваги і недоліки багатопотокових програм? В яких випадках варто створювати додаткові потоки у програмах?
2. Як створити додаткові потоки у програмах для проектів типу "*Console Application*" та типу "*Windows Forms Application*"?
3. Які бувають пріоритети потоків?
4. В яких станах можуть знаходитись потоки?
5. Якими способами можна передати дані в потоки та отримати від них результати роботи?
6. В чому полягає різниця між основними і фоновими потоками?
7. Для вирішення яких проблем використовуються механізми синхронізації потоків?
8. В чому полягає різниця між використанням об'єкта класу *Lock* та використанням класу *Monitor* ?

Український варіант	Англійський варіант
Багатодокументний інтерфейс	*Multiple-Document Interface (MDI)*
База даних	*Database*
Бібліотека елементів керування	*Control Library*
Бібліотека класів	*Class Library*
Вікно введення	*InputBox*
Вікно повідомлення	*MessageBox*
Властивість	*Property*
Делегат	*Delegate*
Загальнодоступний	*Public*
Закритий	*Private*
Запит	*Query*
Збірка	*Building*
Інтегроване середовище розробки	*Integrated Development Environment (IDE)*
Каталог	*Directory*
Консольна програма	*Console Application*
Компонент "Головне меню"	*MenuStrip*
Компонент "Кнопка"	*Button*
Компонент "Комбінований список"	*ComboBox*
Компонент "Мітка"	*Label*
Компонент "Перемикач"	*RadioButton*
Компонент "Прапорець"	*CheckBox*
Компонент "Рядок стану"	*StatusStrip*
Компонент "Список"	*ListBox*
Компонент "Список з прапорцями"	*CheckedListBox*
Компонент "Текстове поле"	*TextBox*
Компонент "Текстове поле з маскою"	*MaskedTextBox*
Компонент "Числове поле"	*NumericUpDown*
Конструктор форм	*Designer*
Криволінійний градієнтний пензель	*PathGradientBrush*
Метод "Зафарбувати багатокутник"	*FillPolygon*
Метод "Зафарбувати еліпс"	*FillEllipse*
Метод "Зафарбувати прямокутник"	*FillRectangle*
Метод "Нарисувати багатокутник"	*DrawPolygon*
Метод "Нарисувати еліпс"	*DrawEllipse*
Метод "Нарисувати криву Без'є"	*DrawBezier*
Метод "Нарисувати лінію"	*DrawLine*
Метод "Нарисувати прямокутник"	*DrawRectangle*
Оглядач розв'язків	*Declaration*

Оголошення	*Solution Explorer*
Однодокументний інтерфейс	*Single-Document Interface (SDI)*
Панель елементів керування	*Toolbox*
Панель інструментів	*Toolbars*
Перо	*Pen*
Підпрограма	*Subroution*
Прикладний програмний інтерфейс	*Application Program Interface*
Прикріплення	*Dock*
Покращений графічний інтерфейс	*Graphics Device Interface+ (GDI+)*
Початкова сторінка	*Start Page*
Пріоритет	*Priority*
Пріоритет "Вище нормального"	*AboveNormal priority*
Пріоритет "Найвищий"	*Highest priority*
Пріоритет "Найнижчий"	*Lowest priority*
Пріоритет "Нижче нормального"	*BelowNormal priority*
Пріоритет "Нормальний"	*Normal priority*
Потік введення-виведення	*Stream*
Потік виконання	*Thread*
Провайдер	*Provider*
Провідник	*Explorer*
Проект	*Project*
Процес	*Process*
Прямолінійний градієнтний пензель	*LinearGradientBrush*
Пул	*Pool*
Редактор коду	*Code Editor*
Режим роботи	*Mode*
Розв'язок	*Solution*
Розмір	*Dim (dimension)*
Статичний	*Static*
Суцільний пензель	*SolidBrush*
Текстурний пензель	*TextureBrush*
Тип доступу	*Access*
Тип дозволу	*Share*
Файл	*File*
Функція	*Function*
Форми	*Forms*
Штриховий пензель	*HatchBrush*
Штриховка діагональна	*BackwardDiagonal*
Штриховка хрест-навхрест	*Cross*
Штриховка точками	*Divot*
Штрихпунктирна лінія	*DashDot*

ЛІТЕРАТУРА

1. Дейтел П., Дейтел Х., Эйр Г. Просто о Visual Basic 2008. – СПб.: БХВ-Петербург, 2009. – 1232 с.
2. Шевякова Д. А., Степанов А. М., Карпов Р. Г. Самоучитель Visual Basic 2005. – СПб.: БХВ-Петербург, 2007. – 576 с.
3. Шевякова Д. А., Степанов А. М., Дукин А. Н. Самоучитель Visual Basic 2008. – СПб.: БХВ-Петербург, 2008. – 592 с.
4. Богданов М. Р. Visual Basic 2005 на примерах. – СПб.: БХВ-Петербург, 2007. – 592 с.
5. Хальворсон М. Microsoft Visual Basic 2005. – М.: ЭКОМ Паблишерз, 2007. – 640 с.
6. Долженков В., Мозговой М. Visual Basic.NET. – СПб.: Питер, 2003. – 464 с.
7. Малачівський П. С. Програмування в середовищі Visual Basic: Навчальний посібник. – Львів: "Бескид Біт", 2004. – 260 с.

Навчальне видання

Семеренко Василь Петрович

ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

Навчальний посібник

Редактор О. Скалоцька

Оригінал-макет підготовлено В. Семеренком

Підписано до друку 12.06.2010 р.
Формат 29,7х42 ¼. Папір офсетний.
Гарнітура Times New Roman.
Друк різографічний. Ум. друк. арк. 7.1.
Наклад 85 прим. Зам. № 2010-105

Вінницький національний технічний університет,
науково-методичний відділ ВНТУ.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, к. 2201.
Тел. (0432) 59-87-36.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.

Віддруковано у Вінницькому національному технічному університеті
в комп'ютерному інформаційно-видавничому центрі.
21021, м. Вінниця, Хмельницьке шосе, 95,
ВНТУ, ГНК, к. 114.
Тел. (0432) 59-81-59.
Свідоцтво суб'єкта видавничої справи
серія ДК № 3516 від 01.07.2009 р.