

Міністерство освіти і науки України
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання самостійної роботи студентів
з дисципліни

“Основи програмної інженерії”

для студентів

спеціальності 121 Інженерія програмного забезпечення
(всіх форм навчання)

Методичні вказівки до виконання самостійної роботи студентів з дисципліни “Основи програмної інженерії” для студентів спеціальності 121 Інженерія програмного забезпечення (всіх форм навчання) / А. О. Олійник, Є. М. Федорченко. – Запоріжжя: ЗНТУ, 2016. – 133 с.

Автори: Андрій Олександрович Олійник, к.т.н., доцент
Євген Миколайович Федорченко, ст. викладач

Рецензент: С. О. Субботін, д.т.н., професор

Відповідальний
за випуск: С. О. Субботін, д.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол № 1
від “16” серпня 2016 р.

ЗМІСТ

Вступ	6
1 Самостійна робота № 1 Розширення можливостей MS Office за допомогою Visual Basic for Application.....	7
1.1 Мета роботи	7
1.2 Основні теоретичні відомості.....	7
1.2.1 Форми користувача у VBA.....	7
1.2.2 Створення форми користувача.....	10
1.2.3 Тестування й відлагодження програм	11
1.2.4 Діалогові вікна	13
1.2.5 Алгоритмічні конструкції мови VBA	16
1.2.6 VBA у застосування до MS Word.....	19
1.3 Завдання до роботи.....	20
1.4 Зміст звіту.....	28
1.5 Контрольні запитання	28
2 Самостійна робота № 2 Розробка текстового редактору у C# ...	29
2.1 Мета роботи	29
2.2 Основні теоретичні відомості.....	29
2.2.1 Створення головного меню	30
2.2.2 Створення MDI-додатків	33
2.2.3 Перерахування MdiLayout	36
2.2.4 Вирізання, копіювання і вставка текстових фрагментів	38
2.2.5 Контекстне меню	39
2.2.6 Діалогові вікна	42
2.2.7 Збереження файлу при закритті форми	48
2.2.8 OpenFileDialog і SaveFileDialog для SDI-додатків	49
2.2.9 FontDialog	51
2.2.10 ColorDialog	52
2.2.11 StatusBar.....	53
2.2.12 Закривання форми	56
2.2.13 CheckBox	57
2.2.14 Властивість TabIndex елементів управління.....	61
2.3 Завдання до роботи.....	64
2.4 Зміст звіту.....	64

2.5 Контрольні запитання	65
--------------------------------	----

3 Самостійна робота №3 Робота з меню та обробка винятків у

С#	66
3.1 Мета роботи	66
3.2 Основні теоретичні відомості.....	66
3.2.1 Меню та його пункти	66
3.2.1.1 Контекстне меню	66
3.2.1.2 Відмічені пункти меню	67
3.2.1.3 Поділ елементів меню	67
3.2.1.4 Клавіші доступу.....	67
3.2.1.5 Гарячі клавіші	68
3.2.1.6 Активація та деактивація команд меню	68
3.2.1.7 Вставка зображень для пунктів меню.....	68
3.2.1.8 Створення меню програмно	69
3.2.2 Обробка та генерація винятків	71
3.2.2.1 Принципи обробки винятків.....	72
3.2.2.2 Генерація винятків.....	76
3.2.2.3 Повторна генерація винятків	77
3.2.2.4 Нестандартний виняток	78
3.2.2.5 Генерація власних винятків	79
3.2.2.6 Корисні поради генерації винятків	80
3.3 Завдання.....	83
3.4 Зміст звіту.....	83
3.5 Контрольні питання.....	84

4 Самостійна робота № 4 Робота з друком та створення

елементів керування у С#	85
4.1 Мета роботи	85
4.2 Основні теоретичні відомості.....	85
4.2.1 Друк	85
4.2.1.1 Компонент PrintDocument.....	85
4.2.1.2 Вивід на друк	88
4.2.1.3 Застосування елемента керування PrintPreviewControl	92
4.2.1.4 Налаштування параметрів друку.....	94
4.2.1.5 Приклад друку вмісту різних компонентів	97
4.2.2 Створення елементів керування за допомогою. NET Framework.....	105
4.2.2.1 Застосування GDI + ...	105

4.2.2.2 Створення елементів керування.....	116
4.2.2.3 Розв'язання типових задач, пов'язаних з розробкою елементів керування.....	125
4.3 Завдання до роботи.....	130
4.4 Зміст звіту.....	131
4.5 Контрольні питання.....	131
Література.....	133

ВСТУП

Дане видання призначене для вивчення та практичного освоєння студентами денної форми навчання основ програмної інженерії.

Для виконання розрахунково-графічного завдання студенти повинні ознайомитися з рекомендованою літературою, а також з електронними джерелами з середовища Internet.

Для одержання заліку з розрахунково-графічного завдання студент здає викладачу цілком оформлений звіт, а також демонструє на екрані комп'ютера результати виконання роботи.

Звіт має містити:

- титульний аркуш (на ньому вказують назву міністерства, назву університету, назву кафедри, номер, вид роботи, виконавця та особу, що приймає звіт, рік);
- завдання до роботи;
- результати виконання роботи, зокрема текст розробленого програмного забезпечення, а також змістовний аналіз отриманих в ході розв'язання практичного завдання результатів, висновки.

Звіт виконують на білому папері формату А4 (210 x 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок або вміщують у канцелярський файл. Звіт повинен бути оформлений відповідно до вимог діючих стандартів.

Під час співбесіди при захисті розрахунково-графічного завдання студент повинний виявити знання про мету роботи, по теоретичному матеріалу, про методи виконання кожного етапу роботи, по змісту основних розділів оформленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати.

Загальна оцінка з розрахунково-графічного завдання включає оцінку змісту та рівня виконання пояснювальної записки, оцінку доповіді студента під час захисту, своєчасність завершення виконання розрахунково-графічного завдання, оцінку якості та функціональності розробленої програмної системи.

1 САМОСТІЙНА РОБОТА № 1

РОЗШИРЕННЯ МОЖЛИВОСТЕЙ MS OFFICE ЗА ДОПОМОГОЮ VISUAL BASIC FOR APPLICATION

1.1 Мета роботи

1.1.1 Вивчити основні можливості Visual Basic for Application щодо створення екранних форм.

1.1.2 Навчитися створювати діалогові вікна за допомогою Visual Basic for Application.

1.2 Основні теоретичні відомості

Обмеженість макросів, створених за допомогою стандартних засобів офісних програм MS Office обумовлює необхідність створення програм на VBA.

1.2.1 Форми користувача у VBA

Visual Basic for Application дозволяє створювати графічний інтерфейс розроблювальних додатків на основі використання елементів керування (кнопки, текстові поля і т.ін). Елементи керування (рис. 1.1) є об'єктами. Тому, як будь-які об'єкти, вони мають властивості, методи й можуть відгукуватися на події. Елементи керування можна вставляти як у робочі аркуші, так і в екранні форми.

Екранні форми (їх також називають користувацькими формами, від англ. *Userform*) — це вікна (звичайного виду або діалогові), що є частиною інтерфейсу додатка.

Робота з користувацькою формою складається з декількох етапів:

- відкрити або створити файл для роботи;
- перейти в редактор Visual Basic;
- створити користувацьку форму;
- застосувати до створеної форми властивості;
- створити у формі елементи керування;
- написати процедури обробки подій.

Scrollbar (смуга прокручування)  – це не зовсім те, чому ви звично користуєтеся для прокручування, наприклад, робочого аркуша. У цьому випадку смуги прокручування застосовуються як механізми вибору. У графічних програмах ви, напевно, бачили, як за допомогою смуг прокручування можна вибрати колір або зробити зображення більш світлим або більш темним.

Spinbutton (лічильник)  – використовується для введення або зміни числових значень.

Optionbutton (перемикач)  – дозволяє вибрати один з декількох взаємовиключних параметрів або дій. Перемикачі звичайно відображаються групами, забезпечуючи можливість альтернативного вибору.

Checkbox  – надає користувачеві можливість вибору. Прапорець звичайно має два стани: установлене й скинуте.

Togglebutton (вимикач)  – кнопка, яка залишається натиснутою після клацання на ній, і вертається у вихідний стан після повторного клацання.

Frame (рамка)  – використовується для візуального об'єднання яких-небудь елементів керування в групу, показуючи, що ці елементи зв'язані між собою.

Image (рисунок)  – дозволяє вставляти графічні елементи в екранні форми. За допомогою цього елемента можна вставляти зображення із графічних файлів наступних типів: *bmp*; *cur*; *gif*; *ico*; *jpg*; *wmf*.

Multipage (набір сторінок)  – цей елемент керування зовні схожий на набір вкладок і також містить одну або кілька сторінок. Відмінність між ними полягає в тому, що сторінки є формами, що містять власні елементи керування (включаючи набори вкладок), які можна отформатувати всіма способами форматування екранних форм.

Tabstrip (набір вкладок)  – елемент керування, який містить одну або кілька вкладок. Використовується для організації в групі зв'язаної інформації.

Refedit  – цей елемент схожий на звичайне поле введення, але має кнопку із правої сторони поля, яка звертає екранну форму, що дозволяє легко вибрати будь-який діапазон гнізд на робочому аркуші.

Найбільш часто використовуваними є:

- Label (напис);
- Textbox (поле);
- Commandbutton (кнопка керування);

- Image (малюнок);
- Optionbutton (перемикач);
- Checkbox.

1.2.2 Створення форми користувача

Для створення форми користувача необхідно виконати наступні дії:

1) Відкрити новий файл або будь-який файл, у якому ви вже працювали.

2) Виконати команду меню Сервіс ⇒ Макрос ⇒ Редактор Visual Basic. Відкриється вікно редактора Visual Basic.

3. Клацнути на кнопці Insert Userform панелі інструментів вікна редактора Visual Basic або виконати команду Insert ⇒ Userform (Вставка ⇒ Екранна форма). З'явиться нова екранна форма (рис. 1.2).

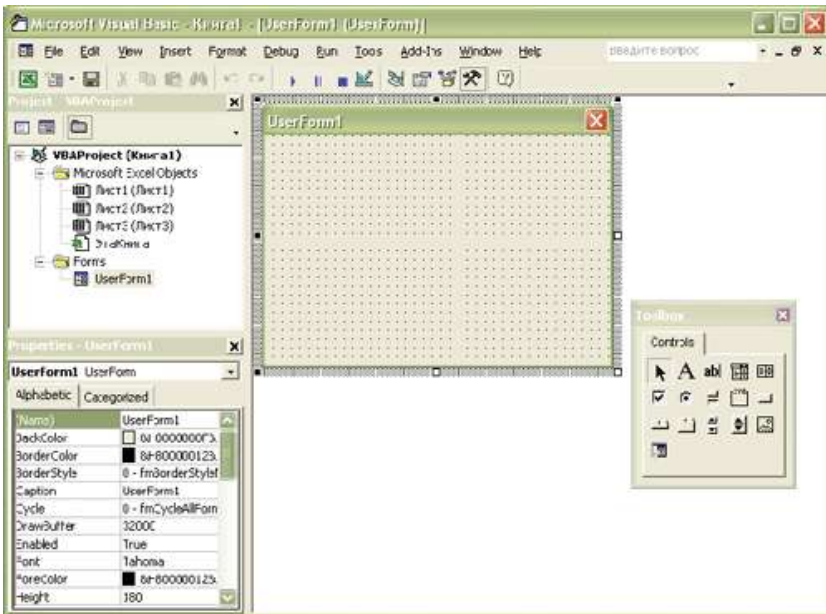


Рисунок 1.2 – Створена екранна форма

Створена екранна форма має за замовченням рядок із заголовком Userform1. Сітка на макеті форми допомагає розміщенню елементів керування, при відображенні форми в додатку сітка не видна.

Після створення форми необхідно переглянути властивості цього об'єкта й змінити установки, якщо це потрібно за умовою завдання. Для відображення вікна властивостей треба вибрати команду View ⇒ Properties Window (Вид ⇒ Вікно властивостей) або клацнути по однійменній кнопці на панелі інструментів .

1.2.3 Тестування й відлагодження програм

Відлагодження програм і обробка помилок завжди виступає як частина процесу розробки. Якби всі програмували ідеально, то необхідність налагодження програм і обробки помилок відпала б сама собою: кожна створювана програма виконувала б усі необхідні дії з першого разу. Як правило, такого не буває, тому в більшості систем розробки є інструменти, за допомогою яких можна розв'язати проблеми, що виникають у процесі програмування. В VBA також є засобу, які дозволяють або виключити помилки при розробці, або задати відгук на помилки при виконанні програм.

Налагодження програм і обробка помилок – це не те саме, але вони тісно зв'язано один з одним.

Налагодження програм – це перевірка й внесення виправлень у програму при її розробці. Налагодження дозволяє ідентифікувати помилки, допущені при програмуванні. Наприклад, синтаксичні помилки в тексті програми, іменах функцій і змінних або логічні помилки.

Обробка помилок – це завдання реакції на помилки, які виникають під час виконання програми. Причиною помилок можуть бути як помилки в самій програмі, так і інші обставини, що перебувають поза сферою впливу програміста. Наприклад, відсутність файлів, до яких відбувається програмний обіг, відмова апаратних засобів або неправильні дії користувача.

Неможливо запобігти виникненню всіх помилок, але слід прагнути до зменшення їх числа. У маленькій програмі досить просто виявити помилку. Однак у міру збільшення розмірів і складності програм знаходити їх стає усе суцужніше. У таких випадках необхідно користуватися засобами налагодження VBA.

Середовище розробки програм на VBA надає користувачеві сучасні зручні засоби налагодження програми.

Припустимо, що вже написаний код вашої процедури. Наступний етап у створенні будь-якої процедури – тестування написаного коду.

Тестування – це процес виконання процедури й дослідження всіх аспектів її роботи.

Мета тестування – перевірити правильність результатів виконання процедури і її реакцію на різноманітні дії користувача.

Якщо під час роботи процедури отримані невірні результати обчислень, непередбачена реакція на ті або інші дії користувача, або взагалі відбулася зупинка виконання, то це говорить про те, що в тексті програми є помилки.

Усі можливі помилки можна розділити на три види:

1. *Помилки компіляції*. Виникають, якщо VBA не може інтерпретувати введений текст, наприклад, при використанні неправильного синтаксису інструкції або завданні невірного імені методу або властивості. Деякі помилки компіляції виявляються при введенні інструкції, а інші — тільки перед виконанням програми. Даний тип помилок звичайно просто ідентифікувати й виправити, оскільки VBA виявляє їх автоматично, а самі помилки очевидні.

2. *Помилки виконання*. Виникають при виконанні програми, тобто після успішної компіляції. Причиною таких помилок може бути відсутність даних або неправильна інформація (наприклад, дані, введені користувачем). Помилки виконання, як і помилки компіляції, легко ідентифікуються VBA. При цьому виводиться інструкція, при виконанні якої відбулася помилка. Помилки даного типу важче усунути: може знадобитися вивести значення змінних або властивостей, а також інші дані, які впливають на успішне виконання програми.

3. *Логічні помилки* сутужніше всього помітити й усунути. Логічні помилки не приводять до припинення компіляції або виконання. Однак вони є причиною того, що програма не видає бажаних результатів. Помилки даного типу ідентифікуються шляхом ретельної перевірки за допомогою засобів налагодження VBA.

Компіляція – це процес перетворення програми, написаної алгоритмічною мовою, у мову машинних кодів. Якщо в програмі є синтаксичні помилки, то процес компіляції припиняється, рядка з помилкою зафарбовуються жовтим кольором і видається відповідне повідомлен-

ня. Для продовження виконання програми необхідно виправити помилку й нажати кнопку “Continue”  на стандартній панелі редактора VBA. Або перервати виконання програми, нажавши на кнопку “Reset” , виправити помилку в програмі, а потім заново запустити її.

При виявленні помилки компілятор видає повідомлення із вказівкою номера помилки. У цьому випадку корисно, скориставшись довідковою системою редактора VBA, визначити характер помилки й виправити її. Редактор Visual Basic відразу реагує на синтаксичні помилки.

Використовуючи панель елементів керування, з незаповненої форми можна сконструювати будь-яке необхідне діалогове вікно. Розміщення елементів керування у формі проводиться наступною послідовністю дій:

- натиснути значок того елемента, який ви прагнете розміститися у формі;
- помістити покажчик миші на те місце, де буде розташовуватися елемент керування;
- нажати ліву кнопку миші й, не відпускаючи її, розтягти, що з'явився прямокутник до потрібних розмірів;
- відпустити кнопку миші, елемент створений.

Після успішного завершення розробки екранної форми можна переглянути в дії отриманий результат за допомогою команди “Run ⇒ Run Sub/Userform” (“Виконати ⇒ Виконати процедуру/екранну форму”) або нажати клавішу F5. Форма відобразиться понад поточну робочу книгу. Клацання на кнопці “Закрити” (перебуває у верхньому правому куті форми) закриє екранну форму й поверне вас у редактор Visual Basic.

Розроблювальний мовою додаток називається проектом. Проект містить у собі не тільки форму з розміщеними на ній керуючими елементами, але й програмні модулі обробки подій, які описують поведінку об'єктів додатка й взаємодію об'єктів між собою.

1.2.4 Діалогові вікна

Основними діалоговими вікнами є:

- вікно повідомлень;
- вікно введення.

Функція для створення **вікна повідомлень** записується так:

Msgbox (Повідомлення [,Атрибути] [,Заголовок]).

Повідомлення – текст, відображуваний у діалогові вікні, є обов'язковим аргументом. Цей рядок повинен бути укладений в подвійні лапки. Текст може містити до 1024 символів. Використання круглих дужок указує на те, що в цьому випадку Msgbox є функцією, що повертає значення.

Атрибути визначають особливості вікна, тобто різні кнопки й значки, відображувані в ньому. Аргумент *Атрибути* дозволяє управляти наступними параметрами вікна повідомлення: кількістю кнопок у вікні, типом кнопок і їх розміщенням у вікні, піктограмою, відображуваною у вікні.

Цей аргумент є цілим числом і для досягнення бажаного результату може бути представлений як сума двох доданків:

Атрибути = *Параметр1* + *Параметр2*.

Значення *Параметр1* установлює число й тип кнопок у вікні повідомлень (табл. 1.1).

Таблиця 1.1 – Можливі значення параметра 1 функції Msgbox

Параметр 1	Набір кнопок
0	Відображається тільки кнопка “OK”
1	Відображаються кнопки “OK” та “Cancel” (Відмінити)
2	Відображаються кнопки “Abort” (Стоп), “Retry” (Повтор) та “Ignore” (Пропустити)
3	Відображаються кнопки “Yes” (Так), “No” (Ні) та “Cancel” (Відмінити)
4	Відображаються кнопки “Yes” (Так), “No” (Ні)
5	Відображаються кнопки “Retry” (Повтор) та “Ignore” (Пропустити)

Якщо аргумент *Атрибути* не визначений, то VBA припускає, що в діалоговому вікні повідомлень присутнє тільки кнопка “OK”.

Значення *Параметр2* визначає вид повідомлення й піктограму, яка міститься у вікні повідомлень (табл. 1.2).

Таблиця 1.2 – Можливі значення параметра 2 функції MsgBox

Параметр 2	Вид повідомлення	Піктограма
16	Відображається значок критичного повідомлення	
32	Відображається знак питання (попередження)	
48	Відображається знак оклику (попередження)	
64	Відображається значок інформаційного повідомлення	

Заголовок – рядок у заголовку вікна повідомлень.

Msgbox як функція повинна повертати значення. Синтаксис функції MsgBox збігається із синтаксисом однойменного оператора. Значення, що повертається, залежить від кнопки, натиснутої користувачем.

У табл. 1.3 наведено список значень, що повертаються функцією MsgBox.

Таблиця 1.3 – Список значень, які повертаються функцією MsgBox

Значення	Натиснута кнопка
1	“OK”
2	“Cancel” (Відмінити)
3	“Abort” (Стоп)
4	“Retry” (Повтор)
5	“Ignore” (Пропустити)
6	“Yes” (Так)
7	“No” (Ні)

Вікно введення, як і вікно повідомлень, може відображатися оператором або функцією.

Функція Inputbox застосовується для введення чисел або тексту. Ця функція відображає діалогове вікно введення, що містить поле введення текст, що й пояснює; її синтаксис такий:

Inputbox(Повідомлення[, Заголовок] [, Умовчання]).

Повідомлення – єдиний обов'язковий аргумент; він служить підказкою користувачеві, яку інформацію необхідно занести в поле введення.

Заголовок – це напис у рядку заголовка вікна введення.

Умовчання – значення, яке буде відображатися в поле введення за замовчуванням, поки користувач не введе своє значення. Якщо цей аргумент вилучити, то поле введення відображається порожнім.

Значення, що повертається даною функцією, є інформація, що вводиться користувачем. значення, що вертається, можна використувати у вікнах повідомлень, помістити в гніздо робочого аркуша, застосувати в обчисленнях і т.д.

На відміну від MsgBox параметр *Атрибути* відсутній, але з'являється параметр *Умовчання*: у ньому задається рядок, який за замовчуванням буде перебувати в текстовім полі. При натисканні кнопки “OK” уміст поля привласнюється змінної, а при натисканні на кнопку “Cancel” вертається рядок нульової довжини.

Варто розуміти, що функція Inputbox повертає рядок, а функція MsgBox повертає значення цілого типу.

1.2.5 Алгоритмічні конструкції мови VBA

Код VBA розташовується у модулях. Модуль складається із процедур. Кожна процедура, що міститься в модулі, є або процедурою-функцією, або процедурою-підпрограмою.

Процедура – будь-яка сукупність коду VBA, розглянута як єдине ціле. Як правило, процедура складається з операторів, що виконують яке-небудь завдання або, що обчислюють значення. Кожна процедура ідентифікується своїм унікальним іменем. Часте виконання процедури є реакцією на яку-небудь *подію*. У цьому випадку говорять, що процедура *обробляє подію*.

Створення першої **процедури** вимагає виконання двох послідовних кроків:

- спочатку треба вставити модуль у робочу книгу;
- потім у цьому модулі створити процедуру.

Для будь-якого створюваного додатка треба створити свій модуль. Додаток може містити кілька модулів, але це не обов'язково.

Наприклад, створити нову процедуру можна в такий спосіб.

1. Відкрийте нову робочу книгу (в Excel) або новий документ у Word.

2. Виконаєте команду **Сервіс ⇒ Макрос ⇒ Редактор Visual Basic**. Відкриється вікно редактора Visual Basic.

3. Відкрийте проект Vbaproject (Книга1). У меню виберіть команду **Insert ⇒ Module** (Вставка ⇒ Модуль). У ваш додаток буде доданий модуль 1.

4. Виконаєте команду **Insert ⇒ Procedure** (Вставка ⇒ Процедура). Відкриється діалогове вікно Add Procedure (Додати процедуру).

5. Уведіть ім'я процедури. У секції Type (Тип) встановіть перемикач Sub (Підпрограма). Натисніть на кнопці "ОК". Нова процедура буде додана в модуль.

Після створення процедури ви можете її відразу **виконати**. Для цього існує кілька шляхів:

1. Вибрати команду меню **Run Sub ⇒ Userform** (Виконати підпрограму ⇒ екранну форму) з меню Run (Виконати) або клацнути на кнопці **Run Sub ⇒ Userform** стандартної панелі інструментів.

2. Натиснути клавішу F5.

3. Створити на будь-якому аркуші Excel автофігуру й прив'язати до неї підготовлену процедуру.

Програма на VBA являє собою послідовність операторів. Для того щоб зробити програму зручною для читання, використовують оператор коментарів. У мові VBA існують два способи введення коментарів: застосування апострофа ('), який можна поставити в будь-якому місці рядка, і зарезервоване слово Rem замість апострофа. При цьому коментується текст до кінця рядка.

Оператор With/End with – звільняє програміста від великої кількості повторень імені того самого об'єкта.

Синтаксис:

With об'єкт

Оператор 1

Оператор 2

.....

Оператор N

End with

Керуючі структури VBA визначають послідовність виконання програми. Без них усі оператори програми будуть виконуватися зліва направо і зверху вниз. Однак іноді потрібно багаторазово виконати деякий набір інструкцій або розв'язати завдання по-іншому, залежно від значення змінних або параметрів, заданих користувачем під час виконання. У цих випадках допомагають конструкції керування й цикли.

Конструкція If . . . Then

Конструкція ***If ... Then*** застосовується, коли необхідно виконати один або групу операторів залежно від деякої умови. Синтаксис цієї конструкції дозволяє задавати її в одному рядку або в декількох рядках програми:

1-й спосіб:

If умова Then вираз

2-й спосіб:

If умова Then

вираз

End If

Звичайна умова є простим порівнянням, але вона може бути будь-яким виразом з обчислюваним значенням. Якщо умова дійсна, то виконуються всі вираження, що коштують після ключового слова ***Then***.

Оператор циклу For (фіксоване число повторень)

Синтаксис:

For змінна = M1 To M2 [Step M3]

оператори

Next

Тут M1, M2, M3 – вирази. Оператор циклу повторює виконання групи операторів, поки змінна (лічильник) змінюється від початкового значення M1 до кінцевого M2 із зазначеним кроком M3. Якщо крок не зазначений, то він покладається рівним 1.

1.2.6 VBA у застосування до MS Word

Пакет Word у комбінації з VBA надає широкі можливості для роботи з текстом або фрагментами тексту. Можна викликати для обробки який-небудь абзац, пропозиція і т.д.

Для доступу до тих або інших фрагментів документа використовують такі об'єкти, як Sections (розділи), Paragraphs (абзаци), Sentences (речення), Words (слова) і ін.

Приклад:

Words(6) – дає можливість перейти до шостого слова в документі.

Sentences(11) – перехід до одинадцятого речення.

Після переходу до певної частини документа можна використовувати об'єкт Font, що дозволяє змінювати властивості шрифту. Цей об'єкт містить методи, що перебувають у панелі меню “Формат ⇒ Шрифт”.

Об'єкт Range є базовим в операції редагування. Його можна розглядати як безперервний фрагмент тексту (ніяк не пов'язаний з виділеною областю), який не залежить від положення курсору. Range містить у собі такі об'єкти, як Words (слова) або Sentences (пропозиції); при звертанні до них сам об'єкт Range указувати не потрібно, тоді як для Sections (розділи) або Paragraphs (абзаци) це робити необхідно.

Об'єкт Range підтримує основні методи, якими користуються при форматуванні тексту (такі, як “Копіювати”, “Вставити”, “Вирізати”, команди меню “Формат”).

Для виконання завдань буде потрібно застосування функцій, що працюють із символьними змінними.

Для знайомства із прикладами кодів доцільно скористатися прийомом, коли ви форматуєте частину документа в режимі запису макросу, а потім аналізуєте код макросу й редагуєте його.

Для запису макросу необхідно:

1. Вибрати в меню “Сервіс | Макрос | Почати запис” (рис. 1.3).

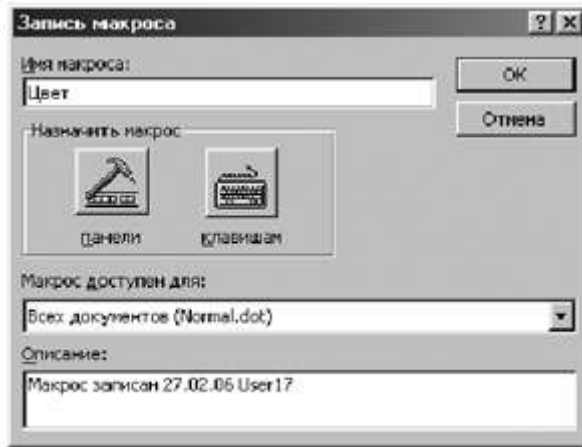


Рисунок 1.3 – Запис макроса

2. Увести у вікні “Запис макросу” у поле “Ім'я макросу” ім'я нового макросу.

3. Вибрати в списку “Макрос доступний для” шаблон або документ, у якому буде зберігатися макрос.

4. Увести опис макросу в поле “Опис”.

5. Увести в поле “Призначити макрос” кнопку “Панелі” (для призначення макросу кнопки панелі інструментів або команди меню); кнопку “Клавішам” (для призначення макросу комбінації клавіш); якщо макросу призначення не потрібно, натисніть кнопку “ОК”. Якщо в процесі роботи ви часто використовуєте який-небудь макрос, то доцільно призначити йому кнопку або комбінацію клавіш.

6. Виконати дії, які потрібно включити в макрос.

Слід пам'ятати, що для запису дій треба використовувати клавіатуру, оскільки дії, зроблені за допомогою миші, не записуються.

7. Натиснути після запису макросу кнопку “Зупинити запис”.

1.3 Завдання до роботи

1.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

1.3.2 За допомогою VBA створити екранну форму для виконання дій «множення», «додавання», «віднімання», «ділення» двох чисел, що задані генератором випадкових чисел.

1.3.2.1 Запустити MS Word.

1.3.2.2 Завантажити VBA (Сервіс $\Rightarrow$ Макрос $\Rightarrow$ Редактор Visual Basic).

1.3.2.3 Перевірити, щоб на екрані з'явилося вікно проекту й вікно властивостей.

1.3.2.4 Виконати команду Вставити форму або Insert $\Rightarrow$ Userform, назвіть форму “Обчислення” і викличте панель “Toolbox”.

1.3.2.5 Спроектуйте у формі 5 командних кнопок “Commandbutton” (X, +, -, :, Выход), три написи “Label” (Первое число, Второе число, Ответ), а також два текстові поля “TextBox” (рис. 1.4). Значення властивостей об'єктів встановіть відповідно до рис. 1.4.

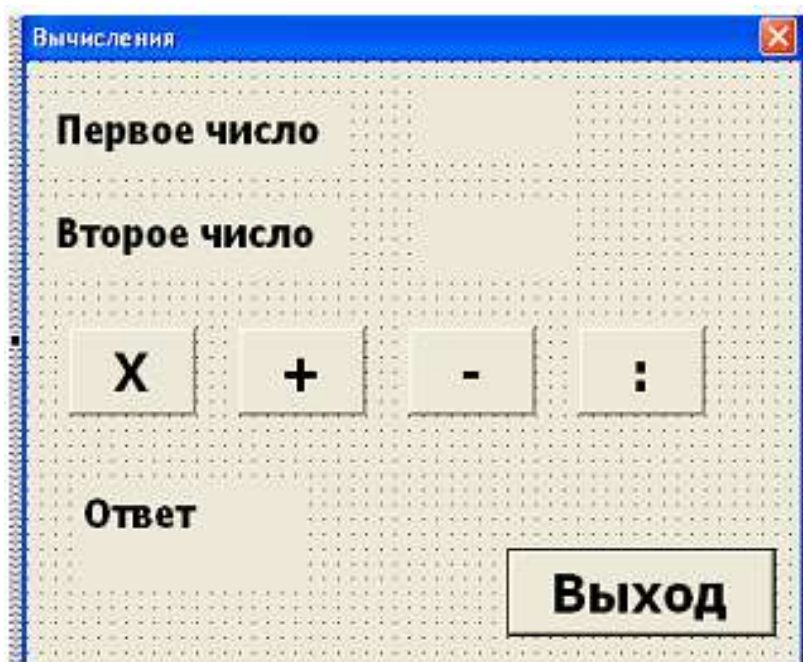


Рисунок 1.4 – Екранна форма до завдання 1.3.2

1.3.2.6 У формі двічі клацніть на кнопці “X”, щоб вивести заготовку програми. Програма починається із заголовка Private Sub і закінчується пропозицією End Sub (кінець процедури). Між цими реченнями запишемо текст програми:

```
Label3.Caption = Str(CInt(TextBox1.Text) * CInt(TextBox2.Text))
```

У цьому рядку виконується операція множення двох чисел, що задає користувач у полях введення TextBox1 та TextBox2.

Для можливості виконання арифметичної операції множення над даними, що подані у вигляді строки (властивість Text об'єктів TextBox1 та TextBox2) виконується перетворення типів з строкового до цілового за допомогою функції CInt.

Оскільки результатом множення є число, а властивість Caption об'єкту Label3 має бути рядком, то виконується перетворення отриманого числа до рядкового типу даних за допомогою функції Str.

1.3.2.7 Аналогічно для кнопок “+”, “-”, “:” і “Вихід” програма буде наступна:

```
Label3.Caption = Str(CInt(TextBox1.Text) + CInt(TextBox2.Text))
```

```
Label3.Caption = Str(CInt(TextBox1.Text) - CInt(TextBox2.Text))
```

```
Label3.Caption = Str(CInt(TextBox1.Text) / CInt(TextBox2.Text))
```

```
UserForm1.Hide.
```

1.3.2.8 Перевірте працездатність форми за допомогою команди “Run ⇒ Run Sub/Userform” (“Виконати | Виконати процедуру/екранну форму”). Також можна виконати перевірку за допомогою кнопки



в меню або нажавши на клавішу F5.

1.3.2.9 В Word створіть кнопку виклику форми й назвіть її “Обчислення”, перейменуйте Документ1 в документ “Обчислення”.

1.3.2.10 Викличте “Панель Елементів керування” і намалюйте з її допомогою кнопку.

1.3.2.11 Виділивши кнопку, викличте через праву кнопку миші “Властивості” і отформатуйте виділену кнопку.

1.3.2.12 Виділіть кнопку “Обчислення”, натисніть на панелі елементів керування кнопку “Вихідний текст” .

1.3.2.13 Запишіть програму виклику форми: Userform1.Show.

1.3.2.14 Збережіть свою роботу.

1.3.3 Програмно визначити кількість абзаців в документі, ввести заголовок тексту (ЗАГОЛОВОК), в кінці документу вивести повідом-

лення червоним кольором та шрифтом Arial розміром 14пт про кількість абзаців у документі (рис. 1.5).

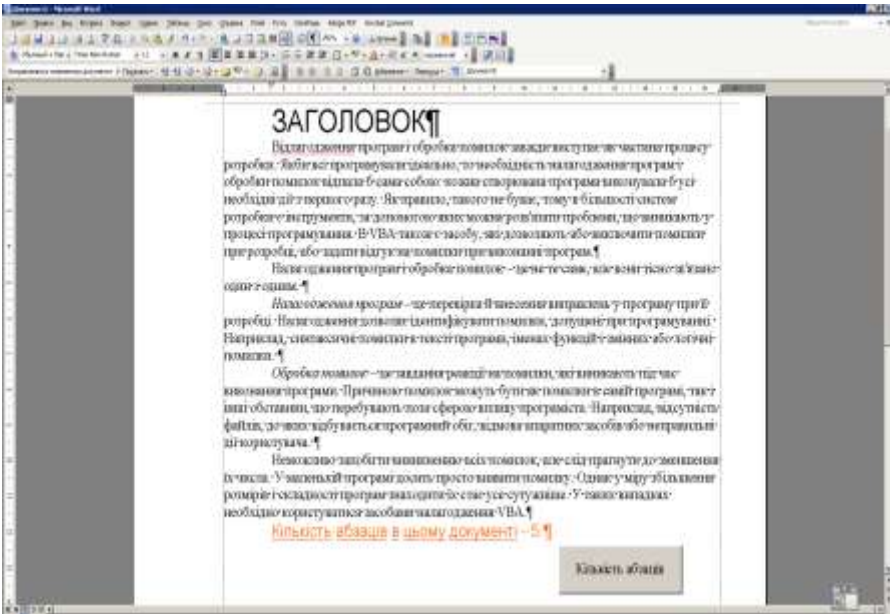


Рисунок 1.5 – Вид документа до завдання 1.3.3

1.3.3.1 Набрати текст у редакторі MS Word.

1.3.3.2 Створити в цьому документі необхідну кнопку.

1.3.3.3 Перейменувати кнопку в “Кількість абзаців”. Для цього на панелі елементів керування викличте “Свойства”. З’явиться панель “Properties”. Виберіть властивість “Caption” і в правому стовпчику наберіть назву.

1.3.3.4 Додати кнопці властивість обтікання по контуру, щоб кнопку можна було переміщати по аркушу. Для цього на об’єкті кнопка варто викликати контекстно-залежне меню й вибрати “Формат об’єкта”. З’явиться діалогове вікно “Формат елемента управління”. На вкладці “Положение” вибрати варіант “По контуру”.

1.3.3.5 Вибрати в контекстно-залежному меню цього об'єкта команду “Исходный текст” або два рази клацнути по кнопці. Відкриється редактор режиму написання програми.

1.3.3.6 У вікні, що відкрилося, оголосити змінні:

a) k – змінна для підрахунку кількості абзаців у документі;
b) Text – змінна для формування тексту-відповіді про кількість абзаців;

c) ZAGOLOVOK – змінна, що визначає область для виведення заголовка.

d) REZULTAT – змінна для збереження параметрів абзацу, в якому буде виводитися повідомлення про кількість абзаців.

1.3.3.7 Визначити кількість абзаців за допомогою методу Count об'єкта Paragraphs.

1.3.3.8 Сформуванати текст виведення повідомлення.

1.3.3.9 Вставити додатковий абзац після останнього абзацу документа для виведення повідомлення.

1.3.3.10 Вивести текст-повідомлення в підготовлений абзац.

1.3.3.11 Сформуванати область для виведення заголовка “ЗАГОЛОВОК”.

1.3.3.12 Вивести назву тексту “ЗАГОЛОВОК”.

1.3.3.13 Установити шрифт і розмір заголовка.

1.3.3.14 Вставити символ кінця абзацу для того, щоб назва й початок тексту були на різних рядках.

Текст розробленої програми з коментарями наведено у табл. 8.4.

Таблиця 1.4 – Текст програми до завдання 1.3.3

Рядок програми	Пункт алгоритму
Public Sub CommandButton1_Click()	Початок
Dim k As Integer	1.3.3.6 а
Dim Text As String	1.3.3.6 б
Dim ZAGOLOVOK As Range	1.3.3.6 в
Dim REZULTAT As Range	1.3.3.6 г
k = ActiveDocument.Paragraphs.Count	1.3.3.7

Продовження табл. 1.4

Рядок програми	Пункт алгоритму
Text = "Кількість абзаців в цьому документі - " & k & "."	1.3.3.8
ActiveDocument.Paragraphs(k).Range.InsertParagraphAfter	1.3.3.9
ActiveDocument.Paragraphs(k + 1).Range.InsertBefore Text	1.3.3.10
Set REZULTAT = ActiveDocument.Paragraphs(k + 1).Range With REZULTAT .Font.Name = "Arial" .Font.Size = 14 .Font.ColorIndex = wdDarkRed End With	Установка параметрів області для виведення заголовка: – шрифт – Arial – розмір шрифту – 14пт; – колір – червоний
Set ZAGOLOVOK = ActiveDocument.Range(Start:=0, End:=0)	1.3.3.11
With ZAGOLOVOK	Початок структури With
.InsertBefore "ЗАГОЛОБОК"	1.3.3.12
.Font.Name = "Arial"	1.3.3.13
.Font.Size = 24	1.3.3.13
.InsertParagraphAfter	1.3.3.14
End With	Кінець структури With
End Sub	Кінець

1.3.4 За допомогою VBA виконати одне з завдань, наведених у п.1.3.4.1–1.3.4.10, узгодивши номер завдання з викладачем.

1.3.4.1 Створити форму користувача, розмістити в ній елементи керування Label (напис), CommandButton (кнопка керування). При натисканні на кнопку “Запуск” повинне генеруватися випадкове число. Це число повинне відображатися в елементі Label (напис). Також створити елементи для обчислення кореня квадратного з цього числа, його синусу і косинусу та виведення відповідних результатів. Крім цього, у формі вивести поточну дату. По натисканню на кнопку “Ви-

хід” форма повинна прибиратися з екрана. Для запуску форми на листі MS Excel або у документі MS Word необхідно створити кнопку з написом “Функції VBA”.

1.3.4.2 Використовуючи діалоги MsgBox та InputBox для введення та виведення значень, підрахуйте площу круга, якщо відомий його радіус. Для цього у документі MS Word або на листі MS Excel створіть кнопку та приєднайте до неї відповідну програму.

1.3.4.3 Використовуючи форму користувача та елементи керування (кнопки, поля для введення/виведення) для введення та виведення значень, обчисліть периметр та площу прямокутника, якщо відомі його сторони. Для запуску форми у документі MS Word або на листі MS Excel необхідно створити кнопку з відповідним написом.

1.3.4.4 Спроектуйте у документі MS Word або на листі MS Excel кнопки виклику форм, за допомогою яких можна скласти, відняти, помножити (і т.ін.) два числа, введені за допомогою діалогових вікон. Результат виводити у вікні виведення (див. рис. 8.6).

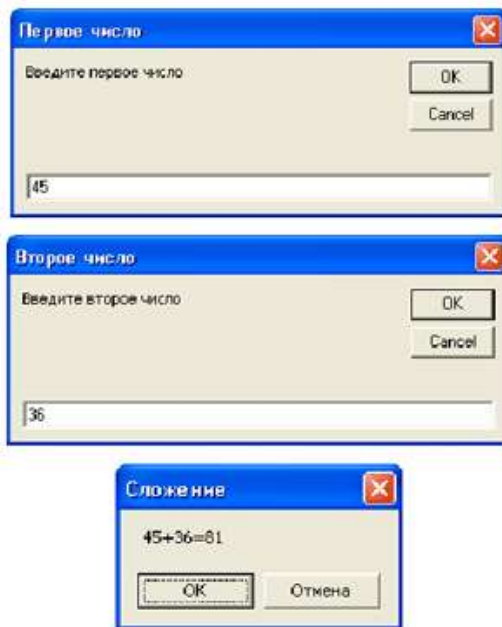


Рисунок 1.6 – Вид діалогових вікон для виконання завдання 1.3.4.4

1.3.4.5 На Листе1 книги MS Excel в чарунки A1–A20 занести випадковим чином числа з інтервалу (–50; 50).

Написати програму на VBA для знаходження максимального і мінімального елементів, суми значень з комірок заданого діапазону, середнього значення.

Для цього можна використовувати стандартні функції MS Excel. Викликати їх можна за допомогою об'єкту Application: Application.Функциярабочеголиста(Аргументи).

1.3.4.6 Написати програму, що в заданому користувачем абзаці документу MS Word обчислює кількість букв «а» та наприкінці документа (або після абзацу, у якому відбувався підрахунок) виводить повідомлення про кількість букв.

1.3.4.7 Написати програму, що в документі MS Word обчислює кількість заданих користувачем слів та наприкінці документа виводить повідомлення про обчислену кількість слів.

1.3.4.8 Написати програму, що в документі MS Word знаходить усі задані користувачем слова та встановлює наступні параметри шрифту цих слів: тип шрифту – Arial, розмір – 20 пт, жирний, колір – зелений.

Також розроблювана програма повинна виділяти курсивом всі слова абзацу, що містить найбільшу кількість заданих слів.

1.3.4.9 Написати програму, що в документі MS Word знаходить абзац з найбільшою кількістю речень, змінює колір шрифту тексту знайденого абзацу на червоний та наприкінці документу виводить повідомлення про номер знайденого абзацу та про кількість речень в ньому.

Для цього в кожному абзаці обчислюється кількість крапок (кількість речень), після чого виходячи з розрахованої кількості крапок обирається абзац з найбільшою кількістю речень.

1.3.4.10 Написати програму, що в документі MS Word знаходить слова-паліндроми та виділяє їх червоним кольором (до слів-паліндромів належать слова, що пишуться однаково зліва направо та справа наліво, наприклад: ШАЛАШ, НАГАН, МАДАМ та ін.).

1.3.5 Оформити звіт з роботи.

1.3.6 Відповісти на контрольні питання.

1.4 Зміст звіту

1.4.1 Тема та мета роботи.

1.4.2 Завдання до роботи.

1.4.3 Короткі теоретичні відомості.

1.4.4 Текст розробленого програмного забезпечення.

1.4.5 Результати виконання роботи.

1.4.6 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

1.5 Контрольні запитання

1.5.1 Що таке Visual Basic for Application?

1.5.2 Поясніть призначення Visual Basic for Application.

1.5.3 Як запустити редактор Visual Basic в Ms Word?

1.5.4 Як розташовувати елементи керування на формі?

1.5.5 Які основні елементи керування Ви знаєте?

1.5.6 Опишіть етапи створення користувацької форми.

1.5.7 Як привести створену форму в «дію»?

1.5.8 Як зробити так, щоб при натисканні на кнопку форма зникла?

1.5.9 Як вивести на екран повідомлення?

1.5.10 Як створити процедуру?

1.5.11 Як реалізуються коментарії у VBA?

1.5.12 Які існують керуючі структури у VBA?

1.5.13 Чи дозволяють засоби VBA керувати форматуванням в Word?

1.5.14 Як дізнатися про функції VBA роботи з текстом в Word?

1.5.15 Як запустити форму на виконання?

1.5.16 Які існують типи помилок у VBA?

1.5.17 Як змінити безпеку захисту від макросів?

1.5.18 Які клавіші швидкого доступу використовуються при роботі з VBA?

1.5.19 Проаналізуйте керуючі структури у VBA. Порівняйте їх з керуючими структурами інших мов програмування.

1.5.20 У яких випадках доцільно використовувати VBA?

2 САМОСТІЙНА РОБОТА № 2

РОЗРОБКА ТЕКСТОВОГО РЕДАКТОРУ У C#

2.1 Мета роботи

Закріпити навички розробки програмного забезпечення за допомогою C#.

2.2 Основні теоретичні відомості

Елементи керування – це компоненти, що забезпечують взаємодію між користувачем і програмою. Серед Visual Studio.NET надає велику кількість елементів, які можна згрупувати з кількох функціональних груп.

Група меню. Багато користувачів налаштовують інтерфейс додатків на свій смак: одним подобається наявність певних панелей інструментів, іншим – індивідуальне розташування вікон. Але в будь-якому додатку буде присутній меню, що містить у собі доступ до всіх можливостей і налаштувань програми. Елементи MainMenu, ContextMenu представляють собою готові форми для внесення заголовків і пунктів меню. Практична робота з різними елементами управління буде розглянута в цій та наступних лекціях.

Діалогові вікна. Виконуючи різні операції з документом – відкриття, збереження, друк, попередній перегляд, – ми стикаємося з відповідними діалоговими вікнами. Розробникам .NET не доводиться займатися створенням вікон стандартних процедур: елементи OpenFileDialog, SaveFileDialog, ColorDialog, PrintDialog містять вже готові операції.

Елементи управління Button, LinkLabel, ToolBar реагують на натискання кнопки миші і негайно запускають будь-яку дію. Найбільш поширена група елементів.

Більшість додатків надають можливість користувачу вводити текст і, у свою чергу, виводять різну інформацію у вигляді текстових записів. Елементи TextBox, RichTextBox приймають текст, а елементи Label, StatusBar виводять її. Для обробки введеного користувачем тексту, як правило, слід натиснути на один або кілька елементів з групи командних об'єктів.

Група перемикачів. Додаток може містити кілька визначених варіантів виконання дії або завдання; елементи управління цієї групи надають можливість вибору користувачеві. Це одна з найбільш великих груп елементів, в яку входять ComboBox, ListBox, ListView, TreeView, NumericUpDown і багато інших.

Група контейнерів. З елементами цієї групи наявність практично ніколи не зв'язуються, але вони мають велике значення для організації інших елементів управління, їх угруповання та загального дизайну форми. Як правило, елементи цієї групи, розташовані на формі, служать підкладкою кнопкам, текстових полів, списками – тому вони і називаються контейнерами. Елементи Panel, GroupBox, TabControl, крім усього іншого, поділяють можливості програми на логічні групи, забезпечуючи зручність роботи.

Група графічних елементів. Навіть самий простий додаток Windows містить графіком-іконки, заставку, вбудовані зображення. Для розміщення і відображення їх на формі використовуються елементи для роботи з графікою – Image List, PictureBox.

2.2.1 Створення головного меню

Більшість Windows-додатків оснащена головним меню, яке представляє собою ієрархічну структуру виконуваних функцій і команд. Практично всі функції, які можна здійснити за допомогою елементів управління, мають свій аналог у вигляді пункту меню. Для створення головного меню використовується елемент управління MainMenu, розташований на панелі інструментів Toolbox. Створіть новий додаток і назвіть його NotepadCSharp. Встановіть наступні властивості форми. Приклади створення головного меню наведено у табл. 8.1.

Таблиця 2.1 – Створення головного меню

Form1, форма, властивість	Значення
Name	frmmain
Icon	 Code\Glava2\NotepadCSharp\Icon\README.ICO
Text	Notepad C#
WindowState	Maximized

Переглядаємо елемент управління на форму. Нам належить заповнити рядки меню наступними пунктами (рис.2.1):

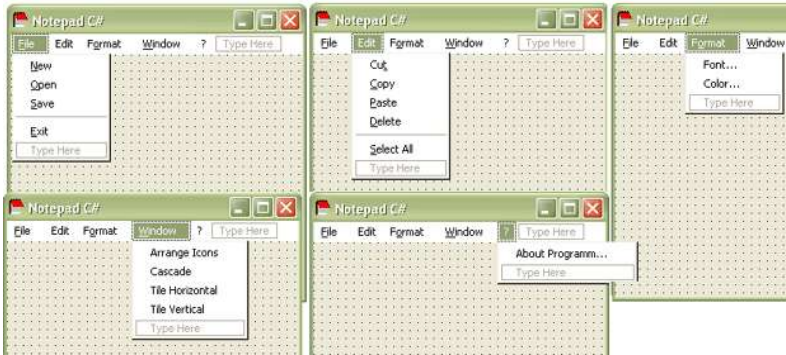


Рисунок 2.1 – Пункти головного меню програми Notepad C #

Кожен пункт головного меню має своє вікно властивостей, у якому, подібно до інших елементів керування, задаються значення властивостей Name і Text (рис. 2.2). У полі Text перед словом New стоїть знак & – так званий амперсанд, який вказує, що N повинне бути підкреслена і буде частиною вбудованого клавіатурного інтерфейсу Windows. Коли користувач на клавіатурі натискає клавішу Alt і потім N, виводиться підменю New.

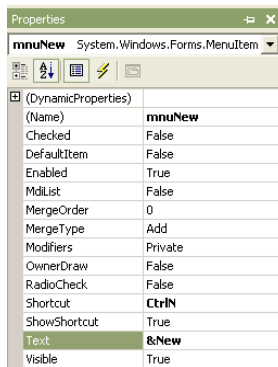


Рисунок 2.2 – Властивості пункту меню New

У Windows є ще інтерфейс для роботи з так званими швидкими клавішами, або акселераторами. Поєднання клавіш вказують з перерахування Shortcut. Слід призначати стандартним пунктам загальноприйнятій поєднання клавіш. Горизонтальна розділова лінія використовується в тих випадках, коли треба візуально відокремити подібні групи завдань; для її появи у властивості Text пункту меню просто вводимо знак тире. Для використання пунктів меню в коді, їм також призначають імена (властивість Name), які особливо важливі, тому що пунктів меню звичайно буває багато. Властивості пунктів меню в додатку Notepad C # наводяться у табл. 2.2.

Таблиця 2.2. Пункти головного меню програми Notepad C #

Name	Text	Shortcut
mnuFile	&File	
mnuNew	&New	CtrlN
mnuOpen	&Open	CtrlO
mnuSave	&Save	CtrlS
menuItem5	-	
mnuExit	&Exit	AltF4
mnuEdit	&Edit	
mnuCut	Cu&t	CtrlX
mnuCopy	&Copy	CtrlC
mnuPaste	&Paste	CtrlV
mnuDelete	&Delete	Del
mnuSelectAll	&SelectAll	CtrlA
mnuFormat	F&ormat	
mnuFont	Font...	
mnuColor	Color...	
mnuWindow	&Window	
mnuArrangeIcons	Arrange Icons	
mnuCascade	Cascade	
mnuTileHorizontal	Tile Horizontal	
mnuTileVertical	Tile Vertical	
mnuHelp	?	
mnuAbout	About Programm...	

Ви можете самостійно вибрати сполучення клавіш, не вказані в таблиці 2.2, для відповідних пунктів меню.

2.2.2 Створення MDI-додатків

Такі програми, як блокнот і Microsoft Paint, відносяться до SDI (Single – Document Interface) додаткам, здатним працювати тільки з одним документом. Інші, такі як Microsoft Word або Adobe Photoshop, підтримують роботу відразу з декількома документами і називаються MDI (Multiple – Document Interface) додатками (рис. 2.3 та 2.4).

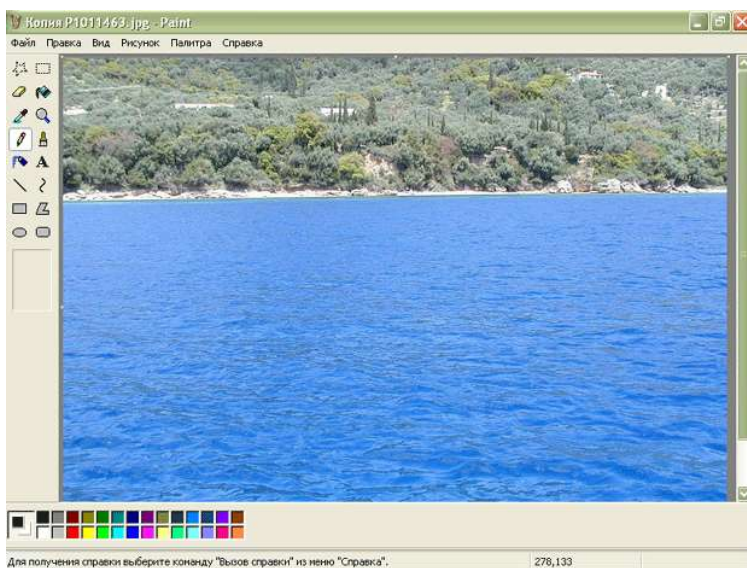


Рисунок 2.3 – SDI додаток Microsoft Paint

У MDI-додатках головна форма містить у собі кілька документів, кожен з яких є полотном в графічних програмах або полем для тексту в редакторах. Продовжимо роботу над додатком Notepad C #. У вікні Solution Explorer клацаємо правою кнопкою на імені проекту і в контекстному меню вибираємо Add / Add Windows Form У вікні називаємо форму – blank.cs. У нашому проєкті з'явилася нова форма - будемо називати її дочірньою. У режимі

дизайну перетаскиваем на неї елемент управління RichTextBox: на відміну від елемента textBox, розмір вмісту тексту в ньому не обмежується 64 Кб; крім того, RichTextBox дозволяє редагувати колір тексту, додавати зображення. Властивості Dock цього елемента встановлюємо значення Fill (рис. 2.5):



Рисунок 2.4 – MDI-додаток Adobe Фотошоп

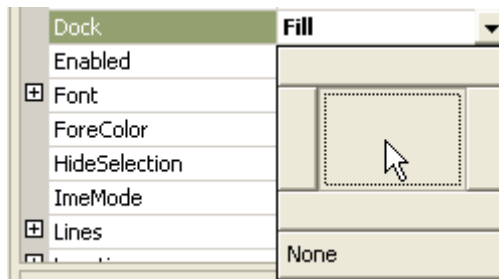


Рисунок 2.5 – Властивість Dock елемента RichTextBox

Переходимо в режим дизайну форми `Frmmain` і встановлюємо властивості `IsMdiContainer` значення `true`. Колір форми при цьому стає темно-сірим. Нові документи будуть у нас з'являтися при натисканні пункту меню `New` (якому ми встановили поєднання клавіш `Ctrl + N`), тому двічі клацаємо в цьому пункті і переходимо в код:

```
private void mnuNew_Click(object sender, System.EventArgs e)
{
    blank frm = new blank();
    frm.MdiParent = this;
    frm.Show();
}
```

Запускаємо програму. Тепер при натисканні клавіш `Ctrl + N` або виборі пункту меню `New` з'являється кілька вікон, розташованих каскадом. Однак заголовок у них всіх однаковий – `blank`. При створенні декількох документів, наприклад, у `Microsoft Word`, вони називаються `ДокументN`, де `N` – номер документа. Перемикаємося в код форми `blank`, і в класі `blank` оголошуємо змінну `DocName`:

```
public string DocName = "";
```

Перемикаємося в код форми `frmmain` і в класі `frmmain` оголошуємо змінну `openDocuments`:

```
private int openDocuments = 0;
```

Надаємо змінної `DocName` частину назви за шаблоном, до якого включено лічильник числа відкритих документів, потім це значення передаємо властивості `Text` створюваної форми `frm`:

```
private void mnuNew_Click(object sender, System.EventArgs e)
{
    blank frm = new blank();
    frm.DocName = "Untitled " + ++openDocuments;
    frm.Text = frm.DocName;
    frm.MdiParent = this;
    frm.Show();
}
```

Запускаємо програму. Тепер нові документи містять різні заголовки (рис. 2.6).

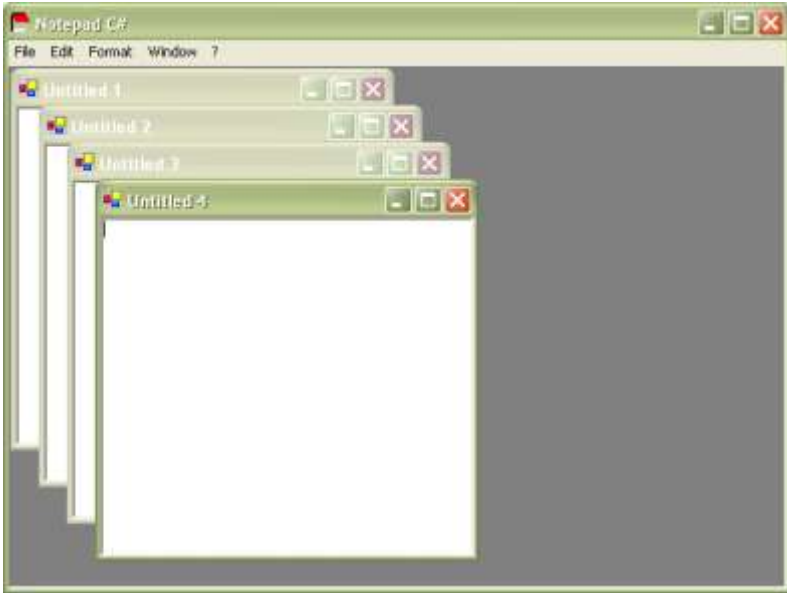


Рисунок 2.6 – Нові документи мають впорядковані назви

2.2.3 Перерахування MdiLayout

При роботі з декількома документами в MDI-додатках зручно впорядковувати їх на екрані. Можна, звичайно, розподіляти форми вручну, але при роботі з великою кількістю документів це представляється скрутним. У пункті меню Window реалізуємо процедуру вирівнювання вікон.

Створюєння обробників:

```
private void mnuArrangeIcons_Click(object sender, System.EventArgs e)
{
    this.LayoutMdi(MdiLayout.ArrangeIcons);
}

private void mnuCascade_Click(object sender, System.EventArgs e)
{
    this.LayoutMdi(MdiLayout.Cascade);
}
```

```
private void mnuTileHorizontal_Click(object sender, System.EventArgs e)
{
    this.LayoutMdi(MdiLayout.TileHorizontal);
}

private void mnuTileVertical_Click(object sender, System.EventArgs e)
{
    this.LayoutMdi(MdiLayout.TileVertical);
}
```

Метод `LayoutMdi` містить перерахування `MdiLayout`, що містить чотири члени. `ArrangeIcons` перемикає фокус на обрану форму, у властивості `MdiList` пункту меню `ArrangeIcons` встановлюємо також значення `true`. При відкритті кількох нових документів вікна розташовуються каскадом (рис. 2.6), їх можна розташувати горизонтально – значення `TileHorizontal` або вертикально – значення `TileVertical`, а потім знову повернути каскадне розташування – `Cascade` (рис. 2.7):

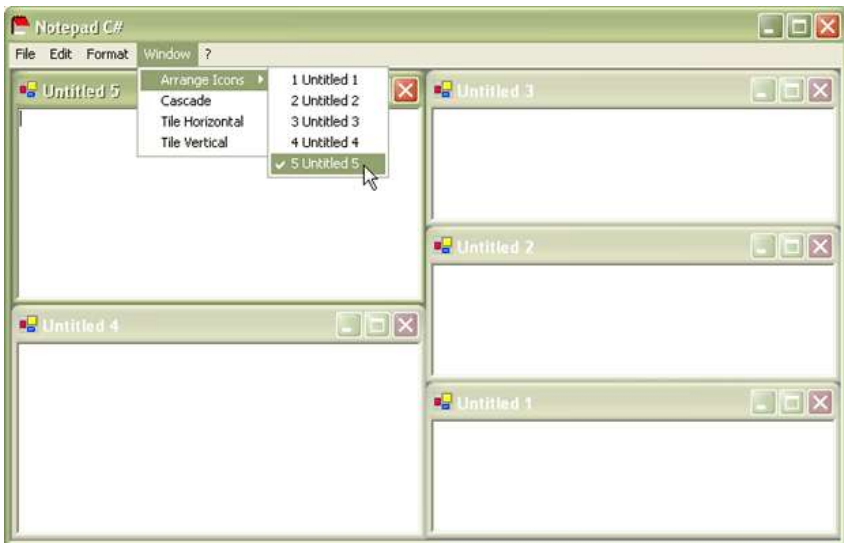


Рисунок 2.7 – Розташування вікон `TileHorizontal` і пункт меню `ArrangeIcons`

2.2.4 Вирізання, копіювання і вставка текстових фрагментів

З додатком працювати буде зручніше, якщо при створенні нового документа він одразу буде займати всю область головної форми. Для цього встановимо властивість `WindowState` форми `blank` `Maximized`. Тепер приступимо до створення обробників для стандартних операцій вирізання, копіювання і вставки. Елемент управління `RichTextBox` має властивість `SelectedText`, яке містить виділений фрагмент тексту. На підставі цієї властивості і будуть реалізовані дії по роботі з текстом. У коді форми `blank` оголошуємо змінну `BufferText`, в якій буде зберігатися буферизована фрагмент тексту:

```
private string BufferText = "";
```

Далі створюємо відповідні методи:

```
public void Cut()
{
    this.BufferText = richTextBox1.SelectedText;
    richTextBox1.SelectedText = "";
}

public void Copy()
{
    this.BufferText = richTextBox1.SelectedText;
}

public void Paste()
{
    richTextBox1.SelectedText = this.BufferText;
}

public void SelectAll()
{
    richTextBox1.SelectAll();
}

public void Delete()
{
    richTextBox1.SelectedText = "";
    this.BufferText = "";
}
```

Перемикаємося в режим дизайну форми і створюємо обробник для пунктів меню:

```
private void mnuCut_Click(object sender, System.EventArgs e)
{
    blank frm = (blank)this.ActiveMdiChild;
    frm.Cut();
}

private void mnuCopy_Click(object sender, System.EventArgs e)
{
    blank frm = (blank)this.ActiveMdiChild;
    frm.Copy();
}

private void mnuPaste_Click(object sender, System.EventArgs e)
{
    blank frm = (blank)this.ActiveMdiChild;
    frm.Paste();
}

private void mnuDelete_Click(object sender, System.EventArgs e)
{
    blank frm = (blank)this.ActiveMdiChild;
    frm.Delete();
}

private void mnuSelectAll_Click(object sender, System.EventArgs e)
{
    blank frm = (blank)this.ActiveMdiChild;
    frm.SelectAll();
}
```

Властивість `ActiveMdiChild` перемикає фокус на поточну форму, якщо їх відкрито кілька, і викликає один з методів, визначених у дочірньої формі. Запускаємо програму. Тепер ми можемо виконувати всі стандартні операції з текстом.

2.2.5 Контекстне меню

Контекстне меню, яке дублює деякі дії основного меню, - не найшвидший спосіб роботи з програмою, але найзвичайніший для користувача. Елемент управління `TextBox` містить в собі просте контекстне меню, яке дублює дії підменю `Edit`. Для того щоб

переконатися в цьому, достатньо нанести цей елемент керування на форму і запустити додаток (рис. 2.8):

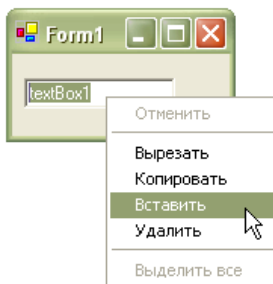


Рисунок 2.8 – Контекстне меню елемента TextBox

У нашому застосуванні Notepad C # в якості текстового елемента ми використовуємо RichTextBox. Додамо елемент управління contextMenu з вікна ToolBox на форму blank. Додаємо пункти контекстного меню точно так само, як ми це робили для головного меню (рис. 2.9):

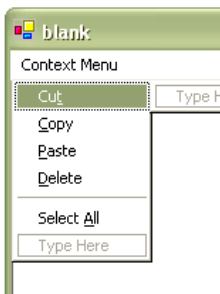


Рисунок 2.9 – Пункти контекстного меню

Властивість Text і Shortcut пунктів меню залишаємо незмінними. Якщо ми встановимо потім для властивості ShowShortcut значення false, то сполучення клавіш будуть працювати, але в самому меню відображатися не будуть – як на рис. 2.8. Властивість Name буде формуватися наступним чином: для пункту Cut – cmnuCut, для Copy – cmnuCopy і т.д.

У обробнику пунктів просто викликаємо відповідні методи:

```

private void cmnuCut_Click(object sender, System.EventArgs e)
{
    Cut();
}

private void cmnuCopy_Click(object sender, System.EventArgs e)
{
    Copy();
}

private void cmnuPaste_Click(object sender, System.EventArgs e)
{
    Paste();
}

private void cmnuDelete_Click(object sender, System.EventArgs e)
{
    Delete();
}

private void cmnuSelectAll_Click(object sender, System.EventArgs e)
{
    SelectAll();
}

```

Останнє, що нам залишилося зробити, – це визначити, де буде з'являтися контекстне меню. Елемент RichTextBox, так само як і форми frmmain і blank, має властивість ContextMenu, де ми і вказуємо contextMenu1, оскільки нам потрібно відображати меню саме в текстовому полі. Запускаємо додаток – тепер у будь-якій точці тексту доступно меню (рис. 2.10).

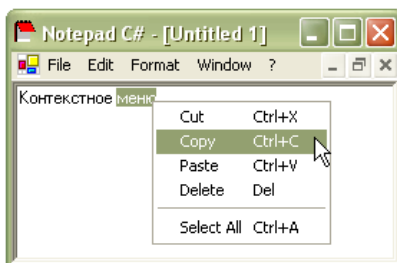


Рисунок 2.10 – Контекстне меню

2.2.6 Діалогові вікна

Середовище Visual Studio .NET містить готові діалогові вікна, що пропонують вибрати файл для відкриття або шлях на диску для збереження поточного файлу (рис. 2.11).

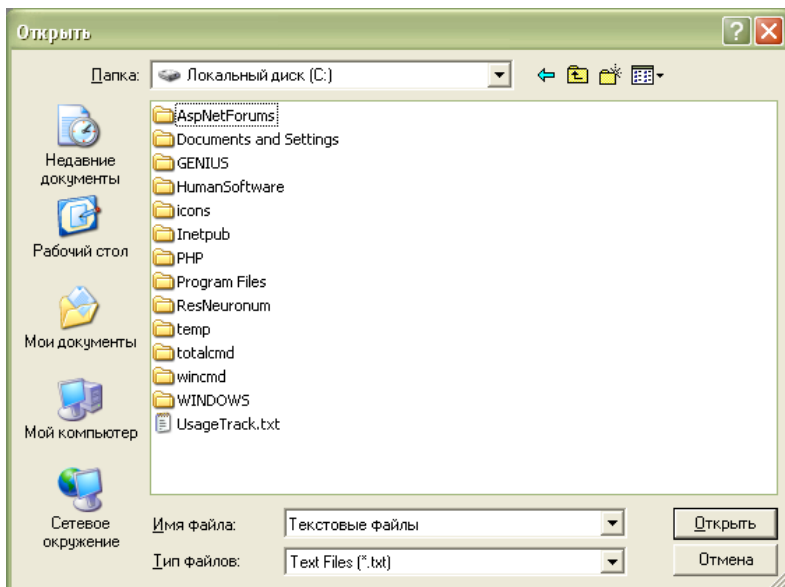


Рисунок 2.11 – Діалогове вікно відкриття файлу

OpenFileDialog. Додайте на форму frmmain елемент управління OpenFileDialog з вікна панелі інструментів ToolBox. Подібно елементу MainMenu, він буде розташовуватися на панелі невидимих компонентів (рис. 2.12):

Властивість FileName задає назву файлу, яка буде знаходитися в полі "Ім'я файлу:" при появі діалогу. На рис. рис. 2.12 назву на цьому полі – "Текстові файли", оскільки на рис. рис. 2.11 був введений саме цей текст. Властивість Filter задає обмеження файлів, які можуть бути обрані для відкриття – у вікні будуть показуватися тільки файли з заданим розширенням. Через вертикальну розділову лінію можна задати зміну типу розширення, яке відображатиметься у випадаючому

списку "Тип файлів". Тут введено Text Files (*.txt) | *.txt | All Files (*.*)|*.* що означає огляд або текстових файлів, або всіх. Властивість InitialDirectory дозволяє задати директорію, звідки буде починатися огляд. Якщо це властивість не встановлено, вихідної директорією буде робочий стіл.

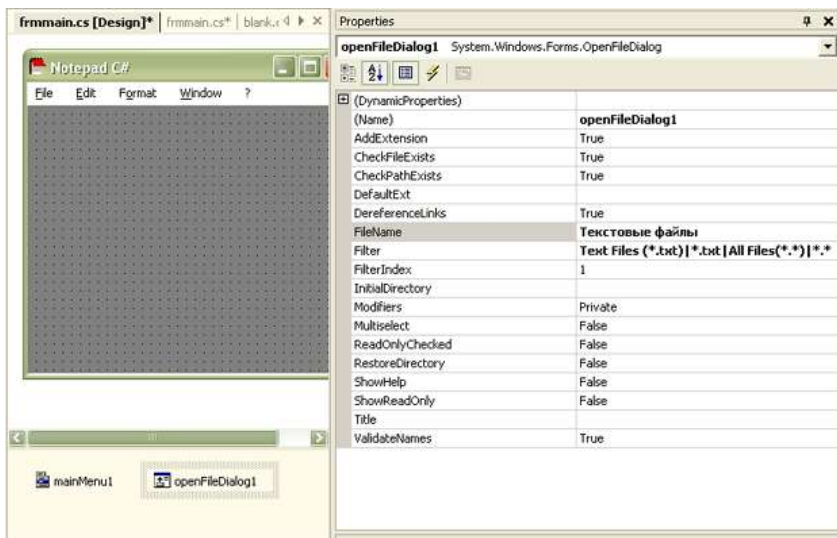


Рисунок 2.12 – Додавання OpenFileDialog на форму

Для роботи з файловими потоками в коді форми blank підключаємо простір імен System.IO:

```
using System.IO;
```

У методі Open зчитуємо вміст файлу в RichTextBox:

```
public void Open(string OpenFileName)
{
    if (OpenFileName == "")
    {
        return;
    }
    else
    {
        StreamReader sr = new StreamReader(OpenFileName);
        richTextBox1.Text = sr.ReadToEnd();
    }
}
```

```
sr.Close();
DocName = OpenFileName;
}
```

Приклад 2.1.

Додамо обробник пункту меню Open форми frmmain:

```
private void mnuOpen_Click(object sender, System.EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        blank frm = new blank();
        frm.Open(openFileDialog1.FileName);
        frm.MdiParent = this;
        frm.DocName = openFileDialog1.FileName;
        frm.Text = frm.DocName;
        frm.Show();
    }
}
```

Приклад 2.2.

Запускаємо програму і відкриваємо текстовий файл, збережений у форматі блокнота (рис. 2.13).



Рисунок 2.13 – Заголовок форми представляє собою адресу та ім'я відкритого файлу

Для коректного читання кирилиці текст в блокноті повинен бути збережений в кодуванні Unicode. На жаль, вбудовані діалогові вікна OpenFileDialog Visual Studio .NET не містять додаткового поля, що дозволяє вибирати кодування файлу при його відкритті або збереженні, як це реалізовано, наприклад, в блокноті.

SaveFileDialog. Для збереження файлів додаємо на форму frmmain елемент управління saveFileDialog1. Властивості цього діалогу в точності такі ж, як у OpenFileDialog (рис. 2.12). Переходимо в код форми blank:

```
public void Save(string SaveFileName)
{
    if (SaveFileName == "")
    {
        return;
    }
    else
    {
        StreamWriter sw = new StreamWriter(SaveFileName);
        sw.WriteLine(richTextBox1.Text);
        sw.Close();
        DocName = SaveFileName;
    }
}
```

Приклад 2.3.

Додаємо обробник пункту меню Save форми frmmain:

```
private void mnuSave_Click(object sender, System.EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        blank frm = (blank)this.ActiveMdiChild;
        frm.Save(saveFileDialog1.FileName);
        frm.MdiParent = this;
        frm.DocName = saveFileDialog1.FileName;
        frm.Text = frm.DocName;
    }
}
```

Запускаємо програму. Тепер файли можна відкривати, редагувати і зберігати. Однак, при збереженні внесених змін у вже збереженому файлі замість його перезапису знову з'являється вікно SaveFileDialog. Змінимо нашу програму так, щоб можна було зберігати й перезаписувати файл. У конструкторі форми frmmain після InitializeComponent відключимо доступність пункту меню Save:

```
mnuSave.Enabled = false;
```

Перемикаємося в режим дизайну форми frmmain і додаємо пункт меню Save As після пункту Save. Встановлюємо наступні властивості цього пункту: Name – mnuSaveAs, Shortcut – CtrlShiftS, Text Save & As. У обробнику Save As вставляємо вирізаний обробник пункту Save і додаємо включення доступності Save:

```
mnuSave.Enabled = true;
```

Зберігати зміни потрібно як у щойно збережених документах, так і в документах, створених раніше і відкритих для редагування. Тому додамо в метод Open включення доступності пункту меню Save:

```
private void mnuOpen_Click(object sender, System.EventArgs e)
{
    mnuSave.Enabled = true;
}
```

У обробнику пункту Save додамо просту перезапис файлу – виклик методу Save форми blank:

```
private void mnuSave_Click(object sender, System.EventArgs e)
{
    blank frm = (blank)this.ActiveMdiChild;
    frm.Save(frm.DocName);
}
```

Запускаємо програму. Тепер, якщо ми працюємо з незбережених документів, пункт Save неактивний (рис. 2.14), після збереження він стає активним (рис. 2.15) і, крім того, працює поєднання клавіш Ctrl + S. Можна зберегти копію поточного документа, знову скориставшись пунктом меню Save As (рис. 2.16).



Рисунок 2.14 – Новий документ

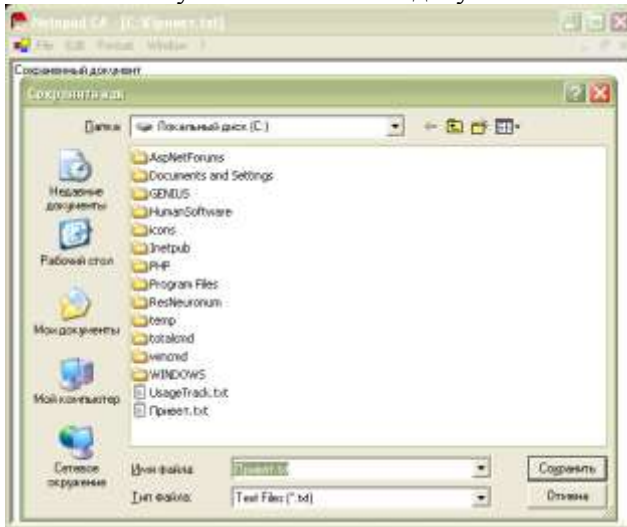


Рисунок 2.15 – Збережений документ



Рисунок 2.16 – Збереження копії

Звичайно, це завдання можна вирішити більш елегантно – поєднавши обидва пункти збереження документа в один. Але мені хотілося показати різницю між простим збереженням і перезаписом файлів, крім того, ми навчилися відключати доступність пунктів меню.

2.2.7 Збереження файлу при закритті форми

Кожного разу, коли ми закриваємо документ Microsoft Word, у який внесли зміни, з'являється вікно попередження, що пропонує зберегти документ. Додамо аналогічну функцію в наш додаток. У класі `blank: System.Windows.Forms.Form` форми `blank` створюємо змінну, яка буде фіксувати збереження документа:

```
public bool IsSaved = false;
```

У обробник методів `Save` і `Save As` форми `frmmain` додаємо зміну значення цієї змінної:

```
private void mnuSave_Click(object sender, System.EventArgs e)
{
    ...
    frm.IsSaved = true;
}

private void mnuSaveAs_Click(object sender, System.EventArgs e)
{
    ...
    frm.IsSaved = true;
}
```

Переходимо в режим дизайну форми `blank` і у вікні властивостей перемикається на події форми, клацнувши на значок з блискавкою. У полі події `Closing` двічі клацаємо і переходимо в код:

```
private void blank_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    if(IsSaved == true)
        if(MessageBox.Show("Do you want save changes in " + this.DocName + "?",
            "Message", MessageBoxButtons.YesNo,
            MessageBoxIcon.Question) == DialogResult.Yes)
        {
            this.Save(this.DocName);
        }
}
```

Приклад 2.4.

Запускаємо програму. При закритті форми blank або frmmain з'являється вікно попередження (рис. 2.17):

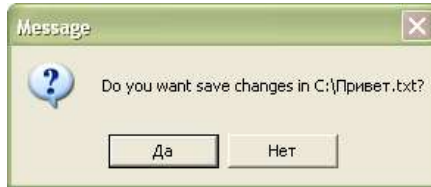


Рисунок 2.17 – Вікно попередження при закритті форми

2.2.8 OpenFileDialog і SaveFileDialog для SDI-додатків

При створенні MDI-додатків доводиться розділяти код для відкриття та збереження файлів, як ми робили для програми Notepad C #. У разі SDI-додатків весь код буде знаходитися в одному обробнику. Створити новий додаток, називаємо його TextEditor. На формі розміщуємо елемент управління TextBox і встановлюємо такі властивості. Приклади управління TextBox наведено у табл. 2.3.

Таблиця 2.3 – управління TextBox

TextBox, властивість	Значення
Name	txtBox
Dock	Fill
Multiline	true
Text	Да

Додаємо на форму елемент MainMenu, в якому буде всього три пункти – File, Open і Save (властивості цих пунктів див. у таблиці 2.2). З вікна ToolBox перетаскиваєм елементи OpenFileDialog і SaveFileDialog – властивості цих елементів в точності такі ж, як і у

діалогів програми Notepad С #. Переходимо в код форми. Підключаємо простір імен для роботи з файловими потоками:

```
using System.IO;
```

Додаємо обробник для пункту меню Open:

```
private void mnuOpen_Click(object sender, System.EventArgs e)
{
    openFileDialog1.ShowDialog();
    string fileName = openFileDialog1.FileName;
    FileStream filestream = File.Open(fileName, FileMode.Open, FileAccess.Read);
    if(filestream != null)
    {
        StreamReader streamreader = new StreamReader(filestream);
        txtBox.Text = streamreader.ReadToEnd();
        filestream.Close();
    }
}
```

Приклад 2.5.

Додаємо обробник для пункту меню Save:

```
private void mnuSave_Click(object sender, System.EventArgs e)
{
    saveFileDialog1.ShowDialog();
    string fileName=saveFileDialog1.FileName;
    FileStream filestream = File.Open(fileName, FileMode.Create, FileAc-
cess.Write);
    if(filestream != null)
    {
        StreamWriter streamwriter = new StreamWriter(filestream);
        streamwriter.Write(txtBox.Text);
        streamwriter.Flush();
        filestream.Close();
    }
}
```

Приклад 2.6.

Запускаємо програму. Як вже зазначалося, елемент управління TextBox має вбудоване контекстне меню, підтримує клавіші для редагування, а діалогові вікна відкриття і збереження дозволяють цьому найпростіший додатком працювати із зовнішніми файлами.

На диску, прикладеному до книги, ви знайдете додаток TextEditor (Code \ Glava2 \ TextEditor).

2.2.9 FontDialog

Продовжимо роботу над додатком Notepad C #. Додамо тепер можливість вибирати шрифт, його розмір і накреслення. У режимі дизайну перетягнемо на форму frmmain з вікна ToolBox елемент управління FontDialog. Не змінюючи нічого в властивості цього елемента, переходимо в обробник пункту Font головного меню:

```
private void mnuFont_Click(object sender, System.EventArgs e)
{
    blank frm = (blank)this.ActiveMdiChild;
    frm.MdiParent = this;
    fontDialog1.ShowColor = true;
    fontDialog1.Font = frm.richTextBox1.SelectionFont;
    fontDialog1.Color = frm.richTextBox1.SelectionColor;
    if (fontDialog1.ShowDialog() == DialogResult.OK)
    {
        frm.richTextBox1.SelectionFont = fontDialog1.Font;
        frm.richTextBox1.SelectionColor = fontDialog1.Color;
    }
    frm.Show();
}
```

Приклад 2.7.

Запускаємо програму, і у вікні Output з'являється список помилок (рис. 2.18).

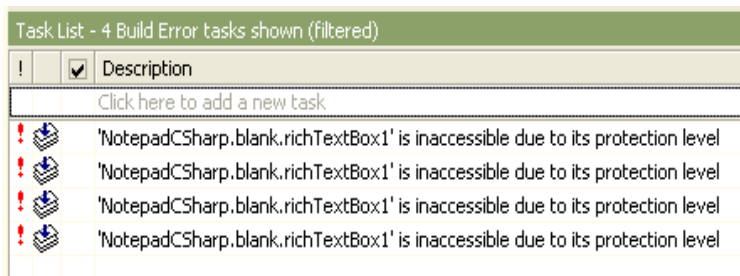


Рисунок 2.18 – Список помилок – NotepadCSharp.blank.richTextBox1 недоступний з-за свого рівня захисту

У чому ж справа? При розташуванні на формі blank елемента управління RichTextBox середу Visual Studio.NET згенерувала примірник richTextBox1 класу System.Windows.Forms.RichTextBox з модифікатором доступу private, через що при зверненні до нього і виникає виняток:

```
private System.Windows.Forms.RichTextBox richTextBox1;
```

Змінимо модифікатор на public і знову запустимо програму. При виборі пункту меню Font тепер можна змінювати параметри поточного тексту.

2.2.10 ColorDialog

Діалогове вікно FontDialog містить список кольорів, які можуть бути застосовані до тексту, але пропонуванй список обмежений. Більш цікавою видається можливість призначати користувальницький колір, який може бути визначений у великому діапазоні (рис. 2.19):

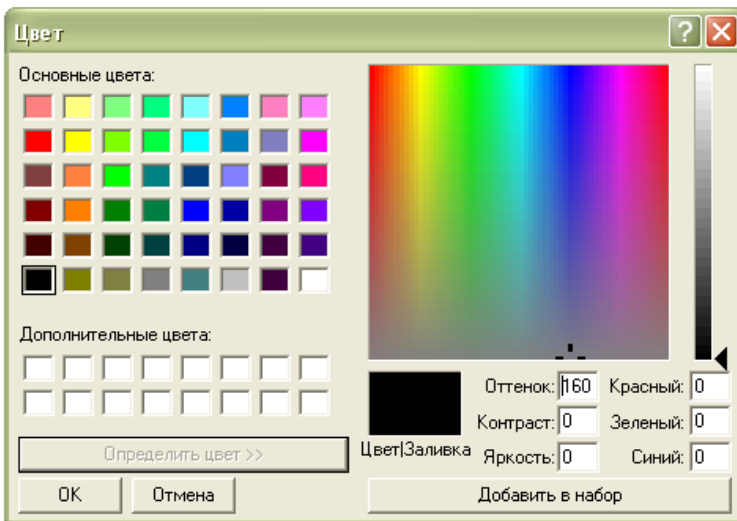


Рисунок 2.19 – Діалогове вікно ColorDialog

З вікна ToolBox додаємо елемент управління ColorDialog і, знову не змінюючи його властивостей, переходимо до обробникові пункту Color головного меню форми frmmain:

```
private void mnuColor_Click(object sender, System.EventArgs e)
{
    blank frm = (blank)this.ActiveMdiChild;
    frm.MdiParent = this;
    colorDialog1.Color = frm.richTextBox1.SelectionColor;

    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        frm.richTextBox1.SelectionColor = colorDialog1.Color;
    }

    frm.Show();
}
```

Зверніть увагу на те, що код для ColorDialog в точності такий же, як і частина коду для властивості Color діалогу FontDialog. Це й не дивно: адже ми пов'язуємо ці діалоги з властивостями одного і того ж об'єкта – RichTextBox.

2.2.11 StatusBar

Елемент управління StatusBar застосовується в програмах для виведення інформації в рядок стану – невелику смужку, розташовану внизу програми. У Microsoft Word, наприклад, у ньому відображається кількість сторінок, мова введення, стан перевірки правопису та інші параметри. Додамо до додатка Notepad С # рядок стану, на якій здійснюється підрахунок символів, що вводять і виводиться системний час.

Додаємо на форму blank елемент управління StatusBar. Видаляємо вміст поля властивості Text. У полі властивості Panels клацаємо на кнопку (...). Відкривається StatusBarCollectionEditor, в якому ми створюємо панелі для відображення. Створіть дві панелі, двічі клацаючи на кнопці Add, і встановіть їм наступні властивості (змінені значення виділяються жирним шрифтом) (рис. 2.20 та рис. 2.21):

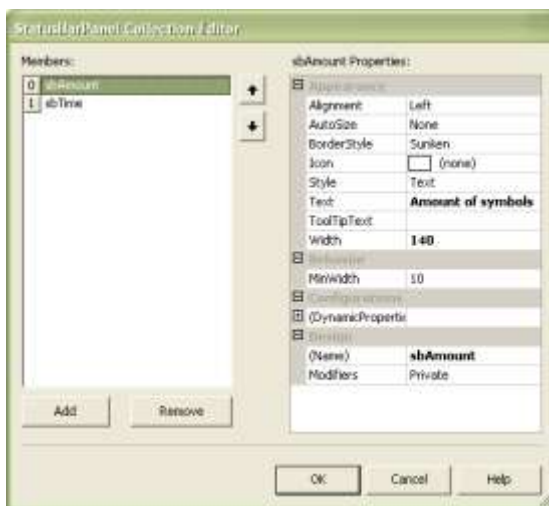


Рисунок 2.20 – Властивості панелі sbAmount

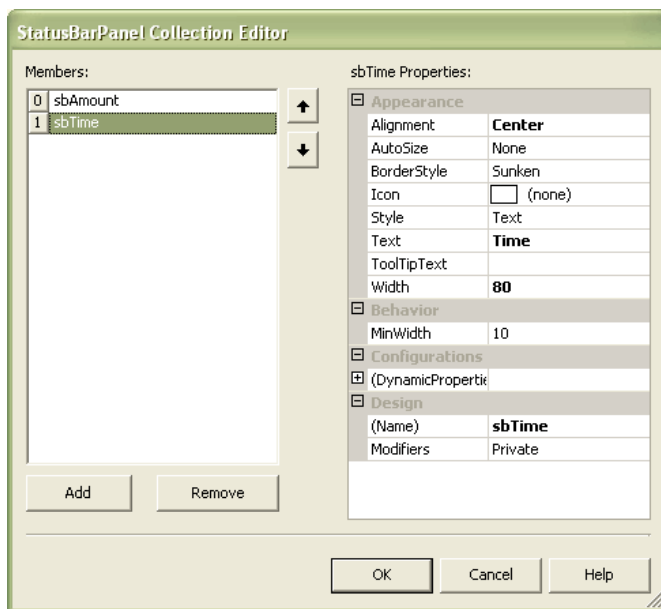


Рисунок 2.21 – Властивості панелі sbTime

Значення деяких властивостей панелі наводяться у табл. 2.4.

Таблиця 2.4 – Властивості панелі

Властивість	Значення
Align ment	Вирівнювання вмісту властивості Text на панелі
AutoSi ze	Зміна розмірів панелі по вмісту
Border Style	Зовнішній вигляд панелі – утоплена, піднесена або без виділення
Icon	Додавання іконки
Style	Стиль панелі
Text	Текст, що розташовується на панелі
ToolTi pText	Спливаюча підказка – з'являється при наведенні курсору на панель
Width	Ширина панелі в пікселях
Name	Назва панелі для звернення до неї в коді

Властивості панелі, які призначаються у вікні редактора StatusBarCollectionEditor, можна змінювати в коді – саме так ми зараз і зробимо.

Після завершення роботи над панелями закриваємо редактор. Властивості ShowPanels елемента керування StatusBar встановлюємо значення True. На формі негайно відображаються дві панелі. Виділяємо елемент управління RichTextBox, у вікні його властивостей перемикаємося на події і створюємо обробник для події TextChanged:

```
private void richTextBox1_TextChanged(object sender, System.EventArgs e)
{
    sbAmount.Text = "Amount of symbols" + richTextBox1.Text.Length.ToString();
}
```

Властивість Text Панелі sbAmount ми змінюємо програмно: навіть якщо б ми нічого не написали у вікні редактора StatusBarCollectionEditor, при вінікненні події TextChanged на панелі з'явиться напис.

Займемося тепер другою панеллю – тієї, на яку будемо виводити системний час. У конструкторі форми blank додаємо код:

```
public blank()
{
    InitializeComponent();
    sbTime.Text = Convert.ToString(System.DateTime.Now.ToLongTimeString());
    sbTime.ToolTipText = Convert.ToString(System.DateTime.Today.ToLongDateString());
}
```

Запускаємо програму. Панель із виведенням часом розташовується досить незвично (рис. 2.22), при необхідності для виведення часу в звичному правому нижньому куті можна додати третю порожню панель.

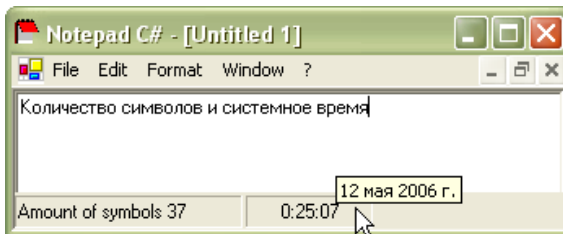


Рисунок 2.22 – Рядок стану з двома панелями

2.2.12 Закривання форми

При запуску програми Notepad 3 # у формі є три стандартні кнопки - "Згорнути", "Розгорнути" і "Закрити". Більшість користувачів вважають за краще використовувати саме кнопку "Закрити" для виходу з програми. Проте прийнято дублювати кнопку форми пунктом меню Exit. У обробнику цієї кнопки додаємо код

```
private void mnuExit_Click(object sender, System.EventArgs e)
{
    this.Close();
}
```

Метод Close закриває форму і може бути призначений інших елементів керування – наприклад, кнопки.

2.2.13 CheckBox

Чекбокси відносяться до так званих кнопок відкладеної дії, тобто їх натискання не запускає негайно будь-якого процесу. З їх допомогою користувач встановлює певні параметри, результат дії яких позначиться після запуску інших елементів управління. Додамо в додаток Notepad C # форму для пошуку заданого тексту, в якій будуть використані елементи управління CheckBox. У вікні Solution Explorer клацаємо правою кнопкою миші на назві проекту – NotepadCSharp – і в меню вибираємо пункт Add Windows Form. Називаємо нову форму FindForm.cs і встановлюємо такі властивості. Приклади нової форми наведено у табл. 2.5.

Таблиця 2.5 – Нова форма

FindForm, форма, властивість	Значення
Name	FindForm
FormBorderStyle	FixedTool Window
Size	328; 112
Text	Find

Маємо на формі TextBox, дві кнопки і два елементи CheckBox так, як показано на рис. 2.23:

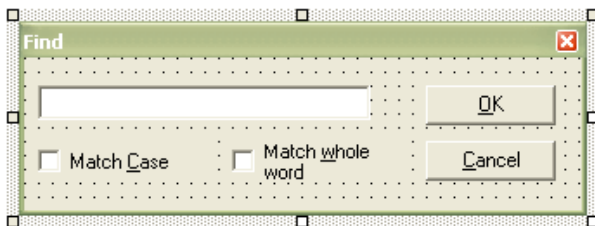


Рисунок 2.23 – Форма FindForm в режимі дизайну

Встановлюємо наступні властивості елементів керування у табл. 2.6 – 2.10 :

Таблиця 2.6 – Форма FindForm, TextBox

TextBox, властивість	Значення
Name	txtFind
Size	192; 20
Text	

Таблиця 2.7 – Форма FindForm, checkBox1

checkBox1, властивість	Значення
Name	cbMatchCase
Text	Match &Case

Таблиця 2.8 – Форма FindForm, checkBox2

checkBox2, властивість	Значення
Name	cbMatchWhole
Text	Match &whole word

Таблиця 2.9 – Форма FindForm, button1

button1, властивість	Значення
Name	btnOK
DialogResult	OK
Text	&OK

Таблиця 2.10 – Форма FindForm, button2

button2, властивість	Значення
Name	btnCancel
DialogResult	Cancel
Text	&Cancel

Вибір першого чекбокса – `cbMatchCase` – буде встановлювати пошук слова з урахуванням регістру, другого – `cbMatchWhole` – пошук по цілому слову. Зверніть увагу на властивість кнопок `DialogResult` – вибираючи відповідні властивості, ми встановлюємо обробники для кнопок без звернення до коду. Значення `OK` закриває форму, виконуючи встановлене дію, – в даному випадку – пошук; значення `Cancel` просто закриває форму. У головному меню форми `frmmain` додаємо розділову лінію і пункт `Find` і встановлюємо значення властивостей: `Name` – `mnuFind`, `Shortcut` – `CtrlF`, `Text` – `& Find`. Переходимо в обробник цього пункту:

```
private void mnuFind_Click(object sender, System.EventArgs e)
{
    FindForm frm = new FindForm();
    if(frm.ShowDialog(this) == DialogResult.Cancel) return;
    blank form = (blank)this.ActiveMdiChild;
    form.MdiParent = this;
    int start = form.richTextBox1.SelectionStart;
    form.richTextBox1.Find(frm.FindText, start, frm.FindCondition);
}
```

Основою пошуку буде метод `Find` елемента управління `RichTextBox`. Нам залишається лише вказати параметри пошуку, які можуть приймати різні значення (рис. 2.24):

```
int start = form.richTextBox1.SelectionStart;
//Вызываем предопределенный метод Find элемента richTextBox1.
form.richTextBox1.Find()
```

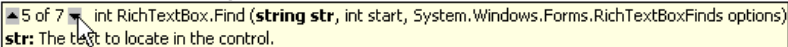


Рисунок 2.24 – Параметри методу `Find`. Клацаючи на маленьку кнопку підказки `Intel Sense`, обираємо різні підказки для груп переда-них параметрів

У кодї форми `FindForm` залишилося реалізувати логіку роботи, що залежить від положення елементів `CheckBox`:

```
public RichTextBoxFinds FindCondition
{
    get
```

```

    {
        if (cbMatchCase.Checked && cbMatchWhole.Checked)
        {
            return RichTextBoxFinds.MatchCase | RichTextBoxFinds.WholeWord;
        }
        if (cbMatchCase.Checked)
        {
            return RichTextBoxFinds.MatchCase;
        }
        if (cbMatchWhole.Checked)
        {
            return RichTextBoxFinds.WholeWord;
        }
        return RichTextBoxFinds.None;
    }
}

```

І нарешті, створюємо властивість FindText, повертає в якості змінної пошуку введений текст у текстове поле форми FindForm:

```

public string FindText
{
    get{return txtFind.Text;}
    set{txtFind.Text = value;}
}

```

Запускаємо програму. Вибираючи різні варіанти, можна одноразово шукати задане слово (рис. 2.25):

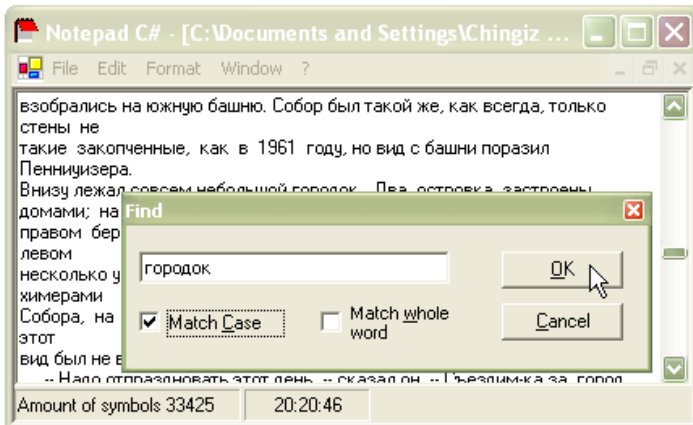


Рисунок 2.25 – Пошук слова в тексті

2.2.14 Властивість TabIndex елементів управління

Багато користувачів вважають за краще працювати по більшій частині з клавіатурою і вдаватися до допомоги миші тільки там, де без неї не можна обійтися. При роботі з формою Find було б зручно ввести слово в текстове поле, потім, натиснувши клавішу Tab і Enter, здійснити пошук, – або, двічі натиснувши клавішу Tab, переключити фокус введення на чекбокс Match Case. Для цього в режимі дизайну форми Find слід встановити властивість TabIndex елементів керування у вигляді послідовної нумерації, починаючи з нуля. Приклади властивості TabIndex наведено у табл. 2.7.

Таблиця 2.7 – Властивість TabIndex

Елемент управління	TabIndex
TxtFind	0
BtnOK	1
CbMatchCase	2
CbMatchWhole	3
BtnCancel	4

При такій нумерації фокус введення буде послідовно переходити в порядку зростання – від елемента txtFind до btnCancel. Властивість TabIndex має особливо велике значення при створенні ряду текстових полів, де користувачеві буде пропонуватися ввести інформацію.

Label, LinkLabel і PictureBox

Програми, як правило, містять пункт головного меню "Про програму", де в окремому вікні міститься логотип компанії, ліцензійна угода, гіперпосилання на сайт розробника і інша інформація. Створимо подібну форму, використовуючи нові елементи управління – Label, LinkLabel і PictureBox.

Отже, додамо в наш проект нову форму і назвемо її About.cs. Встановимо наступні властивості форми:

Таблиця 2.8 – Нова форма About.cs

About, форма, властивість	Значення
Name	About
FormBorderStyle	FixedSingle
MaximizeBox	False
MinimizeBox	False
Size	318; 214
Text	About Notepad C#

Додамо на форму елемент управління PictureBox – він представляє собою підкладку, яка розміщується на формі, яка може містити в собі малюнки для відображення. У полі властивості Image клацаємо на кнопку (...) і вибираємо Рисунок за адресою Code \ Glava2 \ NotepadCSharp \ Icon \ logo.gif. Оскільки logo.gif є анімованим малюнком, елемент PictureBox починає відтворювати анімацію відразу ж, навіть у режимі дизайну.

З вікна ToolBox перетягнемо на форму кнопку, Label і LinkLabel. У полі властивості Text кнопки введемо & OK. Елемент Label призначений для розміщення на формі написів, які в готовому додатку будуть доступні тільки для читання. У полі властивості Text введемо Notepad C # 2006 All rights reserved. Елемент LinkLabel відображає текст на формі в стилі web-посилань і зазвичай використовується для створення навігації між формами або посилання на сайт. У полі Text цього елемента вводимо адресу гіпотетичного сайту – www.notepadcsharp.com. Користувач буде переходити на сайт, натиснувши на це посилання, тому реалізуємо перехід за гіперпосиланням для події Click. У вікні Properties клацаємо двічі на події Click і додаємо обробник:

```
private void linkLabel1_Click(object sender, System.EventArgs e)
{
    try
```

```

        {
            VisitLink();
        }
        catch (Exception ex )
        {
            MessageBox.Show(ex +"Unable to open link that was clicked.");
        }

    }

    private void VisitLink()
    {
        linkLabel1.LinkVisited = true;
        System.Diagnostics.Process.Start("http://www.notepadsharp.com");
    }

```

Кнопка ОК просто буде закривати форму:

```

private void button1_Click(object sender, System.EventArgs e)
{
    this.Close();
}

```

У пункті головного меню About Programm ... форми frmmain додаємо процедуру виклику форми About:

```

private void mnuAbout_Click(object sender, System.EventArgs e)
{
    About frm = new About();
    frm.Show();
}

```

Запускаємо додаток (рис. 2.26):

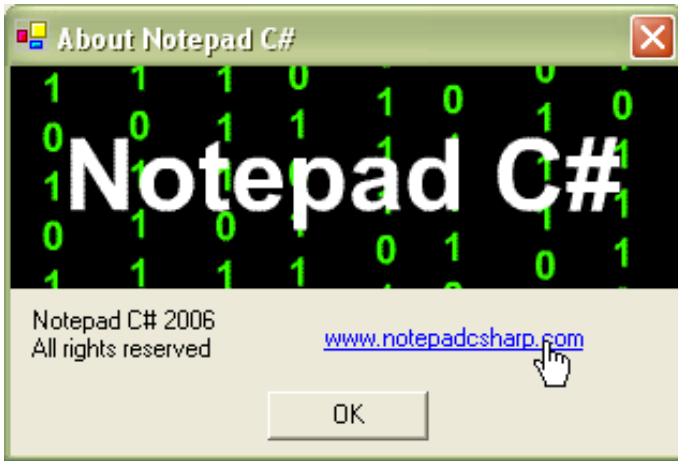


Рисунок 2.26 – Вікно About Notepad C #

2.3 Завдання до роботи

2.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

2.3.2 Виконати наступні завдання:

Реалізувати програму «Notepad C #». Передбачити завантаження, збереження, та створення, редагування текстів. Програма повинна дозволяти перегляд текстових файлів у формі програми, надавати змогу обирати папку для переглядання та відповідний перегляд усіх файлів у папці.

2.3.4 Оформити звіт з роботи.

2.3.5 Відповісти на контрольні питання.

2.4 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Завдання до роботи.

2.4.3 Короткі теоретичні відомості.

2.4.4 Текст розробленого програмного забезпечення.

2.4.5 Результати виконання роботи.

2.4.6 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

2.5 Контрольні запитання

- 2.5.1 Що таке елементи керування у C#?
- 2.5.2 Що таке група перемикачів?
- 2.5.3 Які є графічні елементи у C#?
- 2.5.4 Що таке MDI?
- 2.5.5 Що таке група контейнерів?
- 2.5.6 Властивість ActiveMdiChild у C#?
- 2.5.7 Застосовування StatusBar?
- 2.5.8 Метод LayoutMdi?
- 2.5.9 Що таке контекстне меню?
- 2.5.10 Що таке діалогові вікна?
- 2.5.11 Що таке OpenFileDialog?
- 2.5.12 Що таке SaveFileDialog?
- 2.5.13 Що таке FontDialog?
- 2.5.14 Властивість TabIndex елементів управління?
- 2.5.15 Призначення .NET Framework?
- 2.5.16 Інструменти .NET Framework?
- 2.5.17 Бібліотека класів .NET Framework перерахувати
- 2.5.18 Які є способи використання бібліотек.
- 2.5.19 Що таке доступ до даних.
- 2.5.20 Що таке ClassView?
- 2.5.21 Що таке вікно властивостей?
- 2.5.22 Що таке панель компонентів?
- 2.5.23 Клас Console?
- 2.5.24 Клас MessageBox?
- 2.5.25 Клас Random?
- 2.5.26 Типы данных C#
- 2.5.27 Що таке події C #?
- 2.5.28 Що таке атрибути C #?
- 2.5.29 Основні властивості Form
- 2.5.30 Що таке потік?

3 САМОСТІЙНА РОБОТА №3

РОБОТА З МЕНЮ ТА ОБРОБКА ВИНЯТКІВ У C#

3.1 Мета роботи

3.1.1 Навчитися створювати головне та контекстне меню.

3.1.2 Навчитися обробляти та генерувати винятки.

3.2 Основні теоретичні відомості

3.2.1 Меню та його пункти

Меню – це простий та звичний для користувачів інтерфейс, що відкриває доступ до команд та функцій верхнього рівня. Рационально сконструйоване меню впорядковує функціональність додатку, що робить його легким для вивчення та використання. У протилежному випадку, до невдало сконструйованому меню будуть звертатися тільки у випадку крайньої необхідності.

3.2.1.1 Контекстне меню

Контекстне меню зручно для доступу до набору команд, який залежить від поточного контексту програми. Контекстне меню створюють засобами компоненту `ContextMenu`. Для того щоб прив'язати його до певного об'єкта необхідно виділити об'єкт, при натисканні на який правою кнопкою буде з'являтися контекстне меню —> у властивості `ContextMenuStrip` вибрати потрібне контекстне меню.

Працюючи з меню можна перехопити змінну яка містить назву вибраного пункту, її синтаксис:

`MenuItem item = (MenuItem) sender;` // Змінний `item` присвоюємо об'єкт `sender` від якого прийшло повідомлення. Оскільки відомо, що повідомлення завжди посилаються від пунктів меню, виконуємо приведення до `MenuItem`.

`string text = item.Text;` // Об'єкти класу `MenuItem` мають властивість `Text`, яке в нашому випадку характеризує пункт меню.

3.2.1.2 Відмічені пункти меню

Якщо взяти до уваги стандартну програму MS Windows — «Калькулятор» та заглянемо до вкладки «Вигляд», то бачимо що там є відмічені пункти меню. Для того щоб відмітити пункти меню нам потрібно вибрати у властивостях пункт `Checked` та поставити йому значення `True`, у пункті `CheckState` вибираємо вигляд відмітки: `Indeterminate` – вигляд крапки, `Checked` – вигляд прапорця, `Unchecked` – не показувати відмітку.

Для того щоб при виборі пункту меню пункт відмічався чи переставав бути відміченим виберіть потрібні пункти меню—> змінити властивість `CheckOnClick` поставити в `true`.

3.2.1.3 Поділ елементів меню

При необхідності елементи меню відділяють один від одного горизонтальною лінією-роздільником. Розділювачі дозволяють упорядковувати меню, яке складається з безлічі елементів, розбивши його на логічні групи.

Щоб додати до меню роздільник, введіть у полі меню дефіс замість імені — під час виконання він перетвориться на роздільник.

3.2.1.4 Клавiші доступу

Натискаючи одночасно `Alt` і букву, призначену клавiшею доступу, ви швидко відкриє меню. Скориставшись клавiшею доступу, можна також вибрати команду з відкритого меню. Наприклад, у багатьох програмах `Alt+F` відкриває меню `File`. Визначити, яка буква призначена в якості клавiші доступу, просто: потрібна буква підкреслена в назвах елементів меню.

Дозволяється призначати однакові клавiші для доступу до елементів меню, якщо останні розташовані в різних групах. Наприклад, поєднання `Alt+C` приміняють для доступу до команд `Close` з групи `File` і `Copy` з групи `Edit`. Однак не варто призначати однакові клавiші доступу до елементів меню з однієї групи, наприклад `Alt C` для команд `Cut` і `Сміття` з групи `Edit`. Якщо знехтувати цієї рекомендацією, клавiша доступу буде по черзі активувати елементи меню, для яких вона

призначена, але вибрати який-небудь з них вдасться лише натисненням клавіші Enter.

3.2.1.5 Гарячі клавіші

Гарячу клавішу встановлюють значенням з перерахування Shortcut – це перерахування 150 клавіатурних сполучень, що рекомендовані для меню.

Для звертання до пунктів меню за допомогою гарячих клавіш потрібно у властивостях знайти пункт ShortKeys та вибрати потрібне сполучення клавіш.

У Win32 – програмуванні гарячі клавіші можна використовувати не тільки для швидкого виклику пунктів меню. Але в програмі Windows Forms гарячі клавіші служать тільки для виклику пунктів меню. Гарячу клавішу, що не зв'язану з пунктом меню, можна визначити, зв'язавши спочатку пункт меню з цією клавішею, а потім включивши цей пункт у своє меню та встановивши властивість меню Visible в false. Пункт не буде виводитися, але швидка клавіша буде працювати.

3.2.1.6 Активація та деактивація команд меню

У кожного елемента меню є властивість Enabled. Встановивши його в false, ви деактивуєте елемент меню, забороняючи йому реагувати на дії користувача, у тому числі на клавіші доступу і швидкі клавіші. Деактивовані елементи меню виділені блідо-сірим кольором.

3.2.1.7 Вставка зображень для пунктів меню

Виділіть потрібний пункт меню та виберіть властивість Image. Перед вами з'явиться вікно (рис.3.1):

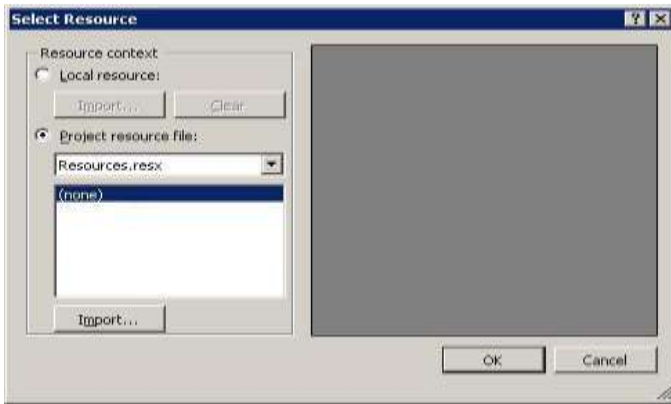


Рисунок 3.1 – Зображення вікна для завантаження зображень для пунктів меню

Вибираємо необхідний пункт:

Local resources – якщо вибрати цей пункт, то зображення не записуються у програму та будуть відкриватися з встановленого вами шляху. При перенесенні програми на інший комп’ютера вам необхідно буде разом з програмою скопіювати зображення, які ви вибрали для цієї програми.

Project resource file – якщо вибрати цей пункт, то зображення зберігаються у програмі, і для перенесення програми на інший комп’ютер вам достатньо тільки скопіювати програму.

Далі натискаємо на **Import** та вибираємо потрібний файл.

Потім натискаємо на **OK**. Все вибране зображення прикрашає пункт вашого меню.

Для кожного пункту меню ви можете робити різне форматування тексту.

3.2.1.8 Створення меню програмно

Програмне створення меню, яке містить два пункти верхнього рівня меню “Файл” (з пунктами “Відкрити” та “Вихід”) та Допомога (з пунктом “Про програму”) і при натисканні на пункт з’являється повідомлення.

```
using System;
```

```

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Просте меню";

            // Головне меню
            MenuStrip menu = new MenuStrip();
            this.Controls.Add(menu);

            // Розділ File
            ToolStripMenuItem itemFile = new ToolStripMenuItem();//Створення
            змінної, яка буде відповідати за пункт меню Файл
            itemFile.Text = "&Файл"; // Назва пункту
            menu.Items.Add(itemFile);// Елемент верхнього рівня

            // Розділ Help
            ToolStripMenuItem itemHelp = new ToolStripMenuItem();
            itemHelp.Text = "&Допомога";
            menu.Items.Add(itemHelp);

            // Заповнення розділу File
            // Команда Open...
            ToolStripMenuItem itemOpen = new ToolStripMenuItem();
            itemOpen.Text = "&Відкрити...";
            itemOpen.ShortcutKeys = Keys.Control | Keys.O; // Ctrl+O
            itemOpen.ShowShortcutKeys = true;// Відображення в команді
            itemOpen.Click += OpenOnClick;
            itemFile.DropDownItems.Add(itemOpen);
            // itemOpen.ShowShortcutKeys = false;// Сховати комбінацію

            // Розділювач
            ToolStripSeparator itemSep = new ToolStripSeparator();
            itemFile.DropDownItems.Add(itemSep);

            // Команда Exit
            ToolStripMenuItem itemExit = new ToolStripMenuItem();
            itemExit.Text = "&Вихід";

```

```

itemExit.Click += ExitOnClick;
itemFile.DropDownItems.Add(itemExit);

// Заповнення розділу Help
// Команда About...
ToolStripMenuItem itemAbout = new ToolStripMenuItem();
itemAbout.Text = "&Про програму...";
itemAbout.ShortcutKeys = Keys.Control | Keys.A; // Ctrl+A
itemAbout.ShortcutKeyDisplayString = "Ctrl^A";// Створення "гарячих"
клавiш для цього меню
itemAbout.ShowShortcutKeys = true;// Відображувати в команді
itemAbout.Click += AboutOnClick;
itemHelp.DropDownItems.Add(itemAbout);
}

// Обробник подій
void OpenOnClick(object sender, EventArgs e)
{
    MessageBox.Show("Заготiвля опції \"Вiдкрити\"", this.Text);
}

void ExitOnClick(object sender, EventArgs e)
{
    this.Close();
}

void AboutOnClick(object sender, EventArgs e)
{
    MessageBox.Show("Заготiвля опції \"Про програму...\"", this.Text);
}
}

```

3.2.2 Обробка та генерація винятків

Навіть тестування і налагодження програми не виключають можливість виникнення помилок у період виконання. Розглянемо для прикладу додаток, записуючий дані на гнучкий диск. Якщо воно спробує зберегти дані, коли диск не готовий, неодмінно станеться помилка, навіть якщо код програми сам по собі не містить помилок і працює, як задумано. Мова Visual C # підтримують структурну обробку винятків, яка дозволяє створювати код, здатний коректно обробляти помилки, і значить, продовжити виконання програми. Крім того, можна створювати об'єкти винятків та генерувати винятки всередині компонентів для повідомлення основної програми про виникнення помилок.

3.2.2.1 Принципи обробки винятків

Виявивши помилку періоду виконання, додаток генерує винятку (exception). Винятки становлять собою екземпляри спеціалізованих класів, нащадків класу `System.Exception`. Ці класи підтримують функції, облягаючи діагностику помилок та керування їх обробкою, наприклад:

- властивість `Message` — містить опис помилки в доступній для людини формі та інші важливі відомості про помилку;
- властивість `StackTrace` — зберігає дані трасування стека; забезпечує трасування для пошуку місця в коді, де виникла помилка.

На додаток до винятків, що визначаються системою, дозволяється створювати користувацькі винятки на основі класу `System.ApplicationException`. При виникненні помилки періоду виконання створюється екземпляр відповідаючого винятку і передається від методу, генерувати помилку, вгору по стеку викликів. У результаті виняток отримує метод, з якого був викликаний той, що згенерував помилку. Якщо в цьому методі є структура, здатна обробити цей виняток, воно обробляється, в іншому випадку воно передається наступному методу в стеку викликів і т.д. Якщо структура, обробила виняток та не знайшла його в стеку виклику, то викликається обробник винятків, заданий за замовчуванням. Він відображає повідомлення із зазначенням типу винятку і будь-яких додаткових відомостей, наданих об'єктом винятку, і закриває додаток, не дозволяючи зберегти результати роботи чи виправити помилку.

Застосування структурної обробки винятків дозволяє додатку усунути наслідки непередбачених помилок або, як мінімум, зберегти дані перед закриттям.

Створення обробників винятків

Обробники винятків реалізують для окремих методів. У кожного методу повинен бути власний обробник винятків, орієнтований на особливості даного методу і пристосований для обробки винятків, які він зазвичай генерує. Обробку винятків неодмінно слід реалізувати в методах, які часто генерують винятку, наприклад використовують файловий доступ. Для обробників винятків прийнятий синтаксис `try ... catch ... finally`.

try–блок інкапсулює код, що формує частину нормальних дій програми, та в якому потенційно може відбутися серйозні помилкові ситуації.

catch–блок інкапсулює код, який оброблює помилкові ситуації, що відбуваються в коді блока try.

finally–блок інкапсулює код, що очищає будь-які ресурси чи виконує інші дії, які необхідно виконати в кінці блоків try чи catch.

Ось основні етапи створення обробника винятків:

- помістіть код, який здатний генерувати винятки, до блоку try;
- додайте один або кілька блоків catch, що обробляють різні типи винятків, які може генерувати код у блоці try;
- помістіть в блок finally код, який необхідно виконати в будь-якому випадку. Цей код буде виконаний незалежно від того, чи виник виняток.

Вигляд:

```
try
{
// Тут знаходиться звичайний код програми
}
catch
{
// Обробка помилок
}
finally
{
// Очищення
}
```

Оператор catch також дозволяє отримати посилання на об'єкт генерованого винятку. Це посилання корисне, тому що в обробнику помилок через неї вдається отримати доступ до об'єкта винятку. Крім того, вона дозволяє перехоплювати певні типи помилок і створювати кілька блоків catch, призначених для обробки певних помилок. Так, наприклад, можна перехопити виняток певного типу:

У блок finally поміщають код, який необхідно виконати незалежно від того, виник виняток чи ні. Це може бути код, який зберігає данні, який звільняє ресурси або код, виконання якого є критичним для програмування. Ось приклад блоку try. .. catch .. finally:

Нехай у нас є програма для ділення 1/x без операторів винятку вона буде виглядати:

```
static void Main()
{
int x DE4A
```

```
int y = 1 / x;
Console.WriteLine(y);
}
```

Ця програма нормально функціонує при умові $x > 0$. Якщо застосувати оператори винятку, то програма отримає вигляд:

```
static void Main()
{
    try
    {
        int x = int.Parse(Console.ReadLine());
        int y = 1 / x;
        Console.WriteLine("y={0}", y);
        Console.WriteLine("блок try виконано вдало ");
    }
    catch
    {
        Console.WriteLine("виникла помилка");
    }
    Console.WriteLine("кінець програми");
}
```

Деякі класи обробки винятків наведено у табл. 3.1.

Таблиця 3.1 – Деякі класи обробки винятків

Назва	Опис
ArithmeticException	Помилка в арифметичних операціях чи перетвореннях
ArrayTypeMismatchException	Спроба збереження в масиві елемента несумісного типу
DivideByZeroException	Спроба ділення на нуль
FormatException	Спроба передати в метод аргумент невірною діапазону
IndexOutOfRangeException	Індекс масиву виходить за границю діапазону
InvalidCastException	Помилка перетворення типу
OutOfMemoryException	Недостатньо пам'яті для нового об'єкту
OverflowException	Переповнення при виконанні арифметичних операцій
StackOverflowException	Переповнення стеку

Для того щоб створити обробник винятку необхідно:

- додати в блок `try` код, здатний генерувати помилки, які потрібно обробити;
- якщо виняток повинен оброблятися локально, створіть один або декілька блоків `catch` з відповідним кодом;
- код, який необхідно виконати незалежно від виникнення винятку помістіть в блок `finally`.

Клас Exception

Розглянемо пристрій базового класу `Exception`, щоб зрозуміти, яку інформацію може отримати обробник винятку, коли йому передається об'єкт, що задає поточний виняток.

Основними властивостями класу є:

- `Message` — рядок, що задає причину виникнення винятку. Значення цієї властивості встановлюється при виклику конструктора класу, коли створюється об'єкт, що задає виняток;
- `HelpLink` — посилання (URL) на файл, що містить докладну довідку про можливу причину виникнення виняткової ситуації та способи її усунення;
- `InnerException` — посилання на внутрішній виняток. Коли обробник викидає новий виняток для передачі обробки на наступний рівень, то поточний виняток стає внутрішнім для новостворюваного винятку;
- `Source` — ім'я програми, що став причиною винятку;
- `StackTrace` — ланцюжок викликів — методи, що зберігаються в стеку викликів у момент виникнення винятку;
- `TargetSite` — метод, який викинув виняток.

З методів класу відзначимо метод `GetBaseException`. При підйомі по ланцюжку викликів він дозволяє отримати вихідний виняток — першопричину виникнення послідовності викиду винятків.

Клас має чотири конструктора, з яких три вже згадувалися. Один з них — конструктор без аргументів, другий — сприймає рядок, що стає властивістю `Message`, третій — має ще один аргумент: виняток, передані властивості `InnerException`.

У цьому випадку у створюваного винятку заповнюється властивість `InnerExceptions`. Для стеження за властивостями винятків я додав

метод друку всіх властивостей, що викликається у всіх обробниках винятків:

```
static public void PrintProperties (Exception e)
{
    Console.WriteLine ("Властивості винятку:");
    Console.WriteLine ("TargetSite = {0}", e.TargetSite);
    Console.WriteLine ("Source = {0}", e.Source);
    Console.WriteLine ("Message = {0}", e.Message);
    if (e.InnerException == null)
        Console.WriteLine ("InnerException = null");
    else Console.WriteLine ("InnerException = {0}",
        e.InnerException.Message);
    Console.WriteLine ("StackTrace = {0}", e.StackTrace);
    Console.WriteLine ("GetBaseException = {0}",
        e.GetBaseException ());
}
```

На закінчення теми винятків хочу ще раз підкреслити, що коректне застосування механізму винятку повинно підтримуватися цілеспрямованими зусиллями програміста. Слід пам'ятати про дві важливі правила:

- обробка винятків повинна бути спрямована не стільки на повідомлення про виникнення помилки, скільки на коригування ситуації, що виникла;
- якщо виправити ситуацію не вдається, то програма повинна бути перервана так, щоб не були отримані некоректні результати, не задовольняючі специфікаціям програми.

3.2.2.2 Генерація винятків

Дуже часто потрібно генерувати виняток. У загальному випадку їх можна розділити на два типи:

- при виникненні помилки періоду виконання, яку не вдається обробити повністю. Після часткової обробки такої помилки слід генерувати виняток, щоб передати її по стеку для подальшої обробки;
- при роботі компонента виникла неприпустима ситуація, яку не вдається дозволити локально. Про це необхідно повідомити батьківський компонент, згенерувавши стандартний виняток .NET або нестандартний виняток, специфічний для даного компонента.

Обидва даних докладно розглядаються в цьому розділі.

3.2.2.3 Повторна генерація винятків

Іноді структурі обробки винятків не вдається повністю обробити створений виняток. Наприклад, під час роботи методу, що виконує безліч складних обчислень, іноді несподівано виникає виняток `NullReferenceException`. У цьому випадку успішна обробка винятку можлива, вірогідний також результат, коли це не вдається. Ваш код повинен перевірити, чи в змозі він обробити, те що виникло. Якщо це не вдається в контексті блоку `catch`, слід генерувати його повторно за допомогою ключового слова `throw`, наприклад:

```
try
{
    // Тут повинен бути код додатку,
}
catch (System.NullReferenceException e)
{
    // А сюди поміщують код, який перевіряє виняток і поточний контекст,
    // Щоб визначити, чи можна обробити виняток в даному блоці catch,
    // Передано вище по стеку. Якщо воно не може бути оброблено локально.
    // Локально, наступний рядок генерує виняток повторно і передає
    // Його далі по стеку.
    throw e;
}
```

При використанні універсального блоку `catch`, не надає посилання на об'єкт винятку, отримане цим блоком винятку можна генерувати повторно за допомогою ключового слова `throw`, вказавши його без аргументів.

У деяких випадках необхідно передати на наступний рівень програми додаткові відомості. Для цього замість повторної генерації виняткові слід генерувати інший виняток, в цьому випадку другий виняток грає роль оболонки для першого. Посилання на об'єкт вихідного винятку записується у властивість `InnerException` об'єкта нового винятку його конструктором. Цей прийом дозволяє передати з системним типом винятку додаткових даних, записавши їх у відповідні властивості нового винятку і помістивши вихідний виняток у властивість `InnerException`. Нижче демонструється створення і генерація нового винятку, що надає рядок з докладним описом вихідного винятку, оболонкою якої вона є:

```
// Це початок блоку Catch.
catch (NullReferenceException e)
```

```
{
throw new NullReferenceException ("якийсь текст", e);
}
```

Як повторно генерувати виняток:

- щоб повторно генерувати виняток, отримане блоком Catch (catch), скористуйтеся ключовим словом Throw (throw);
- щоб генерувати новий виняток, що надає додаткові зведення про вихідний виняток, створіть об'єкт винятку, що є оболонкою для вихідного винятку, і генеруйте новий виняток за допомогою ключового слова throw.

3.2.2.4 Нестандартний виняток

При розробці компонентів іноді також потрібно генерувати виняток. Якщо під час роботи компонента виникає неприпустима ситуація, слід генерувати виняток для клієнтського додатку. Для цих цілей завжди використовують нестандартний виняток, оскільки загальною виконуюча середа група центрів генерують виняток лише в зумовлених виняткових ситуаціях.

Примітка. Згідно назви, винятки призначені лише для не стандартизованих ситуацій. Не слід використовувати їх для взаємодії між клієнтом і компонентами — для цього призначені події. Виняток слід генерувати, тільки коли продовження нормального виконання програми без зовнішнього втручання неможливо.

Нестандартний виняток, можна створювати на основі класу System.ApplicationException. Це клас інкапсулює всю необхідну функціональність, у тому числі властивості Message, StackTrace і InnerException, і може стати основою для нестандартних винятків, орієнтованих на потреби вашого компонента.

Нижче показаний приклад нестандартного винятку, створеного на основі класу Application Exception. Конструктор нового винятку вимагає фіктивний об'єкт Widget, оскільки новий виняток надає доступ до цього об'єкта через властивість, доступне лише для читання:

```
public class WidgetException: System.ApplicationException
{
// Ця змінна зберігає об'єкт Widget.
Widget mWidget;
public Widget ErrorWidget
{
get
```

```

{
return mWidget;
}
}
// Цей конструктор приймає об'єкт Widget і рядок String, ці
// Параметри можна використовувати для опису контексту програми
// На момент помилки. Дозволяється також створювати перевантажені
версії
// Конструктора, що підтримують різні набори параметрів.
//
// Наступний рядок викликає конструктор з базового класу
// І встановлює властивість Message,
// Успадковане від класу ApplicationException.
public WidgetException (Widget W, string S): base (S)
{
// Встановити властивість Widget.
mWidget = W;
}
}

```

Оголосивши клас винятку, вдається згенерувати новий екземпляр цього класу при виникненні неприпустимій ситуації, наприклад, так:

```

Widget Alpha;
// Тут повинен бути код, що ушкоджує об'єкт Widget з ім'ям Alpha.
throw new WidgetException (Alpha, "Widget Alpha is corrupt!");

```

Як генерувати нестандартний виняток:

— оголосіть клас нестандартного винятку. Він повинен бути нащадком `System.ApplicationException` і може містити будь-які властивості, надають додаткові дані клієнтам, що обробляють виняток. Ці властивості ініціалізують в конструкторі класу нестандартного винятку;

— при виникненні нештатної ситуації генеруйте новий екземпляр класу нестандартного винятку за допомогою ключового слова `throw`, передавши конструктору відповідні параметри.

3.2.2.5 Генерація власних винятків

Цей розділ ми роздивимось на прикладах:

Щоб генерувати власні винятки необхідно скористатися оператором `throw`, вказавши параметри, що визначають вид винятку. Параметром має бути об'єкт, породжений від стандартного класу `System.Exception`. Цей об'єкт використовується для передачі інформації про виняток обробника.

```
static void Main ()
{
    try
    {
        int x = int.Parse (Console.ReadLine ());
        if (x <0) throw new Exception (); // 1
        Console.WriteLine ("ok");
    }
    catch
    {
        Console.WriteLine ("Введено некоректне значення");
    }
}
```

У рядку 1 за допомогою команди `new` був створений об'єкт винятку типу `Exception`. При необхідності можна генерувати виняток будь-якого типу.

При генерації винятків можна визначити повідомлення, яке буде "викидатися" обробником винятків. Наприклад:

```
static void Main ()
{
    try
    {
        int x = int.Parse (Console.ReadLine ());
        if (x <0) throw new Exception ("введено неприпустиме значення"); //
1    Console.WriteLine ("ok");
    }
    catch (Exception error)
    {
        Console.WriteLine (error.Message);
    }
}
```

3.2.2.6 Корисні поради генерації винятків

Розглянемо кілька корисних прийомів використання обробників винятків.

Приклад 1. Один `try`-блок можна вкласти в інший. Виняток, згенерований у внутрішньому `try`-блоці і не перехоплений `catch`-інструкцією, яка пов'язана з цим `try`-блоком, передається в зовнішній `try`-блок. Наприклад, у наступній програмі виняток типу `ArithmeticException` перехоплюється не внутрішнім `try`-блоком, а зовнішнім.

```
static void Main ()
{
    Console.WriteLine ("a =");
```

```

byte a = byte.Parse (Console.ReadLine ());
Console.WriteLine ("b =");
byte b = byte.Parse (Console.ReadLine ());
int f = 1;
try / / Зовнішній блок-try
{
    for (byte i = a; i <= b; + + i)
    {
        try / / Внутрішній блок-try
        {
            f = checked ((int) (f * i));
            Console.WriteLine ("y ({0}) = {1: f6}", i, 100 / (f - 1));
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine ("y ({0}) = Ділення на 0", i);
        }
    }
}
catch (ArithmeticException)
{
    Console.WriteLine ("ERROR");
}

```

Використання вкладених try-блоків обумовлено бажанням обробляти різні категорії помилок різними способами. Одні типи помилок носять катастрофічний характер і не підлягають виправленню. Інші — безпечні для подальшого функціонування програми, і з ними можна справитися прямо на місці їх виникнення. Тому зовнішній try-блок можна використовувати для перехоплення самих серйозних помилок, дозволяючи внутрішнім try-блокам обробляти менш небезпечні.

Приклад 2. Виняток, перехоплений однією catch-інструкцією, можна згенерувати повторно, щоб забезпечити можливість його перехоплення інший (зовнішньої) catch-інструкцією. Це дозволяє декільком обробникам отримати доступ до винятку.

```

static void genException ()
{
    Console.WriteLine ("a =");
    double a = double.Parse (Console.ReadLine ());
    Console.WriteLine ("b =");
    double b = double.Parse (Console.ReadLine ());
    int f = 1;
    try / / Зовнішній блок-try
    {
        for (double i = a; i <= b; + + i)
        {
            try / / Внутрішній блок-try
            {
                f = checked ((int) (f * i));

```

```

        Console.WriteLine ("y ({0}) = {1: f6}", i, 100 / (f - 1));
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine ("y ({0}) = Ділення на 0", i);
    }
}
catch (ArithmeticException)
{
    Console.WriteLine ("ERROR");
    throw; // повторна генерація винятку
}
}

static void Main ()
{
    try
    {
        genException ();
    }
    catch
    {
        Console.WriteLine ("фатальна помилка !!!");
    }
}

```

Потрібно пам'ятати, що при повторному генеруванні винятку воно не буде повторно перехоплюватися тієї ж catch-інструкцією, а передається наступній (зовнішній) catch-інструкції.

Приклад 3. Як згадувалося вище, тип винятку повинен збігатися з типом, заданим в catch-інструкції. В іншому випадку цей виняток не буде перехоплено. Можна перехоплювати всі винятки, використовуючи catch-інструкцію без параметрів. Крім того, з try-блоком можна зв'язати не одну, а кілька catch-інструкцій. У цьому випадку всі catch-інструкції повинні перехоплювати виняток різного типу. Якщо ви все ж не впевнені, що передбачили всі ситуації, то останньої можна додати catch-інструкцію без параметрів.

Зауваження. Іноді виникає потреба в обов'язковому виконанні якихось дій, які повинні виконатися щодо виходу з try/catch-блоку. Наприклад, генерується виняток і відбувається передчасне завершення виконання програмного фрагмента, але при цьому залишається відкритим файл. Для виходу з такої ситуації C # надає блок finally, який додається після всіх блоків catch.

```

static void Main ()
{
    for (int i = 0; i < 5; i++)

```

```

{
    try
    {
        Console.WriteLine ("Введіть два числа");
        int a = int.Parse (Console.ReadLine ());
        int b = int.Parse (Console.ReadLine ());
        Console.WriteLine (a + "/" + b + "=" + a / b);
    }
    catch (FormatException)
    {
        Console.WriteLine ("Потрібно ввести число!");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine ("Ділити на нуль не можна!");
    }
    catch
    {
        Console.WriteLine ("Якась помилка");
    }
    finally
    {
        Console.WriteLine ("після try-блоку");
    }
}
}

```

3.3 Завдання

3.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

3.3.2 Створити інженерний калькулятор, у ньому реалізувати головне, контекстне меню та обробку винятків.

3.3.3 Оформити звіт.

3.3.4 Відповісти на контрольні питання.

3.4 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.2 Завдання до роботи.

3.4.3 Короткі теоретичні відомості.

3.4.4 Код програми.

3.4.5 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

3.5 Контрольні питання

- 3.5.1 Що таке меню головне?
- 3.5.2 Яке головне призначення головного меню?
- 3.5.3 Як створити головне меню?
- 3.5.4 Що таке контекстне меню?
- 3.5.5 Яке головне призначення контекстного меню?
- 3.5.6 Як зробити пункти меню відміченими?
- 3.5.7 Як присвоїти пунктам меню «гарячі» клавіші?
- 3.5.8 Що робить символ & в назві головного меню? Як зробити поділ елементів меню?
- 3.5.9 Як активувати та деактивувати пункти меню?
- 3.5.10 Як присвоїти пункту меню зображення?
- 3.5.11 Які існують операції для створення меню програмно?
- 3.5.12 Що таке виняток?
- 3.5.13 Як створити обробник винятку? Поясніть призначення кожного блоку винятку?
- 3.5.14 Який принцип обробки винятків?
- 3.5.15 Назвіть п'ять класів обробки винятків.
- 3.5.16 Клас Exception. Основні властивості класу?
- 3.5.17 Навіщо потрібна повторна генерація винятків?
- 3.5.18 Як зробити повторну генерацію винятків?
- 3.5.19 Як генерувати нестандартний винятки?
- 3.5.20 Навіщо використовувати вкладені try-блоки?

4 САМОСТІЙНА РОБОТА № 4

РОБОТА З ДРУКОМ ТА СТВОРЕННЯ ЕЛЕМЕНТІВ КЕРУВАННЯ У C#

4.1 Мета роботи

4.1.1 Навчитися реалізовувати друк

4.1.2 Навчитися створювати елементи керування.

4.2 Основні теоретичні відомості

4.2.1 Друк

У будь-якій організації, незалежно від роду її діяльності, часто потрібно друкувати дані та документи. Для цього, в .NET Framework передбачений компонент PrintDocument, що забезпечує виведення на друк і надає ряд класів для підтримки друку і установки пов'язаних з нею параметрів.

Без сумніву, можливість друку є однією з найважливіших умов для ділових додатків. Незважаючи на те, що все більше людей зберігають файли і документи в електронному вигляді, навряд чи в найближчому майбутньому вони повністю відмовляться від друкованих матеріалів. Тому розробникам необхідно вміти створювати підтримку друку в своїх додатках. .NET Framework надає класи і компоненти, які допоможуть вам просто і швидко доповнити свою програму цієї функцією.

4.2.1.1 Компонент PrintDocument

У .NET Framework друкований документ представлений компонентом PrintDocument.

Хоча в PrintDocument немає графічного інтерфейсу, цей компонент знаходиться на вкладці Windows Forms інструментальної панелі Toolbox, звідки його можна перетягти на форму додатку, відкриту у вікні дизайнера. Об'єкт PrintDocument інкапсулює всю інформацію, необхідну для друку сторінок. У нього є ряд властивостей: PrinterSettings зберігає відомості про можливості і конфігурації досту-

пних принтерів, Default PageSettings інкапсулює параметри друку сторінок, а PrintController контролює обробку сторінок при друці.

Ці властивості дозволяють вельми дієво керувати процесом друку.

Створення об'єкта PrintDocument

Об'єкт PrintDocument можна створити в період розробки, за допомогою графічного інтерфейсу дизайнера, або програмно, у період виконання. У першому випадку об'єкт PrintDocument слід перетягнути у вікно дизайнера прямо з панелі Toolbox у області компонентів з'явиться новий екземпляр цього об'єкта, автоматично конфігурований для роботи з системним принтером за замовчуванням. Ось як програмно створюється екземпляр об'єкта PrintDocument:

```
PrintDocument myPrintDocument = new PrintDocument();
```

Отриманий таким чином примірник об'єкта PrintDocument автоматично конфігурується для роботи з системним принтером за замовчуванням.

Підтримка друку

У моделі друку, прийнятої в .NET Framework, що друкують дані безпосередній, але передаються кодом програми. Виклик методу PrintDocument, Print ініціює нове завдання друку, попутно генеруючи події PrintPage. Якщо клієнтський обробник для цієї події відсутній, друк не виконується. Щоб задати дані для друку, необхідно написати обробник для події PrintPage.

Якщо завдання друку складається з декількох сторінок, подія PrintPage генерується для кожної з них, тому обробник цієї події викликається багаторазово. Оброблювач події PrintPage контролює виконання завдання друку, забезпечуючи послідовний висновок багатосторінкового документа; без його участі друкувалися б лише кілька копій першої сторінки документа.

Подія PrintPage

Подія PrintPage грає ключову роль в процесі друку. Щоб реально відправити дані на принтер, необхідно створити обробник для події PrintPage і помістити в нього код, що промальовує друкований вміст. Усі дані та об'єкти, необхідні для друку, передаються обробнику в об'єкті PrintPageEventArgs, властивості якого описані в табл. 4.1.

Таблиця 4.1 – Властивості об'єкта PrintPageEventArgs

Ім'я	Опис
Cancel	Вказує, чи скасовано завдання друку
Graphics	Містить об'єкт Graphics, що формує друковану сторінку
HasMorePages	Отримує або встановлює значення, яке вказує, чи всі сторінки надруковані
MarginBounds	Отримує об'єкт Rectangle, що представляє область сторінки, обмежену полями
PageBounds	Отримує об'єкт Rectangle, що представляє сторінку цілком
PageSettings	Отримує або встановлює об'єкт PageSettings для поточної сторінки

Для перетворення вмісту документа в дані для принтера застосовується об'єкт Graphics, доступний через клас PrintPageEventArgs. У цьому випадку друкуема сторінка відіграє ту ж роль, що і клієнтська область форми, елемента керування і будь-який інший поверхні для малювання, представлена об'єктом Graphics. Друковану сторінку формують тими ж методами, що застосовуються для промальовування вмісту форми на екрані. Ось приклад дуже простого методу, який друкує еліпс, вписаний в область, обмежену полями:

```
//Щоб цей метод був викликаний, він повинен оброблятися подією
//PrintPage.
public void PrintEllipse (object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
    e.Graphics.DrawEllipse (Pens.Black, e.MarginBounds);
}
```

Властивості MarginBounds і PageBounds вказують області сторінки, доступні для друку. Можна кордонами області друку зробити поля сторінки, обчисливши координати друкованих об'єктів на основі прямокутника MarginBounds. Припустимо друкувати і за межами сторінкових полів — наприклад, щоб створити верхній і нижній колонтитули. Для цього слід розраховувати координати об'єктів на основі прямокутника PageBounds. Як і при виведенні на дисплей, при друкуванні, координати за замовчуванням обчислюються в пікселях.

Властивість `HasMorePages` дозволяє вказати, чи є завдання друку багатосторінковим. За замовчуванням цю властивість встановлено у `false`. Якщо ж буде потрібно надрукувати багатосторінковий документ, слід встановити його в `true`, а по завершенні друку останньої сторінки — знову в `false`.

Примітка. Обробник події `PrintPage` повинен стежити за кількістю сторінок заданим для друкування, в іншому випадку результати будуть непередбачуваними. Скажімо, якщо після закінчення виведення останньої сторінки не встановити властивість `HasMorePages` в `false`, додаток буде генерувати події `PrintPage` знову і знову.

Крім того, передбачена можливість скасувати завдання друку, не чекаючи закінчення виведення поточної сторінки. Для цього слід встановити властивість `Cancel` в `true`.

Щоб створити обробник події `PrintPage`, двічі клацніть примірник `PrintDocument` у вікні дизайнера (так створюється обробник за замовчуванням) або оголосити обробник вручну.

4.2.1.2 Вивід на друк

Щоб створити завдання друку, досить викликати метод `PrintDocument`, `Print`. У наступному розділі ви дізнаєтеся, як написати код, який визначає тип вмісту, призначеного для виведення на друк.

Друк графіки

Друкувати графічні елементи не складніше, ніж малювати їх на екрані. Для обох цілей застосовують об'єкт `Graphics`, доступний через об'єкт `PrintPageEventArgs`. Друкують як прості фігури, так і більш складні, у останньому випадку використовують об'єкт `GraphicsPath`. Ось як надрукувати складну фігуру за допомогою об'єкта `Graphics Path`:

```
// / Цей метод повинна обробляти подія PrintPage.
public void PrintGraphics (object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
    System.Drawing.Drawing2D.GraphicsPath myPath = new System.Drawing.Drawing2D.GraphicsPath ();
    myPath.AddPolygon (new Point [] {new Point (1, 1), new Point (12, 55), new Point (34, 8), new Point (52, 53), new Point (99, 5)});
}
```

```
myPath.AddRectangle (new Rectangle (33, 43, 20, 20));
e.Graphics.DrawPath (Pens.Black, myPath);
}
```

Щоб надрукувати багатосторінковий графічний документ, треба вручну розбити його на сторінки і написати відповідний код. Наступний метод розіб'є при друкуванні еліпс, що не вміщається на одній сторінці, на дві сторінки:

```
bool FirstPagePrinted = false;
// Цей метод повинна обробити подія PrintPage
public void PrintBigEllipse (object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
if (FirstPagePrinted == false)
{
FirstPagePrinted = true;
e.HasMorePages = true;
e.Graphics.DrawEllipse (Pens.Black, new Rectangle (0,0,
e.PageBounds.Width, e.PageBounds.Height * 2));
else
e.HasMorePages = false;
FirstPagePrinted = false;
e.Graphics.DrawEllipse (Pens.Black, new Rectangle (0,
- (E.PageBounds.Height), e.PageBounds.Width,
e.PageBounds.Height * 2));
}
```

У цьому прикладі виведення сторінок на друк реалізовано за допомогою змінної `FirstPagePrinted`. Зауважте: ця змінна оголошена поза методу. Якщо оголосити її усередині методу, вона ініціалізується при кожному його виклику і завжди повертає `false`. Насправді під час друку кожної сторінки програма перемальовує еліпс заново, змінюючи його положення так, щоб на поточну сторінку потрапила відповідна частина фігури.

Друк тексту

Вивід на друк тексту відбувається так само, як на екран, — за допомогою методу `Graphics.DrawString`. Як і при виведенні на екран, необхідно вказати шрифт, сам текст, об'єкт `Brush` і координати початку друку, наприклад:

```
Font myfont = new Font ("Batang", 36, FontStyle.Regular, GraphicsUnit.Pixel);
string Hello="Hello World!";
e.Graphics.DrawString (Hello, myfont, Brushes.Black, 30, 30);
```

Примітка. Перед друком тексту слід програмно перевірити, чи не виходить якась його частина за межі сторінки: будь-який вміст, який виходить за межі сторінки, надруковано не буде.

Друк багаторядкового тексту

Для друку багаторядкового тексту, наприклад масиву рядків або вмісту текстів файлу, необхідний код, який обчислює відстань між окремими рядками. Щоб розрахувати кількість рядків на сторінці, розділити висоту клієнтської області сторінки (вона дорівнює висоті сторінки за вирахуванням полів) на кегль шрифту. Положення будь-якого рядка на сторінці можна обчислити, помноживши її номер на кегль шрифту. Наступний приклад демонструє печатку строкового масиву з іменем `myStrings`:

```
// Ця змінна відстежує поточний елемент масиву.
// Щоб уникнути ініціалізації цієї змінної при друці кожної
// Сторінки, слід оголошувати поза обробника події PrintPage.
int ArrayCounter = 0;
// Це обробник події PrintDocument.PrintPage. Він припускає,
// Що рядковий масив з ім'ям myStrings () вже оголошений і заповне-
ний,
// А також що ініціалізований об'єкт myFont, що представляє
// Обраний для друку шрифт.
private void PrintStrings (object sender, PrintPageEventArgs e)
{
    // Оголосити змінні, керуючи міжстроковими
    // Інтервалами документа на сторінки.
    float LeftMargin = e.MarginBounds.Left;
    float TopMargin = e.MarginBounds.Top;
    float MyLines = 0;
    float YPosition = 0;
    int Counter = 0;
    string CurrentLine;
    // Підрахувати кількість рядків на сторінці.
    MyLines = e.MarginBounds.Height / myFont.GetHeight (e.Graphics);
    // Роздрукувати всі рядки файлу, зупиняючись в кінці сторінок.
    while (Counter < MyLines && ArrayCounter <= myStrings.GetUpperBound
        ())
    {
        CurrentLine = myStrings [ArrayCounter];
        YPosition = TopMargin + Counter * myFont.GetHeight (e.Graphics);
        e.Graphics.DrawString (CurrentLine, myFont, Brushes.Black,
            LeftMargin, YPosition, new StringFormat());
        Counter ++;
        ArrayCounter ++;
        // Якщо надруковані ще не всі рядки, то друкувати наступну сторінку,
        if (!(ArrayCounter == myStrings.GetUpperBound (0)))
            e.HasMorePages = true;
        else
            e.HasMorePages = false;
    }
}
```

Робота з кольорами

Набір параметрів для друку на принтері, що підтримує кольори, відрізняється від того, що потрібно для друку на чорно-білому принтері. При кольоровому друкуванні необхідно врахувати, що деякі кольори, які добре помітні на екрані, практично не помітні на папері. Таким чином, код для друку на чорно-білих і кольорових принтерах не однаковий.

Щоб визначити, чи підтримує поточний принтер кольоровий друк, перевірте властивість `PrinterSettings.SupportsColor`. Якщо це так, властивість `DefaultPageSettings.Color` встановлено в `true`, а принтер буде налаштований на кольоровий друк за замовчуванням. Для чорно-білого друку встановіть властивість `DefaultPageSettings.Color` в `false`.

Наступний приклад демонструє організацію підтримки кольорового та чорно-білого друку за допомогою властивості

```
PrinterSettings.SupportsColor.
// Цей приклад припускає наявність операторів Imports System. Drawing,
// System. Drawing. Drawing2D і System. Drawing. Printing
Brush BrushOne;
Brush BrushTwo;
if (PrintDocument1. PrinterSettings.SupportsColor == true)
{
    // Створити пензлі для кольорового друку.
    BrushOne = Brushes. Red;
    BrushTwo = Brushes. Blue;
}
else
{
    // Створити об'єкти HatchedBrush для чорно-білого друку.
    BrushOne = new HatchBrush(HatchStyle. DarkVertical, Color. Black);
    BrushTwo = new HatchBrush (HatchStyle. DashedHorizontal, Color. Black);
}
```

Організація підтримки друку в папці:

- перетягніть об'єкт `PrintDocument` з панелі `Toolbox` у вікно дизайнера — в область компонентів з'явиться новий екземпляр цього компонента, автоматично налаштований для використання принтера за замовчуванням. Створити компонент `PrintDocument` можна і програмно;

- напишіть обробник для події `PrintDocument PrintPage`;

- додайте обробникові події `PrintPage` код, який малює вміст в клієнтській області сторінки. Передайте сформоване зображення на принтер за допомогою об'єкту `PrintPageEventArgs.Graphics`;

– якщо ви працюєте з багатосторінковими документами, напишіть код, що керує розбиттям на сторінки.

4.2.1.3 Застосування елемента керування PrintPreviewControl

У .NET Framework є елемент керування `PrintPreviewControl`, що забезпечує можливість попереднього перегляду документа перед відправкою його на друк. `PrintPreviewControl` знаходиться на вкладці `Windows Forms` інструментальної панелі `Toolbox`, звідки його можна перетягнути на форму.

Щоб налаштувати попередній перегляд сторінки, потрібно пов'язати елемент керування `PrintPreviewControl` з примірником `PrintDocument`, встановивши властивість `PrintPreviewControl.Document` наступним чином:

```
myPrintPreview.Document = myPrintDocument;
```

Після цього елемент керування `PrintPreviewControl` буде відображати вміст, призначений для друку. Викличте обробник події `PrintDocument.Print`, щоб перехопити формований їм графічний образ сторінки і відобразити його в елементі керування `PrintPreviewControl`.

При зміні параметрів програми часто змінюється і вигляд друкованого документа.

Щоб оновити образ сторінки, що відображається при попередньому перегляді, викличте метод `InvalidatePreview`:

```
myPrintPreview.InvalidatePreview();
```

`Zoom` — ще одна важлива властивість елемента керування `PrintPreviewControl`. Воно вказує масштаб сторінки при попередньому перегляді. Якщо воно дорівнює 1, масштаб дорівнює 100%. При значеннях до 1 зображення для попереднього перегляду менше оригіналу, а при значеннях, що перевищують 1, зображення відповідно більше.

Елемент керування PrintPreviewDialog

Елемент керування `PrintPreviewControl` істотно полегшує створення підтримки попереднього перегляду документів, що друкуються, у тому числі додавання нестандартних режимів перегляду. Однак більшості додатків цілком достатньо стандартних функцій попереднього перегляду при друці. Для цих цілей у .NET Framework передбаче-

ний стандартний елемент керування PrintPreviewDialog, підтримує найпоширеніші можливості попереднього перегляду при друку. PrintPreviewDialog відображає діалогове вікно, показане на рис. 4.1.

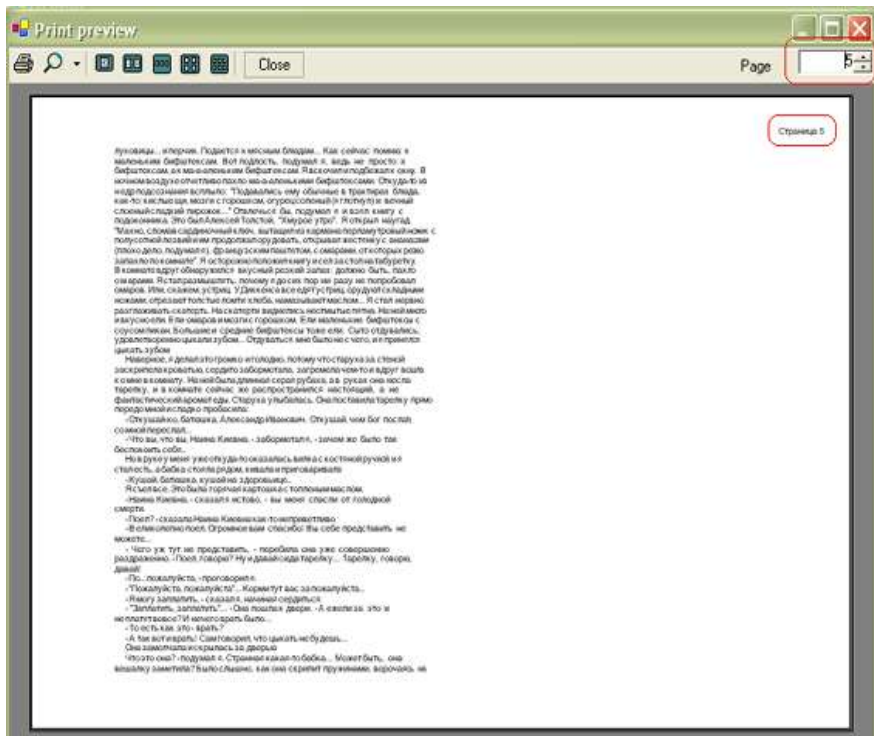


Рисунок 4.1 – Вікно PrintPreviewDialog

Для попереднього перегляду документа слід помістити у властивість Document об'єкта PrintPreviewDialog об'єкт Print Document, який представляє потрібний документ. Елемент керування PrintPreviewDialog дозволяє переглядати кілька сторінок документа одночасно, змінювати масштаб і відправляти документ на друк. Виклик вікна PrintPreviewDialog не відрізняється від виклику будь-якого іншого вікна і здійснюється за допомогою методу Show або ShowDialog.

4.2.1.4 Налаштування параметрів друку

Інтерфейс друку .NET Framework підтримує безліч керуючих параметрів. Властивість `PrintDocument.PrinterSettings` містить відомості про принтери, доступних у системі, властивість `PrintDocument.DefaultPageSettings` — параметри сторінки за замовчуванням (вони застосовуються, якщо в обробнику події `PrintPage` не вказані інші параметри). У свою чергу, у цих властивостей є безліч власних властивостей. На щастя, призначення більшості з них зрозуміло без додаткових пояснень. Крім того, при створенні об'єкта `PrintDocument` у властивість `PrinterSettings` поміщається об'єкт принтера за замовчуванням, конфігурований параметрами за замовчуванням, а у властивість `DefaultPageSettings` записуються параметри сторінки за замовчуванням. Таким чином, для створення і виконання завдання друку можна обійтися параметрами за замовчуванням.

Якщо ви хочете дозволити користувачам самостійно конфігурувати завдання друку, скористайтеся елементами керування Visual Studio .NET, що забезпечує безкоштовну інтеграцію налаштування принтера з призначеним для користувача інтерфейсом додатки — `PrintDialog` і `PageSetupDialog`.

PrintDialog

Вікно `PrintDialog` дозволяє встановити властивість `PrinterSettings` об'єкта `PrintDocument` в період виконання. Щоб додати примірник `PrintDialog` до додатка, перетягніть його з вкладки Windows Forms панелі Toolbox у вікно конструктора. Щоб відобразити вікно `PrintDialog` в період виконання, викличте метод `ShowDialog` наступним чином:

```
PrintDialog1.ShowDialog();
```

Вікно `PrintDialog` необхідно пов'язати з об'єктом `PrintDocument`. Для цього помістіть у властивість `PrintDialog.Document` об'єкт `PrintDocument`, що представляє потрібний документ.

Під час відображення вікно `PrintDialog` (рис. 4.2) зв'язується з властивістю `PrinterSettings` об'єкта `PrintDocument`, заданого властивістю `Document`, і надаються користувачеві графічний інтерфейс для налаштування друку.



Рисунок 4.2 – Вікно PrintDialog

PageSetupDialog

Методики застосування вікон PageSetupDialog і PrintDialog подібні. Вікно PageSetupDialog (рис. 4.3) надає графічний інтерфейс для налаштування параметрів сторінки і друку в період виконання. Користувач може вибрати орієнтацію сторінки, розміри паперу і встановити поля, а також налаштувати принтер.

Як і PrintDialog, вікно PageSetupDialog підтримує властивість Document, яка містить об'єкт PrintDocument, що представляє потрібний документ. Встановивши властивість Document, можна зв'язати об'єкти PrintDocument і PageSetupDialog. При цьому будь-які зміни, зроблені у вікні налаштування друку, будуть відображатися у властивості об'єктів PrintDocument.PrinterSettings і DefaultPageSettings.

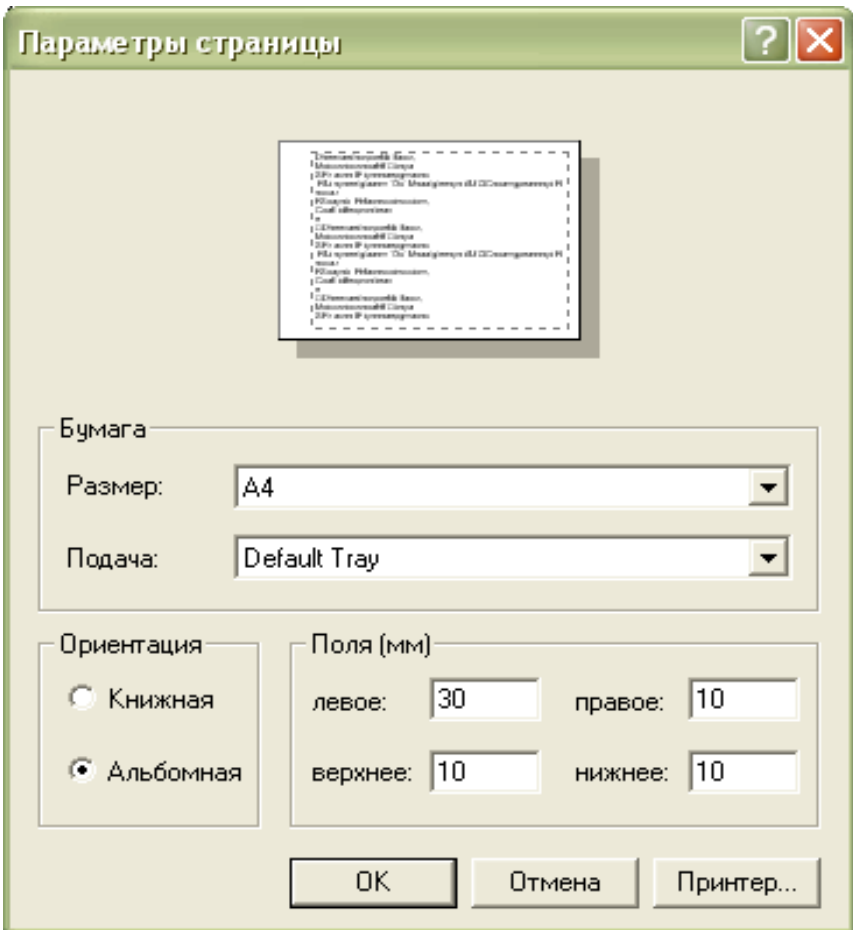


Рисунок 4.3 – Вікно PageSetup Dialog

Конфігурування PageSettings в період виконання

Іноді окремі сторінки багатосторінкового документа потрібно надрукувати не так, як інші, наприклад вивести одну сторінку в альбомній орієнтації, а всі інші — у книжковій. Змінити параметри окремої сторінки дозволяє властивість `PrintPageEventArgs.PageSettings`, яке надає параметри друку сторінки. Зміни параметрів друку, зроблені за допомогою цієї властивості, діють тільки на поточну сторінку, інші

сторінки завдання друкуються, як задано при відправленні документа на друк. Наступний приклад демонструє зміна макету в період виконання, встановлюючи альбомну орієнтацію сторінки:

```
// Цей фрагмент обробника подій PrintPage.  
// Змінна e представляє об'єкт PrintPageEventArgs.  
e.PageSettings.Landscape = true;
```

Конфігурування PageSettings в період виконання:

- документи, що друкуються представлені екземпляром класу PrintDocument. Цей клас підтримує властивості PrinterSettings, що задає параметри принтера, і DefaultPageSettings, що визначає параметри сторінки за замовчуванням;

- друк документа виконують шляхом виклику методу PrintDocument.Print. Виклик цього методу генерує подія PrintPage, в обробник якого поміщають код для промальовування документа;

- об'єкт PrintPageEventArgs містить усі дані і підтримує всі функції, необхідні для промальовування вмісту документа на принтері. Дані передаються на принтер в об'єкта Graphics, що представляє цей принтер; об'єкт Graphics передається через об'єкт PrintPageEventArgs;

- для друку багатосторінкових документів до додатку необхідно додати код, що виконує розбиття документа на сторінки. Для повторної генерації події PrintPage встановіть властивість PrintEventArgs.HasMorePages в true;

- елемент керування PrintPreviewControl забезпечує попередній перегляд документів перед відправкою на друк, а PrintPreviewDialog надає найбільш потребувані можливості PrintPreview в зручній для використання формі;

- вікна PrintDialog і PageSetupDialog дають користувачам можливість налаштовувати принтер і параметри сторінки в період виконання. Передбачена також можливість налаштування параметрів окремої сторінки – для цього достатньо змінити властивість PrintPageEventArgs.PageSettings під час друку.

4.2.1.5 Приклад друку вмісту різних компонентів

Розташуйте на формі компоненти RichTextBox та MenuStrip (табл. 4.2).

Таблиця 4.2 – Пункти головного меню

Name	Text	Shortcut
mnuFile	&Файл	
Name	Text	Shortcut
mnuOpen	&Відкрити	Ctrl+O
mnuSave	&Зберегти	Ctrl+S
menuItem1	-	
mnuPageSetup	Пара&метри сторінки	
mnuPrintPreview	Перед&редньої перегляд	
mnuPrint	&Друк	Ctrl+P

Перетягніть на форму з вікна ToolBox елементи керування: PrintDocument, PageSetupDialog, PrintPreviewDialog і PrintDialog. Подібно іншим елементам діалогів, вони відображаються на панелі компонентів в середовищі Visual Studio. NET. При друкуванні формується одна або кілька сторінок, за які відповідає об'єкт PrintDocument. Елементи керування PageSetupDialog, PrintPreviewDialog і PrintDialog представляють собою діалогові вікна параметрів сторінок, перегляду та друку відповідно. Налаштування властивостей цих об'єктів є настільки гнучким і широким завданням, вирішуване в коді програми, що, на відміну, наприклад, від елементів OpenFileDialog або SaveFileDialog, має сенс створювати їх програмно, в класі форми:

```
// PrintDocument printDocument1 = new PrintDocument ();
// PageSetupDialog pageSetupDialog1 = new PageSetupDialog ();
// PrintPreviewDialog printPreviewDialog1 = new PrintPreviewDialog
();
// PrintDialog printDialog1 = new PrintDialog ();
```

Цей фрагмент повністю аналогічний додаванню на форму елементів. Ми, проте, будемо використовувати візуальний режим.

Властивості Document елементів керування PageSetupDialog і PrintPreviewDialog встановлюємо значення властивості Name елемента PrintDocument – printDocument1. Тим самим ми пов'язуємо об'єкт printDocument1, що відповідає за формування сторінок преси, з діалоговими вікнами. Встановлюємо наступні властивості елемента PrintDialog (табл. 4.3).

Таблиця 4.3 – Властивості елемента PrintDialog

printDialog1, властивість	Значення	Опис
AllowSelection	True	Дозвіл на друк виділеного фрагмента документа
AllowSomePages	True	Дозвіл на друк кількох сторінок
PrintDocument	printDocument1	Зв'язування з примірником об'єкта PrintDocument

На інформаційній панелі вікна Properties при виборі властивостей AllowSelection або AllowSomePages можна помітити повідомлення — Enables and disables the Selection/Pages radio button (Включає або відключає доступність радіоперемикачів "Виділений фрагмент / Сторінки"). Саме по собі визначення цих властивостей ще не включає відповідну функціональність — ще одна причина, по якій конфігурувати об'єкти друку краще програмно:

```
InitializeComponent ();
// PageSetupDialog1.Document = printDocument1;
// PrintPreviewDialog1.Document = printDocument1;
// PrintDialog1.Document = printDocument1;
// PrintDialog1.AllowSomePages = true;
// PrintDialog1.AllowSelection = true;
```

Переходимо до коду форми. Для роботи з друком в бібліотеці .NET Framework застосовується клас System.Drawing.Printing, його потрібно підключити в самому початку роботи:

```
using System.Drawing.Printing;
```

У класі форми оголошуємо наступні змінні:

```
// Змінна для зберігання тексту для друку.
// У неї ми будемо розмішувати текст з RichTextBox
string stringPrintText;
// Змінна, яка визначає номер сторінки, з якої потрібно почати друк
int StartPage;
// Змінна, яка визначає кількість сторінок для друку:
int NumPages;
// Змінна, яка визначає номер поточної сторінки:
int PageNumber;
```

Слід звернути увагу на змінну stringPrintText — саме вона буде відповідати за передачу тексту в об'єкт друку. Оскільки ми визначили для неї рядковий тип, це означає, що ми не зможемо вивести на друк малюнки в тексті — як ми знаємо, елемент RichTextBox підтримує їх наявність (в цьому можна переконатися, скопіювавши вміст докумен-

та Microsoft Word у наш додаток). Визначення повного друку буде більш складним, що, втім, не є істотним для розуміння самої концепції друку).

У конструкторі формі визначаємо діапазон сторінок для друку:

```
public Form1 ()
{
    InitializeComponent ();
    ...
    // Визначаємо номер сторінки, з якої слід почати друк
    printDialog1.PrinterSettings.FromPage = 1;
    // Визначаємо максимальний номер друкованої сторінки.
    printDialog1.PrinterSettings.ToPage = printDia-
    log1.PrinterSettings.MaximumPage;
}
```

Максимальна кількість сторінок в діапазоні — 9999; очевидно, що його цілком вистачить для роздрукування будь-якого документа. Основною подією, в обробнику якого практично і буде здійснюватися весь друк, буде подія `PrintPage` елемента `printDocument1`. Перемикаємося в режим дизайну, виділяємо елемент і, переключившись у вікні `Properties` на вкладку подій, двічі клацаємо в поле цієї події:

```
private void printDocument1_PrintPage (object sender, Sys-
tem.Drawing.Printing.PrintPageEventArgs e)
{
    ...
}
```

Далі всі описувані фрагменти коду будуть ставитися у цей обробник. Клас `Graphics`, що належить простору імен `System.Drawing` — один з найважливіших класів, що відповідають за малювання графіки і виведення тексту на формі. При створенні нової форми простір імен `System.Drawing` за замовчуванням включається в шаблон. Для формування сторінки друку створюємо екземпляр цього класу:

```
Graphics graph = e.Graphics;
// Створюємо об'єкт font, якому встановлюємо
// Шрифт елемента rtbText
Font font = rtbText.Font;
// Отримуємо значення міжрядкового інтервалу - висоту шрифту
float HeightFont = font.GetHeight (graph);
// Створюємо екземпляр stringformat
//класу StringFormat для визначення
// Додаткових параметрів форматування тексту.
StringFormat stringformat = new StringFormat ();
```

Сторінка представляє собою прямокутний об'єкт, параметри якого слід визначити. Для цього використовується клас `RectangleF`

(літера F вказує на використання типу даних з плаваючою точкою float):

```
// Створюємо екземлярів rectanglefFull і rectfText класу RectangleF
для
// Визначення областей друку і тексту.
RectangleF rectanglefFull, rectfText;
// Створюємо змінні для підрахунку числа символів і рядків.
int NumberSymbols, NumberLines;
```

Перша область — повний розмір сторінки, зазвичай має назви, наприклад, формат A4 або A3. Інформацію про цю області надає властивість PageBounds класу PrintPageEventArgs. Його значенням є об'єкт Rectangle, властивості X і Y якого рівні 0, а властивості Width і Height надають розміри паперу за замовчуванням, виражені в сотих частках дюйма. Друга область пов'язана з конструктивними особливостями принтерів — більшість з них використовують для друку площу, меншу розмірів паперу. Область друку визначається значенням властивості VisibleClipBounds класу Graphics, яка надається об'єктом RectangleF. Властивості X і Y цього об'єкта мають значення 0, а її властивості Width і Height рівні горизонтального і вертикального розмірами області друку сторінки. У третій області міститься текст, вона зазвичай визначається користувачем. У Microsoft Word, наприклад, ми встановлюємо задані значення полів. За замовчуванням, ця область розраховується з відступом в один дюйм по периметру сторінки). Її розмір повертається об'єктом Rectangle властивості MarginBounds.

У будь-якому випадку, області формуються об'єктами Rectangle або RectangleF, конструктори яких можуть мати наступний вигляд:

```
Rectangle (int x, int y, int width, int height);
RectangleF (float x, float y, float width, float height);
```

Ці конструктори відрізняються тільки типом даних, які формують структуру. Координатні осі області, що представляє собою прямокутник.

Повернемося до нашого коду. Як область друку встановлюємо об'єкт rectanglefFull:

```
if (graph.VisibleClipBounds.X < 0) rectanglefFull = e.MarginBounds;
else
// Визначаємо об'єкт rectanglefFull
rectanglefFull = new RectangleF (
// Встановлюємо координату X
e.MarginBounds.Left - (e.PageBounds.Width -
graph.VisibleClipBounds.Width) / 2,
// Встановлюємо координату Y
e.MarginBounds.Top - (e.PageBounds.Height -
graph.VisibleClipBounds.Height) / 2,
```

```
// Встановлюємо ширину області
e.MarginBounds.Width,
// Встановлюємо висоту області
e.MarginBounds.Height);
Область тексту буде представляти собою копію області rectangleFull с
урахуванням висоти шрифту:
rectanglefText = RectangleF.Inflate (rectanglefFull, 0, -2 *
HeightFont);
// Визначаємо кількість рядків
int NumDisplayLines = (int) Math.Floor (rectanglefText.Height /
HeightFont);
// Встановлюємо висоту області
rectanglefText.Height = NumDisplayLines * HeightFont;
```

Елемент RichTextBox має властивість WordWrap, при встановленні значення True якого текст автоматично переноситься на новий рядок. Проте розбитий на рядки текст може вийти за межі визначеної для нього області rectanglefText. Щоб цього не сталося, ми створили примірник stringformat класу StringFormat, для якого тепер встановлюємо значення Trimming:

```
if (rtbText.WordWrap)
{
    stringformat.Trimming = StringTrimming.Word;
}
else
{
    stringformat.Trimming = StringTrimming.EllipsisCharacter;
    stringformat.FormatFlags |= StringFormatFlags.NoWrap;
}
```

Далі код достатньо простий, наводжу його з коментарями:

```
// При друку вибраних сторінок переходимо до першої стартовій сторін-
ці
while ((PageNumber < StartPage) && (stringPrintText.Length > 0))
{
    if (rtbText.WordWrap)
    // Вимірюємо текстові змінні,
    // Формуємо друк, і повертаємо число символів NumberSymbols
    // І кількість рядків NumberLines
    graph.MeasureString (stringPrintText, font, rectanglefText.Size,
    stringformat, out NumberSymbols, out NumberLines);
    else
    NumberSymbols = SymbolsInLines (stringPrintText, NumDisplayLines);
    stringPrintText = stringPrintText.Substring (NumberSymbols);
    // Збільшуємо кількість сторінок
    PageNumber ++;
}
// Якщо довжина рядка stringPrintText дорівнює нулю (немає тексту
// для друку),
// Зупиняємо друк
if (stringPrintText.Length == 0)
{
```

```

e.Cancel = true;
return;
}
// Виводимо (малюємо) текст для друку - stringPrintText,
//використовуємо для цього шрифт font,
// Кисть чорного кольору - Brushes.Black, область друку - rectanglef-
Text,
// Передаємо рядок додаткового форматування stringformat
graph.DrawString (stringPrintText, font, Brushes.Black, rectanglef-
Text, stringformat);
// Отримуємо текст для наступної сторінки
if (rtbText.WordWrap)
graph.MeasureString (stringPrintText, font, rectanglefText.Size,
stringformat, out NumberSymbols, out NumberLines);
else
NumberSymbols = SymbolsInLines (stringPrintText, NumDisplayLines);
stringPrintText = stringPrintText.Substring (NumberSymbols);
// Очищаємо об'єкт stringformat, використаний для формування полів.
stringformat = new StringFormat ();
// Додаємо висновки на кожен сторінку її номера
stringformat.Alignment = StringAlignment.Far;
graph.DrawString ("Сторінка" + PageNumber, font, Brushes.Black, rec-
tanglefFull, stringformat);
PageNumber ++;
// Снова перевіряємо, чи є текст для друку і номер сторінки, заданої
для друку
e.HasMorePages = (stringPrintText.Length > 0) && (PageNumber < Start-
Page + NumPages);
// Для друку з вікна попереднього перегляду знову ініціалізували
змінні
if (! e.HasMorePages)
{
stringPrintText = rtbText.Text;
StartPage = 1;
NumPages = printDialog1.PrinterSettings.MaximumPage;
PageNumber = 1;
}
}

```

Ми закінчили розгляд методу `printDocument1_Print-Page`. Досить істотним моментом тут є те, що в коді реалізована перевірка наявності розстановки переносів у текстовому полі `RichTextBox` (властивість `WordWrap`). Метод `MeasureString` повертає число символів, що виводяться. Для обліку символів, що не потрапляють в область `rectanglefText`, додаємо метод `SymbolsInLines`:

```

int SymbolsInLines (string stringPrintText, int NumLines)
{
int index = 0;
for (int i = 0; i < NumLines; i ++ )
{
index = 1 + stringPrintText.IndexOf ('\n', index);

```

```

if (index == 0)
return stringPrintText.Length;
}
return index;
}

```

Створюємо обробники подій пунктів mnuPageSetup і mnuPrintPreview головного меню:

```

private void mnuPageSetup_Click (object sender, System.EventArgs e)
{
// Показуємо діалог
pageSetupDialog1.ShowDialog ();
}
private void mnuPrintPreview_Click (object sender, System.EventArgs
e)
{
// Ініціалізуємо змінні
printDocument1.DocumentName = Text;
stringPrintText = rtbText.Text;
StartPage = 1;
NumPages = printDialog1.PrinterSettings.MaximumPage;
PageNumber = 1;
// Показуємо діалог
printPreviewDialog1.ShowDialog ();
}

```

В обробнику пункту меню mnuPrint визначаємо діапазон сторінок – всі сторінки, ряд або виділена область:

```

private void mnuPrint_Click (object sender, System.EventArgs e)
{
printDialog1.AllowSelection = rtbText.SelectionLength > 0;
if (printDialog1.ShowDialog () == DialogResult.OK)
{
printDocument1.DocumentName = Text;
// Визначаємо діапазон сторінок для друку
switch (printDialog1.PrinterSettings.PrintRange)
{
// Вибрані всі сторінки
case PrintRange.AllPages:
stringPrintText = rtbText.Text;
StartPage = 1;
NumPages = printDialog1.PrinterSettings.MaximumPage;
break;
// Обрана виділена область
case PrintRange.Selection:
stringPrintText = rtbText.SelectedText;
StartPage = 1;
NumPages = printDialog1.PrinterSettings.MaximumPage;
break;
// Вибрано ряд сторінок
case PrintRange.SomePages:
stringPrintText = rtbText.Text;
StartPage = printDialog1.PrinterSettings.FromPage;

```

```

NumPages = printDialog1.PrinterSettings.ToPage - StartPage + 1;
break;
}
PageNumber = 1;
// Викликаємо вбудований метод для початку друку
printDocument1.Print ();
}
}

```

Запускаємо програму. Зміни, зроблені в діалоговому вікні "Параметри сторінки", зберігаються, і у вікні попереднього перегляду можна бачити підготовлену для друку сторінку.

З вікна попереднього перегляду можна відразу перейти до друку. При виборі пункту друку доступний вибір ряду сторінок або печатка виділеного фрагмента.

Все те, що ми бачимо на екрані в діалоговому вікні попереднього перегляду, теоретично повинно відображатися точно так само і на папері. Однак при реалізації власної печатки, напевно, хочеться перевірити все до самого кінця — до виведення на папір. У самому кінці розробки це дійсно варто зробити, але на проміжних етапах робота з принтером може зайняти багато часу. Полегшити завдання допоможе програма pdfFactoryPro — при її установці в системі з'являється віртуальний принтер, що дозволяє зберігати документи у форматі pdf.

4.2.2 Створення елементів керування за допомогою. NET Framework

4.2.2.1 Застосування GDI +

Windows – операційна система з графічним інтерфейсом. Користувачі працюють з вікнами, які в графічній формі представляють дані і функціональність додатків. Щоб користувачі могли застосовувати багаті можливості Windows, конструкція програми повинна відкривати доступ до його функціональності через графічний інтерфейс. .NET Framework надає безліч графічних елементів керування і підтримує різноманітні методи створення програм з графічним інтерфейсом. Проте іноді потрібно формувати і відображати власне графічне подання, а також генерувати елементи керування нестандартного вигляду. Щоб повною мірою скористатися перевагами Windows, необхідно освоїти можливості інтерфейсу графічних пристроїв (Graphic Device

Interface, GDI). У .NET Framework реалізована повністю керована версія цього інтерфейсу під назвою GDI+.

Терміном GDI + в .NET Framework позначають керовану реалізацію GDI, яка застосовується для відображення графічної інформації на екрані комп'ютера. Доступ до цього інтерфейсу надається через шість просторів імен, де реалізовані різні види функцій GDI+.

Простір імен System.Drawing

Простір імен System.Drawing надає надзвичайно широкі можливості. Докладне дослідження компонентів цього простору імен не є метою даного навчального курсу, але тут ми коротко розглянемо застосування його класів і методів (табл. 4.4).

Таблиця 4.4 – Простір імен System.Drawing

Ім'я	Що містить
System.Drawing	Основні класи, які беруть участь у відображенні графіки на екрані. Це ключовий простір імен для програмування графічного інтерфейсу
System.Drawing.Design	Класи, надають користувачу інтерфейсу додаткові можливості періоду розробки
System.Drawing.Drawing2D	Класи, що відображають складні графічні ефекти
System.Drawing.Imaging	Класи, що надають додаткові можливості по маніпулюванню графічними файлами
System.Drawing.Printing	Класи для друку
System.Drawing.Text	Класи, що надають додаткові можливості з керування шрифтами

Об'єкт Graphics

Ключову роль у формуванні та відображенні графіки грає клас Graphics з простору імен System.Drawing. Об'єкт Graphics представляє область для малювання, розташовану на поверхні графічного елемен-

ту, — форми, елемента керування або об'єкта Image. З усіма цими сутностями пов'язаний об'єкт Graphics, який дозволяє малювати на їх поверхні та відображає всі графічні елементи, розміщені на ній. Оскільки об'єкт Graphics повинен бути пов'язаний з графічним елементом, його не можна створити, викликавши конструктор цього класу безпосередньо. Замість цього потрібно викликати конструктор об'єкта Graphics з графічного елемента. Будь-який клас-нащадок Control (включаючи Form) підтримує метод CreateGraphics. Останній повертає посилання на об'єкт Graphics, пов'язаний з даним елементом керування. Наступний приклад демонструє отримання доступу до об'єкта Graphics форми myForm:

```
System.Drawing.Graphics myGraphics;  
myGraphics = myForm.CreateGraphics();
```

У результаті виконання цього коду створюється об'єкт Graphics, яким можна користуватися для відображення графіки на поверхні форми.

При роботі із зображеннями метод Graphics.FromImage дозволяє створити об'єкт Graphics, пов'язаний із заданим об'єктом Image. Це статичний метод, тому для його виклику посилання на об'єкт Graphics не потрібно. Об'єктом Image вважається будь-який об'єкт-нащадок класу Image, наприклад Bitmap. Ось як створити з файлу об'єкт Bitmap і отримати пов'язаний з ним об'єкт Graphics:

```
Bitmap myImage = new Bitmap ("C: \\ myImage.bmp");  
System.Drawing.Graphics myGraphics;  
myGraphics = Graphics.FromImage (myImage);
```

Зауважте, що навіть невидимий об'єкт Image дозволяє створити об'єкт Graphics і маніпулювати ним.

Координати. Формування зображення відбувається в області, заданої межами елемента керування. Тут діє двовірна система координат, кожна точка якої задана парою значень X і Y . За замовчуванням початкова точка системи координат (0,0) кожного елемента перебуває в його верхньому лівому куті, а координати задаються в пікселях. Простір імен System.Drawing містить ряд структур (таблиця 4.5), що описують точку або область у даній системі координат.

Таблиця 4.5 – Структури, що описують координати точок і фігур

Ім'я	Опис
Point	Представляє точку з координатами X та Y, заданими значеннями типу Integer (Int).
PointF	Представляє точку з координатами X та Y, заданими значеннями типу Single (float)
Size	Представляє прямокутну область, розміри якої задані парою значень Height і Width типу Integer
SizeF	Представляє прямокутну область, розміри якої задані парою значень Height і Width типу Single (float)
Rectangle	Представляє прямокутну область, межі якої задані значеннями Top, Bottom, Left і Right типу Integer (Int)
RectangleF	Представляє прямокутну область, межі якої задані значеннями Top, Bottom, Left і Right типу Single (float)

Як видно з таблиці 4.5, існує два типи подібних структур: ті, що приймають цілочисельні значення, і ті, що використовують числа з плаваючою крапкою.

Структури Point, Size і Rectangle першого типу дозволяється явно перетворювати у відповідні структури другого типу. Щоб виконати таке перетворення, слід явно перетворити кожен координату із типу з плаваючою крапкою в цілочисельний тип, наприклад:

```
Point myPoint;
PointF myPointF = new PointF (13.5F, 33.21F);
myPoint = new Point ((int) myPointF.X, (int) myPointF.Y);
```

Зверніть увагу на важливу особливість структур Size і Rectangle: обидві вони визначають прямокутну частину клієнтської області, однак Size визначає тільки розмір, але не положення прямокутника, тоді як структура Rectangle задає реальний стан прямокутника в клієнтській області. Для створення Rectangle потрібні дві структури: Size і Point, що вказують положення верхнього лівого кута прямокутника:

```
Point myOrigin = new Point (10, 10);
Size mySize = new Size (20, 20);
```

```
// Створити прямокутник розміром 20 на 20 пікселів, верхній
// Лівий кут якого розташований у точці з координатами (10,10),
Rectangle myRectangle = new Rectangle (myOrigin, mySize);
```

Малювання фігур. Об'єкт Graphics інкапсулює безліч методів для малювання на екрані простих і складних фігур. Існує два основні різновиди цих методів. До першої відносяться методи, імена яких починається з «Draw» (таблиця 4.6). Ці методи використовуються для малювання прямих, дуг і контурів фігур. Інший різновид методів, імена яких починається з «Fill» (таблиця 4.7), служить для малювання залитих фігур-прямокутників, еліпсів і багатокутників.

Ці методи приймають різні комбінації параметрів, що визначають розміри і положення відображуваних фігур. Крім того, їм потрібен об'єкт, що формує зображення фігури, для контурних фігур це Pen, а для зафарбованих – Brush.

Таблиця 4.6 – Методи для малювання контурів фігур

Ім'я	Опис
DrawArc	Малює дугу, що представляє частину еліпса
DrawBezier	Малює криву Безьє
DrawBeziers	Малює кілька кривих Безьє
DrawClosedCurve	Проводить через кілька точок замкнуту криву
DrawCurve	Проводить через кілька точок незамкнену криву
DrawEllipse	Малює еліпс, вписаний в прямокутник
DrawLine	Поєднує дві точки лінією
DrawPath	Малює заданий об'єкт GraphicsPath, представляє складну фігуру
DrawPie	Малює сектор, кульовий
DrawPolygon	Малює багатокутник за заданою кількістю точок
DrawRectangle	Малює прямокутник
Draw Rectangles	Малює кілька прямокутників

Таблиця 4.7 – Методи для малювання зафарбованих фігур

Ім'я	Опис
FillClosedCurve	Малює замкнуту криву, задану масивом точок, і зафарбовує отриману фігуру
FillEllipse	Малює зафарбований еліпс
FillPath	Малює і зафарбовує об'єкт Graphics Path, представляє складну фігуру
FillPie	Малює зафарбований сектор
FillPolygon	Малює багатокутник, заданий масивом точок, і зафарбовує його
FillRectangle	Малює зафарбований прямокутник
FillRectangles	Малює кілька зафарбованих прямокутників
FillRegion	Малює і зафарбовує об'єкт Region, зазвичай представляє складну фігуру

Об'єкти Color, Brush і Pen

Об'єкти Color, Brush і Pen формують вигляд графічного елементу. Об'єкт Brush використовують для відображення зафарбованих фігур, Pen – для відображення прямих і дуг, а об'єкт Color задає колір фігури.

Color. Структура Color розташовується в просторі імен System.Drawing і представляє певний колір. Кожен колір визначає чотири значення: Alpha, яке представляє прозорість, і три значення Red, Green і Blue з діапазону 0-255. Щоб створити новий колір, необхідно передати ці значення методу Color.FromArgb:

```
Color myColor;
myColor = Color.FromArgb (128, 255, 12, 43);
```

Якщо ви не використовуєте ефект прозорості, параметр Alpha можна пропустити, вказавши лише значення Red, Green і Blue:

```
Color myColor;
myColor=Color.FromArgb (255,12,43);
```

Крім того, безліч стандартних кольорів є в .NET Framework у вигляді іменованих кольорів:

```
Color myColor;
myColor = Color.Tomato;
```

Brush. Об'єкт Brush представляє кисть і служить для малювання зафарбованих фігур. Всі різновиди кистей є нащадками абстрактного класу Brush і дозволяють створювати зафарбовані фігури різного ви-

ду. Типи кистей і простору імен, в яких розташовані відповідні об'єкти, описані в 4.8.

Таблиця 4.8 – Типи кистей

Тип	Простір імен	Опис
SolidBrush	System.Drawing	Суцільна одноколірна кисть
TextureBrush	System.Drawing	Кисть для заливки замкнутого контуру зображенням
HatchBrush	System.Drawing.Drawin g2D	Кисть для візерункової заливки
LinearGradientBrush	System.Drawing.Drawin g2D	Кисть для заливки двоколірним градієнтом
PathGradientBrush	System.Drawing.Drawin g2D	Кисть, що використовує складну градієнтну заливку

Для створення об'єкта SolidBrush достатньо вказати колір, наприклад:

```
SolidBrush myBrush = new SolidBrush (Color.PapayaWhip);
```

Конструктори інших кистей складніше і вимагають додаткових параметрів. Наприклад, для створення кисті типу TextureBrush необхідний об'єкт Image, а для LinearGradientBrush — два кольори і ряд інших параметрів, у залежності від типу вибраного конструктора.

Pen. Об'єкти Pen представляють пір'я, вони призначені для малювання прямих і дуг, а також для застосування до контурним фігур різних графічних ефектів.

Існує тільки один клас Pen, і він є запечатаним (sealed). Створити об'єкт класу Pen нескладно, достатньо вказати його колір:

```
Pen myPen = new Pen (Color.BlanchedAlmond);
```

В результаті виконання попереднього коду створюється перо, ширина якого за замовчуванням дорівнює 1 пікселю. Ширину задають в конструкторі об'єкту Pen наступним чином:

```
Pen myPen = new Pen (Color.Lime, 4);
```

Цей код призначає ширину пера, рівну чотирьом пікселям. Перо можна створити і на основі наявної кисті, вид такого пір'я буде узгоджений зі стилем інтерфейсу. Це особливо зручно при використанні складних тіней та інших ефектів. Наступний приклад демонструє створення пера на основі наявної кисті myBrush:

```
Pen myPen = new Pen (myBrush);
```

При створенні пера на основі кисті також дозволяється ставити її ширину. Системні кисті, пір'я і кольору.

При розробці користувацького інтерфейсу іноді нестандартні компоненти доводиться оформляти в тому ж стилі, що й системні. Для цього .NET Framework надає доступ до системних кольорів через клас SystemColors, який містить набір статичних членів, що представляють поточні системні кольори. Так, при розробці нестандартних компонентів користувацького інтерфейсу можна призначити їм системні кольори, щоб у період виконання вони відображалися з використанням поточної системної палітри. Отримати доступ до системних кольорів можна так:

```
Color myColor = SystemColors.HighlightText;
```

Поряд з SystemColors, у .NET Framework також є класи SystemPens і System Brushes, що надають доступ до пір'я і кистей, які застосовуються за замовчуванням. Ці кисті працюють так само, як будь-які інші, представлені класами Pen або SolidBrush.

Малювання простих фігур

Клас Graphics підтримує ряд методів, що дозволяють малювати прості фігури.

Будь-яким методам для малювання контурних фігур необхідний дійсний об'єкт Pen, а методам, що малює зафарбовані фігури, – дійсний об'єкт Brush. Крім того, при виклику цих методів варто передати будь-які необхідні їм об'єкти. Так, наприклад, можна намалювати прямокутник засобами методу.

```
DrawRectangle:
// Створити об'єкт Rectangle.
Rectangle myRectangle = new Rectangle(0, 0, 30, 20);
// Створити об'єкт Graphics, пов'язаний з формою,
Graphics g = this.CreateGraphics();
// Намалювати прямокутник системним пером.
g.DrawRectangle (SystemPens, ControlDark, myRectangle);
// Звільнити ресурси, зайняті об'єктом Graphics.
g.Dispose();
```

Закінчивши роботу з об'єктом `Graphics`, обов'язково викличте його метод `Dispose`. В іншому випадку швидкодія програми може знизитися, тому що цей об'єкт використовує багато системних ресурсів. Аналогічно слід чинити з будь-якими створеними вами об'єктами `Pen` і `Brush`. Наступний приклад коду показує, як намалювати зафарбований еліпс і правильно звільнити об'єкти `Brush` і `Graphics`:

```
SolidBrush myBrush = new SolidBrush (Color.MintCream);
Graphics g = this.CreateGraphics();
// Намалювати еліпс, вписаний в прямокутник.
Rectangle myRectangle = new Rectangle (0, 0, 30, 20);
g.FillEllipse (myBrush, myRectangle);
// Звільнити ресурси об'єктів Graphics і Brush.
g.Dispose();
myBrush.Dispose ();
```

Як намалювати просту фігуру:

- створить об'єкт `Graphics`, що представляє область, в якій ви хочете намалювати фігуру;
- створить необхідні допоміжні об'єкти. До них відносяться об'єкти, що задають координати і розміри фігур, наприклад `Point` або `Rectangle`, а також об'єкти `Pen` (для малювання контурних фігур) і `Brush` (для зафарбованих фігур);
- викличе відповідний метод об'єкта `Graphics`;
- звільнить всі ресурси, зайняті створеними об'єктами `Pen` або `Brush`;
- звільнить ресурси об'єкта `Graphics`.

Відображення тексту

Метод `DrawString` об'єкта `Graphics` дозволяє відобразити на екрані текстовий рядок як графічний елемент, намалювавши її в певному місці екрану заданим пензлем. Крім об'єкта `Brush`. Цим методом необхідно передати координати верхнього лівого кута відображуваного тексту (у вигляді структури `PointF`). Нижче показаний приклад коду, що відображає рядок за допомогою методу

`Graphics.DrawString`:

```
// У цьому прикладі використовується одна з
// Системних кистей класу SystemBrush.
Graphics g = this.CreateGraphics();
String myString = "Hello World";
Font myFont = new Font ("Times New Roman", 36, FontStyle, Regular);
//Останні два параметри представляють координати рядка.
g.DrawString (myString, myFont, SystemBrushes.Highlight, 0, 0);
```

```
// Не забудьте звільнити об'єкт Graphics!
g.Dispose();
```

Відображення тексту як графічного елементу:

- при необхідності створить об'єкти Font і Brush, що визначають вид об'єкта String;
- отримайте посилання на об'єкт Graphics, пов'язаний з областю, в якій повинен бути відображений текст;
- викличте метод Graphics.DrawString, передавши об'єкти String, Font, Brush і координати верхнього лівого кута області, в якій буде відображено текст.;
- звільніть об'єкт Graphics, викликавши метод Graphics.Dispose.

Малювання складних фігур

Час від часу доводиться малювати фігури більш складні, ніж прямокутники, еліпси і багатокутники. У об'єктів простих фігур є вбудовані методи, які полегшують їх малювання, тоді як малювання складних фігур вимагає додаткової підготовки. Ключовим для візуалізації складних фігур є об'єкт Graphics Path з простору імен System.Drawing.Drawing2D. Він описує довільний замкнутий контур або набір фігур. Так, можна створити об'єкт GraphicsPath, утворений еліпсами, прямокутниками та іншими простими об'єктами, що представляє фігуру неправильної форми.

Створення об'єкта GraphicsPath

Об'єкт GraphicsPath створюють викликом однією з версій його конструктора. Найпростіша версія конструктора об'єкта GraphicsPath не приймає параметрів:

```
GraphicsPath myPath = new Drawing2D.GraphicsPath ();
```

Крім того, об'єкт GraphicsPath допустимо задавати у вигляді масиву точок і значень типу Byte. Точки визначають координати контуру, а значення Byte – тип ліній, якими ці точки з'єднуються. Перетворивши подібний масив з перерахуемого типу System.Drawing.Drawing2D.PathPointType, ви зможете написати більш зрозумілий і зручний у супроводі код. Ось приклад коду, що створює дуже простий об'єкт GraphicsPath:

```
// Цей приклад припускає наявність оператора using
//System.Drawing.Drawing2D
GraphicsPath myPath = new GraphicsPath (new Point [] {new Point (1,
1),
new Point (32, 54), new Point (33, 5)}, new byte [] {
```

```
(Byte) PathPointType.Start, (byte) PathPointType.Line,  
(Byte) PathPointType.Bezier));
```

До створеного об'єкту `GraphicsPath` можна додавати фігури. Фігура — це замкнутий контур, він може бути простим (як еліпс або прямокутник) або складним (як довільні криві і контури символів).

Методи класу `GraphicsPath` дозволяють додавати до контуру нові фігури (таблиця 4.9).

Таблиця 4.9 – Методи додавання фігур до об'єкта `GraphicsPath`

Ім'я	Що додає
<code>AddClosedCurve</code>	замкнуту криву, задану масивом точок
<code>AddEllipse</code>	еліпс
<code>AddPath</code>	поставлений екземпляр об'єкта <code>GraphicsPath</code>
<code>AddPie</code>	окружність
<code>AddPolygon</code>	багатокутник, заданий масивом точок
<code>AddRectangle</code>	Прямокутник
<code>AddRectangles</code>	масив прямокутників
<code>AddString</code>	графічне подання рядка символів із заданим накресленням

Дозволяється не тільки безпосередньо додавати фігури до об'єкта `GraphicsPath`, а й складати контури, додаючи прямі, криві і дуги. Щоб почати новий контур, викличте метод `GraphicsPath.StartFigure`. Далі, використовуючи методи класу `GraphicsPath`, можна додавати до контуру нові фігури. Закінчивши конструювання фігури, викличте метод `GraphicsPath.CloseFigure`, щоб замкнути її контур – остання точка фігури автоматично з'єднується з першою. Ось приклад:

```
GraphicsPath myPath = new Drawing2D.GraphicsPath ();  
myPath.StartFigure ();
```

Тут повинен бути код, який додає елементи контуру.

```
myPath.CloseFigure ();
```

Якщо викликати метод `StartFigure` два рази поспіль, не викликавши між ними метод `CloseFigure`, то фігура, створена першим викликом `StartFigure`, залишиться незамкненою.

Примітка. При відображенні в період виконання будь-яка незамкнута фігура автоматично замикається додаванням прямої, що з'єднує першу і останню крапку контуру цієї фігури.

У таблиці 4.10 описані методи об'єкта GraphicsPath, що дозволяють додавати нові елементи контуру фігури.

Таблиця 4.10 — Методи для додавання елементів контуру

Ім'я	Що додає
AddArc	Дугу
AddBezier	Криву Без'є
AddBeziers	Набір кривих Без'є
AddCurve	Криву, задану масивом точок
AddLine	Пряму
AddLines	Набір з'єднаних прямих

Як намалювати складну фігуру:

- отримайте посилання на об'єкт Graphics, пов'язаний з областю малювання, в якій потрібно намалювати фігуру;
- створіть новий екземпляр класу GraphicsPath;
- додайте до нього фігури за допомогою методів класу GraphicsPath;
- викличте метод Graphics.DrawPath, щоб намалювати контур, або Graphics.FillPath, щоб намалювати закрашеної фігури, задану об'єктом GraphicsPath;
- звільніть об'єкт Graphics.

4.2.2.2 Створення елементів керування

В основі програмування із застосуванням Windows Forms лежить використання елементів керування. Вони інкапсулюють окремі функції програми та є їх графічним поданням для користувача. .NET Framework надає в розпорядження розробника ряд інструментів і методів для створення власних елементів керування. Елементи керування .NET Framework — це спеціалізовані класи, які містять код, що формує їх графічний інтерфейс. Існує декілька джерел елементів керування, що застосовуються в додатках. По-перше, це стандартні елементи

нти керування Visual Studio .NET, розроблені Microsoft. Вони забезпечують широкі можливості та дозволяють розробляти клієнтські програми з багатою функціональністю. По-друге, у додатку припустимо застосовувати елементи керування від сторонніх розробників, а також існуючі елементи керування ActiveX. Елементи керування від сторонніх розробників підтримують можливості, відсутні в стандартних компонентів з .NET Framework. Нарешті, якщо функції, необхідні вашому додатку, не реалізовані ні в стандартних елементах керування .NET Framework, ні в елементах керування сторонніх розробників, дозволяється створювати власні елементи керування.

Принципи створення елементів керування

Всі елементи керування .NET Framework є прямими або непрямими нащадками класу Control і успадковують від нього базовий набір низькорівневої функціональності, необхідної всім елементам керування. Наприклад, Control надає код для обробки даних, введених користувачем за допомогою клавіатури і миші, а також код, який взаємодіє з ОС. Крім того, класи елементів керування отримують від свого прашура набір властивостей, методів і подій, загальних для всіх елементів керування. Однак базовий клас не передає своїм нащадкам ні специфічних функцій, ні коду, що формує інтерфейс елемента керування.

Основних підходів до створення елементів керування три:

- спадкоємство від існуючих елементів керування;
- об'єднання стандартних елементів керування у групи;
- написання нестандартних елементів керування «з нуля».

Спадкування від існуючих елементів керування

Простіше за все створити новий елемент керування, оголосивши його на основі існуючого. Створений таким чином елемент керування не тільки зберігає всю функціональність базового компоненту, але і дозволяє доповнити її новими можливостями.

Крім функціональності, похідний елемент керування успадкує від свого предка вигляд інтерфейсу. Наприклад, елемент керування, похідний від Button, зовні не відрізняється від звичайної кнопки. Це звільняє від створення інтерфейсу для похідних елементів керування, але при необхідності це можна зробити.

Більшість елементів керування Windows Forms, якщо вони не визначені з ключовим словом `NotInheritable (sealed)`, можуть бути базовими класами для нових елементів керування. Наприклад, це дозволяє створити новий елемент керування `TextBox` з вбудованим кодом для перевірки значення поля або `PictureBox` підтримуючий установку користувачем фільтрів для відображаються в ньому зображень. Спадкування також дозволяє створювати елементи керування, функціонально ідентичні базовому елементу керування, але різні за зовнішнім виглядом. Приклад — нестандартна кнопка, яка відрізняється від традиційної прямокутної кнопки круглою формою.

Оголошення нових елементів керування за допомогою наслідування вважається найпростішим способом їх створення і вимагає мінімальних витрат часу. Цей підхід оптимальний, якщо є потреба у більшості або всі можливості існуючого елемента керування Windows Forms новому елементу керування, додавши лише кілька нестандартних функцій. Даний спосіб зручний і в тих випадках, коли потрібно змінити вигляд елемента керування, зберігши його функціональність. Якщо ж потрібно створити елемент керування з кардинально іншою функціональністю, не слід оголошувати його на основі існуючого; до цього випадку застосовуються способи, описані в наступних розділах.

Спадкування від класу `UserControl`

Іноді функціональності одного елемента керування недостатньо для вирішення поставлених завдань. Припустимо, вам потрібен елемент керування, що підтримує зв'язування з джерелом даних і відображає ім'я, прізвище і номер телефону в елементах керування `TextBox`. Можна додати до форми відповідний код, але раціональніше створити новий елемент керування, що містить кілька текстових полів. Це особливо зручно, коли подібні можливості потрібні декільком додаткам. Елементи керування, утворені об'єднаними в групу стандартними елементами керування Windows Forms, називаються призначеними для користувача або складовими.

Складові елементи керування оголошують на основі класу `UserControl` надає базову функціональність, яку можна розширити шляхом додавання властивостей, методів, подій та інших елементів керування. Дизайнер складових елементів керування дозволяє розміщувати на їх поверхні стандартні елементи керування Windows Forms і додавати код, який реалізує специфічні можливості. Таким чином,

користувач може створювати нові елементи керування, об'єднуючи стандартні елементи керування Windows Forms в логічні групи.

При розробці графічного інтерфейсу для користувача елементів керування доступні не всі можливості, але розробнику, як правило, досить лише налаштувати і правильно розташувати стандартні елементи керування.

Створення користувацького елемента керування не завжди просте завдання — це залежить від вимог до функціональності нового компонента. Спосіб, описаний у цьому розділі, зручний для створення нових компонентів, що поєднують функціональність стандартних елементів керування та нестандартні можливості.

Спадкування від класу Control

Якщо потрібен більш складний інтерфейс або можливості, які не вдається реалізувати із застосуванням складових та похідних елементів керування, звертаються до створення формованих елементів керування. Формовані елементи є прямими нащадками Control — базового класу всіх елементів керування. Клас Control передає «у спадщину» великий об'єм функціональності, загальною для всіх елементів керування, наприклад підтримку взаємодії з мишею і спільні події (наприклад, Click), а також полегшують роботу з елементами керування властивості, такі, як Font, ForeColor, BackColor, Visible і т.д.

Тим не менш клас Control не надає похідним від нього елементів керування ніяких спеціальних можливостей. Всю логіку, необхідну для вирішення специфічних завдань даного елемента керування, повинен реалізувати розробник. Хоча клас Control надає значну частину властивостей, що визначають обличчя інтерфейсу, розробнику доведеться самотійно написати весь код графічного інтерфейсу елемента керування.

Створення формованих елементів керування потребує найбільше часу і зусиль від програміста, оскільки на його плечі лягає не тільки реалізація нестандартних можливостей даного елемента керування, а й створення його графічного представлення, що може відняти багато часу. Формувати елементи керування самотійно потрібно, тільки якщо потрібна складна форма або функціональність, яку не надає жоден інший тип елементів керування.

Додавання членів до елементів керування

Процедура додавання властивостей, методів, полів і подій до елементів керування не відрізняється від додавання членів до звичайного класу. Члени класів елементів керування дозволяється оголошувати з будь-яким підходящим рівнем доступу, щоб зробити їх доступними формі, на якій розміщений елемент керування. Відкриті властивості, додані до елементів керування в період розробки, автоматично відображаються у вікні Properties, де їх може відредагувати і налаштувати будь-який користувач. Якщо ви не хочете, щоб властивість відображалася у вікні Properties, додайте до його оголошення атрибут Browsable і встановіть його в false.

Ось як це робиться:

```
<System.ComponentModel.Browsable (false)>
public int StockNumber
{
    // Реалізація властивості опущена.
}
```

Створення похідних елементів керування

Похідний елемент керування містить всю функціональність свого базового класу, а також ряд додаткових можливостей. Щоб створити похідний елемент керування, необхідно вказати для нього базовий клас, наприклад клас стандартного елемента керування Windows Forms. Новий елемент керування, створений таким чином, збереже зовнішній вигляд і поведінку свого прашура. Нижче показаний приклад створення елемента керування, похідного від Button;

```
public class myButton: System.Windows.Forms.Button
{
    // Інший код опущений.
}
```

Реалізація нових можливостей в похідних елементах керування

Як правило, до похідних елементів керування вдаються, коли потрібно реалізувати в них додаткові функції, відсутні в стандартних елементах керування Windows Forms. Для цього до похідних елементів керування додають новий код або перевизначають члени, успадковані від базового класу. Припустимо, що розробнику потрібно створити текстове поле, куди користувач зможе вводити лише числа. Для цього необхідно перевизначити метод OnKeyPress наступним чином:

```
public class NumberBox: System.Windows.Forms.TextBox
{
protected override void OnKeyPress (KeyPressEventArgs e)
{
if (char.IsNumber (e.KeyChar) = false)
e.Handled = true;
}
}
```

Модифікація зовнішнього вигляду існуючих елементів керування

Інша причина, по якій створюють похідні елементи керування, — їх нестандартний зовнішній вигляд. Щоб змінити стиль оформлення існуючого елемента керування, перевизначите метод `OnPaint`, помістивши в нього власний код. Наприклад, щоб змінити форму елемента керування, встановіть відповідним чином його властивість `Region` в методі `OnPaint`. Клас `Region`, подібно `GraphicsPath`, описує область екрану заданої форми, його отримують з класу `GraphicsPath`. Так, можна згенерувати клас `GraphicsPath`, що представляє форму вашого елемента керування, створити на його основі об'єкт `Region` і помістити той у відповідне властивість елемента керування. Цей прийом ілюструється на прикладі створення кнопки у формі тексту:

```
public class WowButton: System.Windows.Forms.Button
{
protected override void OnPaint(PaintEventArgs pe)
{
System.Drawing.Drawing2D.GraphicsPath myPath = new System.Drawing.Drawing2D.GraphicsPath ();
// Наступна команда встановлює властивості FontFamily та FontStyle
// Елементу керування так, щоб об'єкт GraphicsPath представляв
// Контур рядка символів розміром 72 пункти.
// Врахуйте: у C# необхідно
// Явно перетворити значення Font.Style в ціле число,
myPath.AddString("Wow! ", Font.FontFamily, (int) Font.Style, 72,
new PointF(0, 0), StringFormat.GenericDefault);
// Створити на основі GraphicsPath новий об'єкт Region.
Region myRegion = new Region (myPath);
// Помістити новий об'єкт Region в однойменне властивість
// Елементу керування. this.Region = myRegion;
}
}
```

Зауважте, `WowButton` — це не прямокутна кнопка з прозорим фоном, що відображає великі літери, вона дійсно має форму букв, що становлять слово «Wow!». Щоб її клацнути, необхідно потрапити в область, обмежену контуром букв. Подібним чином змінюють форму

звичайних елементів керування, щоб отримати елементи користувацького інтерфейсу з нестандартним виглядом. Примітка. Промальовування деяких елементів керування, таких, як TextBox, виконується не самим елементом керування, а формою, на якій він розташовується. Такі елементи керування ніколи не генерує подія Paint, тому що відповідає за візуалізацію код поміщений в обробник цієї події, викликаний не буде.

Створення похідного елемента керування:

– оголосіть новий клас елемента керування на основі існуючого класу елемента керування;

– реалізуйте всі необхідні нестандартні можливості та додайте код, що формує інтерфейс похідного елемента керування.

Створення складових елементів керування

Користувальницькі, або складові, елементи керування об'єднують нестандартні можливості з функціональністю стандартних елементів керування Windows Forms, дозволяючи швидко створювати нові елементи керування. Користувацький елемент керування складається з одного або декількох стандартних елементів керування, об'єднаних в групу. Дизайнер UserControl призначений для додавання до складових елементів керування дочірніх елементів керування, також званих вкладеними.

Розробник може написати власний код для обробки подій, генеруються вкладеними елементами керування. Припустимо, що вам потрібно створити користувацький елемент керування, який складається з двох текстових полів і написи. При зміні значень будь-якого з текстових полів цей елемент керування має автоматично складати їх і відображати суму в написи. Подібну функціональність вдається реалізувати, перевизначивши методи OnKeyPress вкладених текстових полів наступним чином:

```
// Для події KeyPress елемента керування TextBox2
// Потрібно створити аналогічний обробник.
protected override void OnKeyPress (object sender, KeyPressEventArgs e)
{
    // Перевірити, чи є натиснута клавіша цифровою.
    if (char.IsNumber (e.KeyChar) == false)
    e.Handled = true;
    Label1.Text = (int.Parse (TextBox1.Text) + int.Parse (TextBox2.Text)). ToString ();
}
```

Вкладені компоненти складових елементів керування вважаються закритими. Тому жоден розробник, який буде працювати з вашим елементом керування, не зможе змінити такі властивості його вкладених елементів керування, як колір, форма і розмір. Щоб дозволити іншим розробникам змінювати властивості вкладених елементів керування, зробіть їх доступними через властивості об'єкта складового елементу керування. Розглянемо складовий елемент керування з вкладеною кнопкою (Button1). Щоб надати доступ до властивості BackColor об'єкту Button1, створіть, наприклад, для нього властивість — оболонку в об'єкта елемента керування, як показано нижче:

```
public color ButtonColor
{
    get
    {
        return Button1. BackColor;
    }
    set
    {
        Button1. BackColor = value;
    }
}
```

Щоб надати доступ до вкладеного елемента керування, змініть його властивість Modifiers. Це можливо тільки через вікно Properties в період розробки, оскільки в період виконання воно не існує. Властивість Modifiers дозволяє задати будь-який рівень доступу, який застосовується до відповідного вкладеному елементу керування.

Як створити складовий елемент керування:

- оголосіть клас, похідний від UserControl;
- викличте дизайнер UserControl і додайте з його допомогою вкладені елементи керування Windows Forms, після чого відповідним чином настройте їх;
- при необхідності зробіть доступними властивості вкладених елементів керування;
- реалізуйте нестандартні можливості, необхідні складеного елементу керування.

Створення формованих елементів керування

Формовані елементи керування надають максимум можливостей по налаштуванню і керуванню своєю функціональністю, але їх розробка займає найбільше часу. Оскільки базовий клас Control не надає своїм нащадкам навіть базового графічного інтерфейсу, розробнику

доведеться самостійно писати весь код, що формує образ такого елемента керування. Якщо інтерфейс складний, його створення займає лівову частку циклу розробки елемента керування.

Процес візуалізації елемента керування в клієнтській області форми називається промальовуванням. Отримавши команду на промальовування, елемент керування генерує подія Paint, при цьому виконуються всі обробники події Paint. У класу Control обробником за замовчуванням для події Paint є метод OnPaint.

Метод OnPaint приймає єдиний аргумент — екземпляр класу PaintEventArgs, що містить інформацію клієнтської області елемента керування. Варто звернути увагу на два члени цього класу — Graphics і ClipRectangle.

Graphics — це об'єкт типу Graphics, що представляє клієнтську область елемента керування. Посилання на нього необхідна для формування зображення елемента керування. ClipRectangle — прямокутник, доступну клієнтську область елемента керування. При першому відображенні елемента керування ClipRectangle обмежує зайняту їм область. Якщо вона перекривається з іншими елементами керування, цей елемент може виявитися частково або повністю прихованим. При повторному його промальовуванні ClipRectangle представляє тільки видиму область, яка повинна бути перемальована. Саме тому не рекомендується визначати розміри елемента керування по об'єкту ClipRectangle для цієї мети слід застосовувати властивість Size.

За замовчуванням координати відлічуються від верхнього лівого кута елемента керування, в якому розташовується умовна точка початку координат (0,0), вони обчислюються в пікселях. Наступний приклад демонструє простий метод OnPaint, що виконує промальовування нестандартного елемента керування у вигляді червоного еліпса:

```
// Цей приклад припускає наявність оператора Imports System.Drawing
protected override void OnPaint (PaintEventArgs e)
{
    Brush aBrush = new SolidBrush (Color.Red);
    Rectangle clientRectangle = new Rectangle (new Point (0,0),
    this.Size);
    e.Graphics.FillEllipse (aBrush, clientRectangle);
}
```

При зміні розмірів елемента керування автоматично змінюється розмір ClipRectangle, але повторна промальовування цього елемента керування потрібно не завжди. Щоб при зміні розмірів елемента керування він кожен раз виповнювався заново, викличте його з конструк-

тора методом `Control.SetStyle` і встановить прапор `ResizeRedraw` в `true`, як показано в наступному прикладі:

```
SetStyle (ControlStyles.ResizeRedraw, true);
```

Щоб у будь-який час перемалювати елемент керування вручну, достатньо викликати метод `Refresh`:

```
Refresh();
```

Як створити формований елемент керування:

оголосіть клас, похідний від класу `Control`;

додайте в метод `OnPaint` код, що візуалізує елемент керування.

Реалізуйте всі нестандартні можливості, необхідні вашому елементу керування.

4.2.2.3 Розв'язання типових задач, пов'язаних з розробкою елементів керування

При розробці будь-яких типів елементів керування виникають різні завдання, одні частіше, інші рідше. На цьому занятті ви навчитеся вирішувати типові завдання, супроводжуючи розробці елементів керування.

Додавання елементів керування на панель Toolbox

Зручно, коли власні елементи керування завжди під рукою. Щоб зробити їх доступними при розробці будь-яких проектів і спростити роботу з ними, додайте їх на інструментальну панель `Toolbox`, звідки їх у будь-який час можна перетягнути на форму.

Це завдання дозволяє вирішити команда `Customize Toolbox`, доступна через контекстне меню панелі `Toolbox`. Клацніть правою кнопкою панель `Toolbox` і виберіть команду `Customize Toolbox`, щоб викликати діалогове вікно `Customize Toolbox`. У ньому виберіть завантажений з елементом керування, який потрібно додати на панель `Toolbox`.

Додавання елемента керування на інструментальну панель `Toolbox`:

- клацніть правою кнопкою миші панель `Toolbox` і виберіть команду `Customize Toolbox` — відкриється однойменне діалогове вікно;
- перейдіть на вкладку `.NET Framework Components` та клацніть на кнопці `Browse`, щоб викликати діалогове вікно `File`;
- відкрийте папку з `DLL` або `EXE`-файлом, в якому знаходиться потрібний елемент керування. Виділіть потрібний файл і натисніть кнопку `Open`;

– Переконайтеся, що обраний елемент керування доданий до списку діалогового вікна Customize Toolbox і клацніть ОК — елемент керування з'явиться на панелі Toolbox.

Призначення значка елемента керування

Visual Studio .NET надає значок для відображення нестандартних елементів керування, доданих до панелі Toolbox, але можна призначити для цієї мети і власне растрове зображення. Далі детально розказано, як це зробити. Значок для відображення на панелі Toolbox призначають за допомогою класу `ToolboxBitmapAttribute` — спеціалізованого атрибуту, що надає метадані елемента керування. Він дозволяє призначити елементу керування растрове зображення розміром 16x16 пікселів або задати значок за значенням `Type (type)`. В останньому випадку елементу керування присвоюється значок, прийнятий для об'єкта даного типу.

Атрибут `ToolboxBitmapAttribute` входить до оголошення класу елемента керування. У Visual C# цей атрибут укладають у квадратні дужки і також розташовують у рядку оголошення класу елемента керування, перед самим оголошенням. Як призначити елементу керування значок, вказавши файл з його зображенням: вкажіть файл із зображенням значка за допомогою атрибуту `ToolboxBitmapAttribute`:

```
[ToolboxBitmap (@ "C:\Pasta.bmp")]
public class PastaMaker : Control
{
    // Реалізація опущена.
}
```

Як призначити елементу керування значок на основі його типу: додайте до оголошення класу елемента керування атрибут `ToolboxBitmapAttribute` з посиланням на тип даного елемента керування:

```
[ToolboxBitmap(typeof (Button))]
public class myButton: Button
{
    // Реалізація опущена.
}
```

Налагодження елементів керування

У циклі розробки будь-якого компонента рано чи пізно настає етап налагодження, і елементи керування тут не виняток. Оскільки вони не є проектами, для налагодження їх необхідно помістити в спе-

ціальний проект Windows Forms. Перед тестуванням елемента керування його необхідно скомпілювати.

Після компонування проекту, в якому знаходиться ваш елемент керування, йому можна перешкодити на формі та налагоджувати, як зазвичай.

Якщо ваш елемент керування є частиною проекту виконуваного компоненту, наприклад програми Windows Forms, для цього тестування слід додати до проекту нову форму, а якщо елемент керування входить у проект бібліотеки класів або елементів керування, доведеться додати до вирішення додатковий проект.

Налагодження елемента керування в проекті Windows Forms:

- виконайте компонування рішення, вибравши в меню Build команду Build Solution;

- при необхідності додайте в проект нову форму і зробіть її початковою формою проекту;

- додайте елемент керування на форму через графічний інтерфейс або програмно. У першому випадку спочатку додають елемент керування на панель Toolbox, щоб з ним було простіше працювати в період розробки;

- запустіть програму, натиснувши F5, — тепер можна приступати до налагодження елемента керування: встановіть точки переривання, застосовуйте покрокове виконання та інші налагодження засоби, що дозволяють знаходити помилки в коді;

- виправивши помилки, перекомпілюйте елемент керування, щоб внесені до нього зміни вступили в силу. Для цього виберіть у меню Build команду Rebuild Solution. Налагодження елемента керування в проекті бібліотеки класів або елементів керування:

- виконайте компонування рішення, вибравши в меню Build команду Build Solution;

- у меню File виберіть команду Add Project \ New Project, щоб додати до вирішення новий проект, — відкриється діалогове вікно Add New Project;

- у діалоговому вікні Add New Project виберіть Windows Application, дайте проекту ім'я та клацніть OK;

- у вікні Solution Explorer клацніть правою кнопкою миші вузол нового проекту References і виберіть з контекстного меню команду Add Reference — відкриється діалогове вікно Add Reference;

– перейдіть на вкладку Projects. Якщо проект, що містить елемент керування, зазначений на цій вкладці, виділіть його і клацніть (Ж, у противному випадку натисніть кнопку Browse і знайдіть папку з DLL-файлом проекту, в якому знаходиться відладжується елемент керування. Виділіть його і натисніть кнопку ОК.

– додайте елемент керування на форму через графічний інтерфейс або програмно. У першому випадку іноді спочатку доводиться додати елемент керування на панель Toolbox, спростити роботу з ним в період розробки;

– запустіть програму, натиснувши F5 — тепер можна приступати до налагодження елемента керування: встановіть точки переривання, застосуйте покрокове виконання та інші налагодження засоби, що дозволяють знаходити і усувати помилки в коді.

Керування ліцензуванням елементів керування

В .NET Framework є вбудована підтримка керування ліцензуванням елементів керування. Ліцензування елементів керування дозволяє їх творцям захистити свою інтелектуальну власність: надати доступ до своїх елементів керування тільки авторизованим розробникам. Модель ліцензування, прийнята за замовчуванням, вимагає, щоб всі ліцензовані елементи керування були позначені атрибутом License Provider Attribute, що вказує об'єкт License.

Provider — він необхідний для перевірки автентичності ліцензії. License Provider — це абстрактний клас, що надає стандартний інтерфейс перевірки автентичності. У конструкторі елемента керування слід викликати метод LicenseManager.Validate, щоб отримати посилання на дійсну ліцензію. Якщо ліцензія на елемент керування відсутня, його завантаження закінчиться невдачею. Метод LicenseManager.Validate викликає метод GetLicense заданого об'єкта License Provider, який витягує ліцензію і перевіряє її за допомогою методу IsValid об'єкта LicenseProvider. Якщо IsValid повертає true, то ліцензія вважається дійсною, у противному випадку цей елемент керування не завантажується.

Застосовувана схема перевірки залежить від реалізації LicenseProvider. .NET — реалізація класу LicenseProvider називається LieFileLicenseProvider. Якщо цей клас заданий як LicenseProvider, він застосовується для перевірки ліцензії. Метод GetLicense цього класу шукає в каталозі з DLL-файлом елемента керування текстовий файл з

ім'ям `FullName.LIC`, де `FullName` — повне ім'я елемента керування. Потім `IsValid` перевіряє наявність в першому рядку текстового файлу підрядка «`my ClassName is a licensed component`», де `myClassName` — повне ім'я компонента. Розробник легко замінить цей алгоритм перевірки власним, перевизначивши відповідні методи. У кожному ліцензованій елементі керування повинен бути метод `Dispose`, який слід викликати для звільнення ресурсів, пов'язаних з перевіркою ліцензії.

Ось приклад ліцензується елемента керування `Widget`, що використовує перевірку за допомогою класу `LieFileLicenseProvider`.

```
// Цей приклад припускає наявність оператора Imports System.ComponentModel
// Це атрибут LicenseProvider, який вказує тип вживаного
//LicenseProvider.
[LicenseProvider (typeof (LieFileLicenseProvider))]
public class Widget: System.Windows.Forms.Control
{
    private License myLicense;
    public Widget ()
    {
        // Перевірити ліцензію та отримати посилання на неї.
        myLicense = LicenseManager.Validate (typeof (Widget), this);
        // Інший код конструктора опущений.
    }
    // Це метод Dispose, що звільняє ресурси,
    // Пов'язані з перевіркою ліцензії.
    protected override void Dispose (bool Disposing)
    {
        if (myLicense != null)
        {
            myLicense.Dispose ();
            myLicense = null;
        }
    }
}
```

Реалізація керування ліцензуванням елемента керування:

- вкажіть клас `LicenseProvider`, додавши до оголошення класу атрибут `LicenseProviderAttribute`;

- створіть файл `License`, необхідний обраним класом `LicenseProvider`;

- у конструкторі елемента керування викличте метод `LicenseManager.Validate` для перевірки ліцензії;

- звільніть ресурси, зайняті перевіркою ліцензії, викликавши метод `Dispose` елемента керування.

Завантаження елемента керування в Internet Explorer

Будь-який елемент керування Windows Forms можна завантажити в Internet Explorer. Це дозволяє збагачувати HTML-сторінки широкими можливостями таких елементів керування. З цього розділу ви дізнаєтеся, як це виконати.

Щоб завантажити елемент керування в Internet Explorer, він повинен мати у своєму розпорядженні в кеші глобальних збірок (Global Assembly Cache, GAC) або в одному віртуальному каталозі з HTML-сторінкою, в якій він оголошений. Для оголошення елементів керування Windows Forms в HTML-сторінках служить тег <OBJECT>, який наказує вставити в сторінку скомпільований об'єкт. Тег <OBJECT> визначає тип завантажуваного об'єкта по його властивості classid, тому можна задати завантаження елемент керування Windows Forms через цю властивість.

Властивість classid елемента керування Windows Forms складається з двох частин: шляху до файлу елемента керування та його повного імені, розділених знаками «#».

Нижче показаний приклад тега <OBJECT>, оголошує елемент керування з повним ім'ям ControlLibrary1.myControl, який з файлу ControlLibrary1.dll, який розташований в одному віртуальному каталозі з HTML-сторінкою:

```
<OBJECT id = "myControl"
classid = "http:ControlLibrary1.dll#ControlLibrary1.myControl"
VIEWASTEXT>
</ OBJECT>
```

Завантаження елемента керування в Internet Explorer:

- запишіть DLL-файл елемента керування у віртуальний каталог HTML-сторінки, в якій він оголошений, або встановіть його в GAC;
- додайте до HTML-сторінці тег <OBJECT> з відповідним значенням classid. Властивість classid елемента керування включає шлях до його DLL-файлу і повне ім'я, розділені знаком «#».

4.3 Завдання до роботи

4.3.1 Ознайомитися з основними теоретичними відомостями за темою роботи, використовуючи ці методичні вказівки, а також рекомендовану літературу.

4.3.2 Створити програму, для перегляду та роздрукування зображень, створити власні елементи керування.

4.3.3 Оформити звіт з роботи.

4.3.4 Відповісти на контрольні запитання.

4.4 Зміст звіту

4.4.1 Тема та мета роботи.

4.4.2 Завдання до роботи.

4.4.3 Короткі теоретичні відомості.

4.4.4 Код програми.

4.4.5 Висновки, що містять відповіді на контрольні запитання (5 шт. за вибором студента), а також відображують результати виконання роботи та їх критичний аналіз.

4.5 Контрольні питання

4.5.1 Які, як правило, компоненти використовуються для реалізації друку?

4.5.2 Подія `PrintPage`, її властивості.

4.5.3 Як відбувається друк графіки?

4.5.4 Як відбувається друк тексту?

4.5.5 Елемент `PrintPreviewControl`, призначення та застосування.

4.5.6 Елемент `PrintPreviewDialog`, призначення та застосування.

4.5.7 Елемент `PageSettings`, призначення та конфігурування.

4.5.8 Елементи `PageSetupDialog` та `PrintDialog`, призначення та застосування.

4.5.9 Що таке GDI+ та його призначення?

4.5.10 Яке призначення бібліотеки `System.Drawing`?

4.5.11 Об'єкт `Graphics`, методи для малювання контурів та зафарбовування програм.

4.5.12 Об'єкти `Color`, `Brush` і `Pen` призначення, застосування і типи кистей.

4.5.13 Назвіть п'ять методів, щоб додати фігуру до об'єкта `GraphicsPath`.

4.5.14 Які вам відомі принципи створення елементів керування?

4.5.15 Як намалювати просту фігуру?

4.5.16 Як відбувається спадкування від класу `UserControl`?

4.5.17 Як відбувається спадкування від класу `Control`?

4.5.18 Як додавати створені компоненти на панель `Toolbox`?

4.5.19 Як присвоїти елементу керування значок?

4.5.20 Як керувати ліцензуванням елемента керування?

4.5.21 Як завантажити елемент керування в Internet Explorer?

ЛІТЕРАТУРА

1. Соммервил И. Инженерия программного обеспечения / Соммервил И. – М. : Издательский дом «Вильямс», 2002. – 623 с.
2. Pfleeger S. L. Software Engineering. Theory and practice / S. L. Pfleeger. – New Jersey : Printice Hall, 1998. – 576 p.
3. Рекомендации по преподаванию программной инженерии и информатики в университетах = Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering; Computing Curricula 2001: Computer Science: пер. с англ. – М. : ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», 2007. – 462 с.
4. Guide to the Software Engineering Body of Knowledge (SWEBOOK). – New York : IEEE Publising House, 2004. – 129 p.
5. Орлов С. А. Технологии разработки программного обеспечения. Учебник для Вузов / С. А. Орлов. СПб. : Питер. – 2002. – 463 с.
6. Васильев А. Microsoft Office 2007 / А. Васильев. – СПб. : Питер, 2007. – 160 с.
7. Кузьменко В. Г. Программирование на VBA 2002 / В. Г. Кузьменко. – М.: ООО «Бином-пресс», 2003. – 880 с.
8. Лабор В.В. Си Шарп Создание приложений для Windows / В. В. Лабор. – Минск : Харвест, 2003. – 385 с.
9. Джонсон Б. Основы Microsoft Visual Studio .Net 2003 / Б. Джонсон. – Москва : Русская редакция, 2003. – 463 с.
10. Бишоп Д. С# в кратком изложении / Д. Бишоп. – Москва : БИНОМ Лаборатория знаний, 2005. – 467 с.