БЕЗМЕНОВ Микола Іванович,
кандидат технічних наук, доцент, професор кафедри системного аналізу та інформаційно-аналітичних технологій Національного технічного університету «Харківський політехнічний інститут»; автор шести навчальних посібників, з них чотирьох з грифом МОН України і одного з грифом НТУ «ХПІ»; автор одного посібника, виданого за кордоном

БЕЗМЕНОВА Ольга Миколаївна,
магістр системного аналізу і управління, закінчила аспірантуру при кафедрі програмної інженерії та інформаційних технологій управління Національного технічного університету «Харківський політехнічний інститут», системний аналітик IT-компанії «Cloud Works»; співавтор двох навчальних посібників, з них одного з грифом НТУ «ХПІ»

КАЛІНІН Денис Вікторович,
магістр інформатики, закінчив аспірантуру при кафедрі системного аналізу та інформаційно-аналітичних технологій Національного технічного університету «Харківський політехнічний інститут», менеджер з розробки програмного забезпечення IT-компанії «MedeAnalitics, Inc.»; співавтор одного навчального посібника з грифом НТУ «ХПІ»

ISBN 978-617-8113-28-5

М. І. Безменов
О. М. Безменова
Д. В. Калінін

ОСНОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ МОВОЮ C#

М. І. Безменов
О. М. Безменова
Д. В. Калінін

ОСНОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ МОВОЮ C#

НАВЧАЛЬНИЙ ПОСІБНИК

Харків 2023

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

М. І. Безменов
О. М. Безменова
Д. В. Калінін

ОСНОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ МОВОЮ C#

Навчальний посібник для студентів навчально-наукового інституту комп’ютерних наук та інформаційних технологій

Рекомендовано вченою радою НТУ «ХПІ»

Харків
2023

УДК 004.432(075)
Б 39

Р е ц е н з е н т и :

М. В. Ткачук, д-р техн. наук, проф., Харківський національний університет імені В. Н. Каразіна;

С. Ф. Чалий, д-р техн. наук, проф., Харківський Національний технічний університет радіоелектроніки

Рекомендовано вченою радою НТУ «ХПІ»
як навчальний посібник для студентів
навчально-наукового інституту комп'ютерних наук
та інформаційних технологій,
протокол № 10 від 20.12.2022

Безменов М. І.

Б 39 Основи візуального програмування мовою C# : навч. посіб. для студентів навчально-наукового інституту комп'ютерних наук та інформаційних технологій / М. І. Безменов, О. М. Безменова, Д. В. Калінін. – Харків : ФОП Панов А. М., 2023. – 648 с.

ISBN 978-617-8113-28-5

Описано можливості однієї з найпопулярніших мов програмування – C#. Наведено достатньо повний опис базових елементів C#, інтегрованого середовища розробника, а також основних компонентів. Подано стислу історію розвитку обчислювальної техніки та мов програмування. Викладання супроводжується великою кількістю прикладів програм, а також контрольними запитаннями й завданнями для практичного відпрацьовування матеріалу.

Призначено для студентів навчально-наукового інституту комп'ютерних наук, зокрема, тих, що навчаються за спеціальністю «Видавництво та поліграфія». Посібник може використовуватися студентами вищих навчальних закладів, що навчаються на спеціальностях галузі знань «Інформаційні технології» та поріднених з нею.

Іл. 60. Табл. 9. Бібліогр.: 33 назви.

УДК 004.432(075)

ISBN 978-617-8113-28-5

ЗМІСТ

ПЕРЕДМОВА

Видання даного навчального посібника обумовлено змінами у наповненості навчальних програм на кафедрі системного аналізу та інформаційно-аналітичних технологій Національного технічного університету «Харківський політехнічний інститут». За час існування кафедри ази програмування її студенти починали освоювати, орієнтуючись на мови програмування, які були більш менш сучасними у 80–90-ті роки минулого і на початку цього сторіччя. Це спочатку була мова Turbo Pascal 7.0, яку замінили мова Delphi. Відповідно до цього були випущені навчальні посібники з цих мов програмування (а саме, «Турбо Паскаль 7.0» та «Основи програмування в середовищі Delphi», автором яких є один з авторів даного посібника), що отримали грифи Міністерства освіти і науки України. На цей час базовими у підготовці студентів кафедри є мови C та C++. Слід відзначити, що вже досить давно у навчальному процесі кафедри використовується ще одна мова програмування, а саме мова C#, синтаксис якої дуже схожий із синтаксисом мови C++. Оскільки підготовка студентів у цьому напрямку орієнтована на візуальне програмування, а виданих державною мовою посібників з візуального програмуваннямовою C# явно недостатньо, виникла необхідність у даному виданні.

Структура посібника багато в чому повторює структуру іншого раніше виданого посібника «Основи програмування в середовищі Delphi», а сам він частково запозичує матеріал його першого розділу, присвяченого основним відомостям з історії розвитку обчислювальної техніки та мов програмування. При написанні першого розділу даного навчального посібника було також використано матеріал ще одного навчального посібника з грифом МОН України – «Вступ до інформатики» (автор М. І. Безменов).

Даний посібник містить матеріал, орієнтований на вивчення та освоєння як базових для мови C# елементів, так і основних компонентів, що найчастіше використовуються при написанні нескладних

програм. Викладання матеріалу супроводжується прикладами програм, що пройшли налаштування і тестування.

Варто враховувати, що програмні коди не претендують на те, щоб їх можна було назвати як найкращими, – вони лише ілюструють можливості окремих інструкцій і компонентів мови C#.

Контрольні запитання і завдання для програмування, які вміщені після кожного розділу, дозволяють організувати самотестування і практичне відпрацьовування розглянутого матеріалу. При необхідності в збільшенні кількості задач для програмування можна скористатися ще одним навчальним посібником з грифом МОН України – «Збірник задач із програмування» (автор М. І. Безменов).

Розділи посібника, що присвячені розгляду компонентів C#, можуть бути використані як довідковий матеріал.

Для освоєння матеріалу не потрібні жодних особливих додаткових знань, за винятком хіба що початкових навичок у роботі з файлами в операційній системі Windows.

Виражаємо вдячність рецензентам за зауваження, що дозволили внести зміни в текст посібника, значно покращивши викладання матеріалу.

1. КОМП'ЮТЕР І ЙОГО СТИСЛА ІСТОРІЯ

1.1. Початкові відомості про апаратне і програмне забезпечення комп'ютерів

Комп'ютери в сучасному світі проникнули в усі сфери діяльності людини. Насамперед, звичайно, сказане стосується наукової діяльності. Це передбачав вже розробник першого обчислювального пристрою, який можна було програмувати, – Чарльз Беббідж. Ось його слова: «Тепер є можливість виконувати весь процес розробки і аналізу за допомогою машин. З появою Аналітичної Машини стало ясно, що вона неминуче стане фактором, що визначає шляхи розвитку науки».

Функціонування комп'ютерів засноване на такому понятті, як програма.

Програмою називається набір інструкцій, які призначені для їхнього виконання комп'ютером.

Множина фізичних складових комп'ютера називається його ***апаратним забезпеченням*** (hardware), а множина програм, використовуваних ним, – ***програмним забезпеченням*** (software).

При розгляді будови комп'ютера найчастіше виділяють п'ять основних складових: пристрій введення, пристрій виведення, центральний процесор (central processing unit – CPU), основну (оперативну) пам'ять і вторинну (зовнішню, допоміжну) пам'ять (рис. 1.1).

Центральний процесор – це пристрій, призначений для виконання різного роду арифметичних і логічних операцій і видачі сигналів керування іншими пристроями комп'ютера.

Основна пам'ять – це пристрій, що служить для оперативного (поточного) зберігання інформації. Ця пам'ять є енергозалежною –

собою 6-розрядний десятковий підсумовувальний пристрій, який у подальшому був удосконалений з доведенням кількості розрядів до восьми.

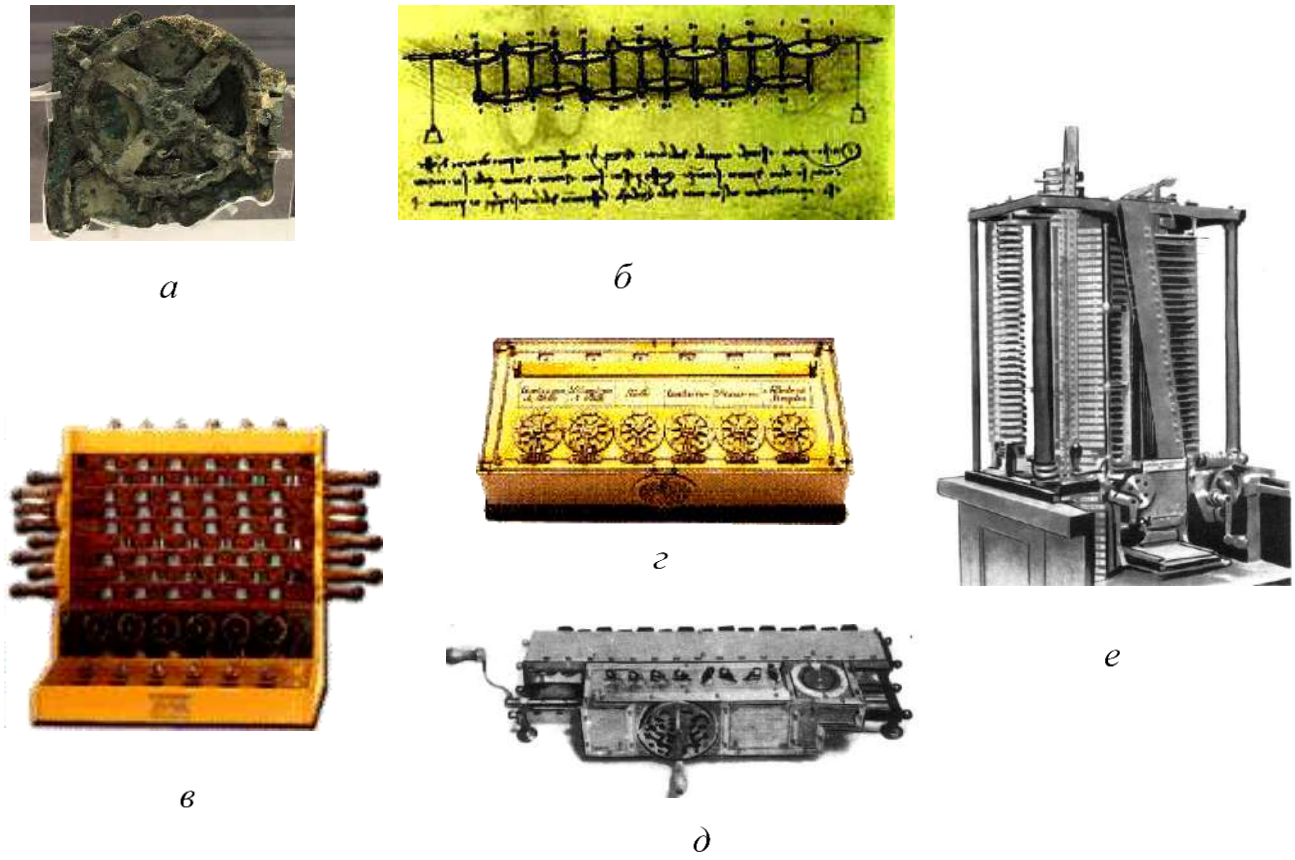


Рис. 1.4. Перші обчислювальні пристрої: *а* – фрагмент Антикітерського механізму; *б* – ескіз обчислювального пристрою Леонардо, *в* – машина Шіккарда, *г* – машина Паскаля, *д* – машина Ляйбніца, *е* – модель аналітичної машини Беббіджа

Якщо всі названі вище обчислювальні пристрої призначалися для роботи з десятковими числами, то першим недесятковим підсумовувальним пристроєм була винайдена в Англії сером Семюелом Морлендом недесяткова машина, призначена для роботи з англійською валютою (1667 рік).

У 1673 році Готфрід Вільгельм Ляйбніц створив свій «арифметичний прилад», що міг виконувати всі чотири арифметичні операції над дванадцятирозрядними десятковими числами з результатом до 16 знаків (див. рис. 1.4, *д*). Ляйбніц же займався дослідженням властивостей ***двійкової системи числення***, що надалі стала базовою системою числення в обчислювальній техніці.

Додайте в міксер 1 яйце.
Додайте в міксер 1 чашку молока.
Якщо ви не за кермом, додайте 30 мл рому.
Додайте екстракт ванілі за смаком.
Збийте до одержання однорідної маси.
Вилийте суміш у красивий фужер.
Посипте мускатним горіхом.

12. З чого необхідно починати створення програми?
13. На які фази можна розбити процес створення програми?
14. Перелічите та охарактеризуйте три основні види помилок програмування.
15. Які види помилок виявляються компілятором?
16. До якого виду належить помилка, що полягає у пропущенні в програмі кінцевої закриваючої круглої дужки?
17. Чи потрібно реагувати на попереджувальні повідомлення компілятора або їх можна ігнорувати? Поясніть відповідь.
18. Припустімо, існує програма, призначена для обчислення банківських відсотків, і в ній допущена помилка у розмірі нараховуваних відсотків. Якого виду ця помилка?

    Дайте характеристику помилок часу виконання.
19. Що таке тестування програми?
20. У чому основна заслуга Чарльза Беббіджа та Ади Лавлейс?
21. Хто є основоположником алгебри логіки?
22. Назвіть розроблювачів перших релейних машин.
23. Яку концепцію заклав у створення обчислювальних машин Джон Атанасов?
24. У чому полягає основний принцип архітектури фон Неймана?
25. Назвіть перші електронні обчислювальні машини.
26. Яка алгоритмічна мова визнається першою мовою програмування високого рівня.
27. У чому особливості процедурного, структурного та об’єктно-орієнтованого програмування?

2. ПОЧАТКИ ПРОГРАМУВАННЯ В СЕРЕДОВИЩІ C#

2.1. Найпростіший приклад розв'язання задачі

Розгляд матеріалу почнемо з найпростішого прикладу, який ілюструє думку, що якщо відомий шлях розв'язання задачі, то написання програми – не така вже й складна справа (приклад 2.1).

```
// Приклад 2.1.
// Дано ціле число. Скільки цифр входить у його десятковий
// запис?
```

Послідовність дій, призначена для розв'язання цієї задачі, може виглядати, наприклад, так:

1. Отримати ціле число `n`.
2. Надати деякому лічильнику `count`, призначеному для підрахунку кількості цифр, значення `0` (це означає те, що поки число `n` ще не опрацьовувалося).
3. Збільшити `count` на 1.
4. Відітнути від числа `n` останню цифру (розділити `n` на `10`, взявши як результат цілу частину від ділення).
5. Якщо значенням `n` не є `0`, повернутися до п. 3.
6. Вивести значення `count`.

Відкидаючи введення числа `n` (п. 1) і виведення числа `count` (п. 6), а також дії, пов'язані з задаванням характеристик змінних, які використовуються в програмі, для реалізації цієї послідовності дій може бути застосовано такий фрагмент програми мовою C#:

```
count = 0;                  // Покласти count = 0.
do                          // Повторювати.
```

```
{
    count++;              // Збільшення count на 1.
    n = n / 10;           // n – ціла частина від ділення n на 10.
} while (n != 0);         // Поки n не дорівнюватиме нулю.
```

Як це видно із запропонованого тексту, якщо послідовність дій з перетворення даних у шуканий результат (інакше – ***алгоритм***) визначена цілком конкретно, то написання програми близьке до формального перекладу цієї послідовності дій на мову програмування.

У програмі повинні бути задані властивості використовуваних змінних `n` і `count`, для чого її можна доповнити, наприклад, такою інструкцією:

```
int n, count;
```

Ця інструкція вказує, що змінні `n` і `count` можуть набувати тільки цілих значень (можливо, від'ємних).

Крім того, необхідно виконати оформлення програми за правилами мови C# і доповнити її засобами введення та виведення даних (у даному посібнику з використанням візуальних компонентів).

2.2. Два етапи розробки програми

Розробка складних програмних продуктів, оформлених відповідно до сучасних вимог, практично неможлива без використання інструментальних засобів швидкої розробки програм (Rapid Application Development, RAD), першим з яких був Visual Basic, створений корпорацією Microsoft. Одним з таких засобів є система програмування Microsoft Visual Studio 2019, призначена для розробки програм із можливим використанням середовища візуального проєктування.

Як і в будь-якому іншому середовищі візуального програмування, процес розробки програми складається з двох етапів, перший з яких орієнтований на роботу з візуальними конструкторами, а другий – на роботу з програмним кодом. Характерним при цьому є те, що в процесі виконання другого етапу (безпосереднього програмування) завжди можна повернутися до першого з метою корекції проєкту та наступним продовженням написання програмного коду. Ця особливість забезпечує гнучкість процесу розробки програми.

Візуальним відображенням працюючої програми є форма, яку і розробляє програміст, починаючи проєктування програми. Спеціальне вікно форми створюється автоматично, як тільки програміст у візуальному середовищі вибере команду Windows Forms Application (Застосунок Windows Forms) у розділі мови програмування Visual C#|Windows. При цьому середовище автоматично створює оформлені за певними правилами спеціальні файли.

Перший етап проєктування програми полягає в наповненні вікна форми елементами керування, які називаються *компонентами*. Такими елементами є різні кнопки, прапорці, списки, вікна редакторів тощо. Програміст оперує компонентами, маючи їхнє візуальне відображення. Можна сказати, що він просто бере їх з так званої палітри компонентів і розміщає на формі.

Якщо програміст поміщає на форму який-небудь компонент, то середовище C# автоматично робить відповідні зміни, зокрема вносячи посилання на доданий компонент. При цьому навіть створеній автоматично порожній формі відповідає програма, що може бути виконана після компіляції.

Компоненти фактично є програмними заготівками для майбутньої програми, оскільки вони містять у собі як програмний код, так і дані, необхідні для її функціонування. Тому і програма в цілому, і розміщені у вікні форми компоненти є працездатними навіть без зміни автоматично створеного програмного коду. Якщо автоматично встановлені налаштування яких-небудь із розміщених на формі компонентів не влаштовують програміста, він може внести зміни без звертання до програмного коду. Для цього в C# передбачено засоби, що дозволяють змінювати характеристики компонентів за допомогою миші або клавіатури вже на етапі проєктування.

На другому етапі розробки програми програміст повинен наповнити компоненти програмним кодом, призначеним для розв'язання поставленої задачі. При цьому, природно, йому багато в чому доводиться орієнтуватися на традиційну техніку програмування. У той же час вживання компонентів істотно змінює техніку програмування: сучасне програмування базується на об'єктно-орієнтованому підході з використанням таких понять, як класи, властивості, методи та події. Установлені при проєктуванні форми початкові налаштування компо-

нентів можуть змінюватися програмно в процесі виконання застосунку. Більш того, у процесі виконання програми компоненти можуть програмно створюватися та знищуватися.

У процесі написання програмного коду програміст завжди може повернутися до етапу проєктування, здійснюючи модифікацію форми в плані не тільки її візуального відображення, але і наповненості компонентами.

Було б дивно, якби в системі програмування C# не було передбачено засоби налаштування програм. Достатньо потужні засоби налаштування дозволяють виконувати програму за інструкціями, стежачи за ходом виконання по тексту і контролюючи при цьому поточні значення змінних. Можлива також установка точок припинення (переривання), при досягненні яких програма автоматично переривається свою роботу, переходячи в налаштовувальний режим.

2.3. Початкові відомості про середовище розроблювача Visual Studio 2019

2.3.1. Стартова сторінка і початок роботи

Завантаження Visual Studio 2019 починається з того, що на екран здійснюється виведення стартової сторінки, зображеної на рис. 2.1.

Зокрема, за допомогою стартової сторінки забезпечується швидкий доступ до недавніх проєктів, створення нового і відкриття існуючого проєкту.

Для створення нового проєкту необхідно виконати команду `Create a new project` (`Створення нового проєкту`), у результаті чого відкриється відповідне вікно (рис. 2.2).

Вибір шаблону `Windows Forms App (.NET Framework) C#` і натискання кнопки `Next` (`Далі`) – рис. 2.2 – забезпечує відкриття вікна `Configure your new project` (`Конфігурування нового проєкту`), де можна, наприклад, задати ім’я проєкту (`Project Name`) та вказати його місцезнаходження (`Location`); рис. 2.3. Після натискання в цьому вікні кнопки `Створити` (`Create`) створюється початкова форма і відкривається інтегроване середовище розроблювача (ІСР) із завантаженою у відповідне вікно початковою формою (рис. 2.4).

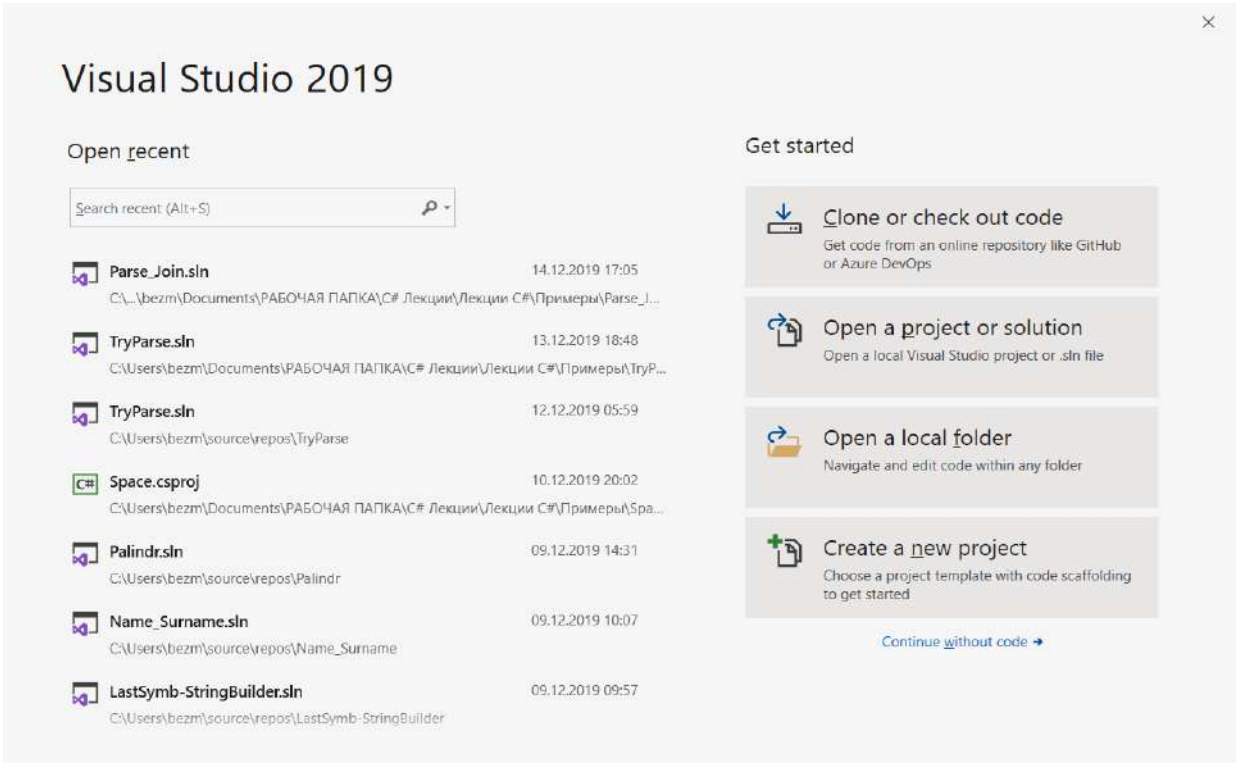


Рис. 2.1. Стартова сторінка

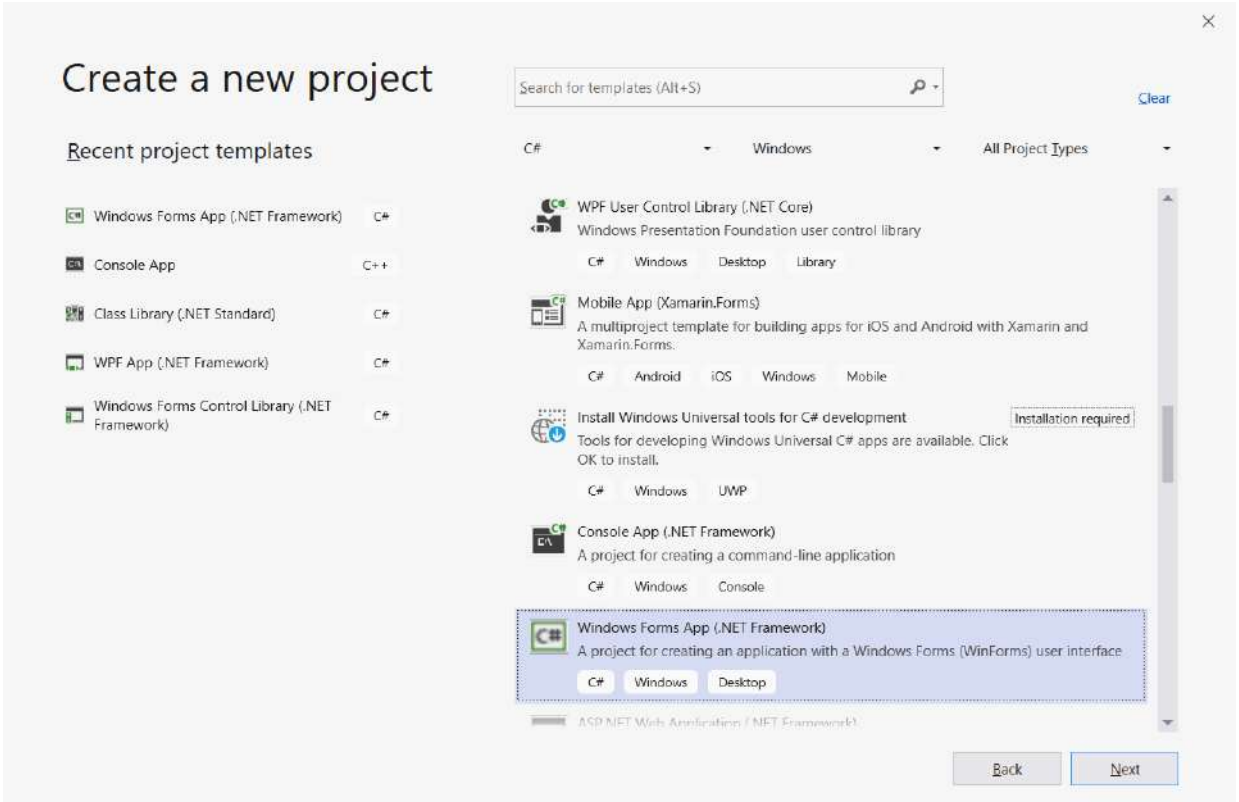


Рис. 2.2. Вікно відкриття нового проєкту

іменами опрацьовувачей подій, яка відкривається у вікні Properties після кліку мишею над кнопкою  . Далі може бути видалений код опрацьовувача події, оскільки при такому методі реєстрація події буде анульована автоматично.

Запитання для контролю і самоконтролю

1. Охарактеризуйте етапи розробки програм.
2. Які дії можна виконати за допомогою стартової сторінки Visual Studio 2019?
3. Дайте характеристику головного вікна.
4. Що таке «вікно форми»?
5. У які вікна завантажується програмний код, створюваний при розробці візуального застосунку?
6. Для чого використовується вікно палітри інструментів?
7. Дайте стислу характеристику оглядача рішень та його призначення.
8. Що міститься у вікні властивостей?
9. Опишіть методику видалення непотрібного опрацьовувача події.

Завдання для практичного відпрацьовування матеріалу

1. Створіть форму і розташуйте на ній кнопку і панель. Виконайте такі дії:
   а) перемістіть кнопку на панель;
   б) здійсніть зворотне перенесення кнопки.
2. Створіть новий застосунок, що містить порожню форму. Перегляньте текст проєкту.
3. Для прикладу 2.2 завантажте у вікно коду текст проєкту, після чого здійсніть перехід до тексту модуля. Закрийте сторінку з текстом проєкту.
4. Створіть новий застосунок, що містить форму з компонентом `TextBox`, який має встановлені за умовчанням властивості, і запустіть програму. Клацніть лівою кнопкою миші над компонентом `TextBox` і переконайтеся, що в його вікні можна виконувати набір тексту. Наберіть рядок, що виходить за межі вікна. Переконайтеся в

тому, що в даному випадку є можливість переглядати і змінювати текст, але ввести декілька рядків неможливо.

5. Розмістить на формі компонент `TextBox` і за допомогою вікна Properties надайте його властивості `Multiline` значення **`true`**. Переконайтесь, що у такого компонента можна змінювати вертикальний розмір, захопивши мишею його нижню межу і розтягуючи його. Переконайтесь, що в такий компонент можна записати декілька рядків тексту. Наберіть текст, деякі з рядків якого виходять за межі вікна. Переконайтеся в тому, що в даному випадку є можливість переглядати і змінювати текст, навіть якщо він виходить за межі вікна. Що відбудеться, якщо вертикальний розмір вікна не дозволяє помістити всі рядки тексту?
6. Виконайте попереднє завдання, змінивши значення властивості `ScrollBars` на `Both`, а властивості `WordWap` на **`false`**.
7. Відкрийте проєкт для прикладу 2.2 і перевірте, як зміниться робота програми, якщо в остаточний варіант тексту модуля в опрацьовувач події `Click` кнопки `button1` видалити рядки

```
textBox1.Visible = false;
button1.Enabled = false;
```

8. Подивіться, як зміниться виконання застосунку, якщо в попередньому завданні додатково вписати рядок

```
button1.Width = 2 * button1.Width;
```

9. Для форми з прикладу 2.2 (див. рис. 2.8) виконайте вставку панелі, розміри якої достатні для розміщення всіх наявних на формі компонентів, виділіть мишею ці компоненти і перетягніть на панель. За допомогою вікна Properties виконайте налаштування панелі так, щоб вона розмістилася у верхній частині форми, розтягшись на всю ширину останньої. Подивіться, як при цьому зміниться розташування інших компонентів.
10. Здійсніть опрацювання описаних методів видалення непотрібних опрацьовувачів подій.

3. БАЗОВІ ЕЛЕМЕНТИ МОВИ C#

3.1. Алфавіт мови

Символи мови C# – літери, арабські цифри і спеціальні символи – становлять його алфавіт.

У мові C# використовуються великі і малі ***літери*** латинського алфавіту від `a` до `z` та від `A` до `Z`, а також символ підкреслення (`_`). При цьому великі та малі літери вважаються різними.

Цифри – це арабські цифри від `0` до `9`, а також ***шістнадцяткові цифри***, в яких перші десять значень позначають цифрами від `0` до `9`, а решту шість – латинськими літерами від `A` до `F` або від `a` до `f`.

Спеціальні символи – це `+ - * / = < > % & ^ ~ . , ( ) [ ] { } : ; ' " $ @ #`

Є також складені спеціальні символи. До них відносять такі ***пари*** символів, як `!= <= >= == :: /* */ // || && ++ -- *= /= %= += -= &= ^= |= => << >>`, а також ***трійки*** символів `<<=` й `>>=`.

Рекомендується активно використовувати пробіли, символи табуляції, а також перехід до нового рядка для написання програм, які будуть легко сприйматися людиною. Вживання пробільних символів у тексті програми (за винятком рядкових констант) не позначається на результатах її компіляції та виконанні.

3.2. Ідентифікатори та змінні

У будь-якій мові, у тому числі в мовах програмування, виділяють таке поняття як лексема. У інформатиці ***лексема*** – це послідовність припустимих символів мови програмування, яка має сенс для компіля-

д) $\dfrac{\cos 1}{\sin 1}+\dfrac{\cos 1+\cos 2}{\sin 1+\sin 2}+\ldots+\dfrac{\cos 1+\ldots+\cos n}{\sin 1+\ldots+\sin n}$.

9. Дано дійсне число x і натуральне число n. Обчислити:

а) x^n; *б*) $x(x+1)\ldots(x+n-1)$; *в*) $x(x-n)(x-2n)\ldots(x-n^2)$;

г) $\dfrac{1}{x^2}+\dfrac{1}{x^2}+\dfrac{1}{x^4}+\ldots+\dfrac{1}{x^{2^n}}$; *д*) $\dfrac{1}{x}+\dfrac{1}{x(x+1)}+\ldots+\dfrac{1}{x(x+1)...(x+n)}$.

10. Дано натуральне число n та дійсне число x. Обчислити:

а) $\sin x+\sin^2 x+\ldots+\sin^n x$; *б*) $\sin x+\sin x^2+\ldots+\sin x^n$;

в) $\sin x+\sin\sin x+\ldots+\underbrace{\sin\sin\ldots\sin}_{n} x$.

11. Дано дійсне число a. Знайти:

а) серед чисел 1, $1+\dfrac{1}{2}$, $1+\dfrac{1}{2}+\dfrac{1}{3},\ldots$ перше значення більше за a;

б) таке найменше n, що $1+\dfrac{1}{2}+\ldots+\dfrac{1}{n}>a$.

12. Дано дійсні числа a, h і натуральне число n. Обчислити $f(a)+2f(a+h)+2f(a+2h)+\ldots+2f(a+(n-1)h)+f(a+nh)$, де $f(x)=(x^2+1)\cos^2 x$.

13. Дано ціле число n.

а) Скільки цифр у числі n?

б) Чому дорівнює сума його цифр?

в) Знайти першу цифру числа n.

г) Одержати нове число, приписавши по одиниці в початок та у кінець запису числа n.

14. Дано ціле число n. Знайти знакопереміжну суму цифр числа n (якщо запис n у десятковій системі є $\alpha_k\alpha_{k-1}\ldots\alpha_0$, то знайти $\alpha_k-\alpha_{k-1}+\ldots+(-1)^k\alpha_0$).

15. Дано натуральні числа n і m. Знайти

а) їх найбільший спільний дільник;

б) їх найменше спільне кратне.

16. Нехай $v_1 = v_2 = 0;\quad v_3 = 1.5;\quad v_i = \dfrac{i+1}{i^2+1} v_{i-1} - v_{i-2} v_{i-3},\quad i = 4,5,\ldots$ Дано натуральне число n ($n \ge 4$). Одержати v_n.
17. Нехай $x_0 = c;\ x_1 = d;\ x_k = qx_{k-1} + rx_{k-2} + b,\ k = 2,3,\ldots$ Дано дійсні числа q, r, b, c, d та натуральне число n ($n \ge 2$). Одержати x_n.
18. Нехай $u_1 = u_2 = 1;\ v_1 = v_2 = 1,$ а для $i = 3,4,\ldots$ значення u_i і v_i обчислюються по формулах $u_i = \dfrac{u_{i-1} - u_{i-2} v_{i-1} - v_{i-2}}{1 + u_{i-1}^2 + v_{i-1}^2};$ $v_i = \dfrac{u_{i-1} - v_{i-1}}{|u_{i-2} + v_{i-1}| + 2}$. Дано натуральне число n ($n \ge 3$). Одержати v_n.
19. Нехай $a_1 = b_1 = 1;\ a_k = \dfrac{1}{2}\left(\sqrt{b_{k-1}} + \dfrac{1}{2}\sqrt{a_{k-1}}\right);\ b_k = 2a^{2k-1} + b^{k-1},\ k = 2,3,\ldots$ Дано натуральне число n. Знайти $\sum\limits_{k=1}^{n} a_k b_k$.
20. Нехай $a_1 = b_1 = 1;\quad a_k = 3b_{k-1} + 2a_{k-1};\quad b_k = 2a_{k-1} + b_{k-1},\quad k = 2,3,\ldots$ Дано натуральне число n. Знайти
$$\sum_{k=1}^{n} \frac{2^k}{(1 + a_k^2 + b_k^2)k!}.$$
21. Обчислити нескінченну суму із заданою точністю ε ($\varepsilon > 0$). Уважати, що необхідна точність досягнута, якщо обчислено суму декількох перших доданків і черговий доданок виявився за модулем меншим, ніж ε (цей та всі наступні доданки можна вже не враховувати). Розглянути такі суми:

а) $\sum\limits_{i=1}^{\infty} \dfrac{(-1)^i}{i^2};$ *б)* $\sum\limits_{i=1}^{\infty} \dfrac{(-2)^i}{i!};$ *в)* $\sum\limits_{i=1}^{\infty} \dfrac{(-1)^{i+1}}{i(i+1)(i+2)};$ *г)* $\sum\limits_{i=0}^{\infty} \dfrac{1}{4^i + 5^{i+2}}.$

22. Нехай $y_0 = 0;\ y_k = \dfrac{y_{k-1} + 1}{y_{k-1} + 2},\ k = 1,2,\ldots$ Дано дійсне значення $\varepsilon > 0$. Знайти перший член y_n, для якого виконане співвідношення $y_1 - y_{n-1} < \varepsilon$.
23. Дано натуральне число n. Обчислити добуток перших n співмножників:

а) $\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{5}{6}\cdot\ldots$; *б*) $\frac{1}{1}\cdot\frac{3}{2}\cdot\frac{5}{3}\cdot\ldots$.

24. Дано натуральне число *n*. Одержати суму тих чисел виду $i^3 - \sin 2 + n$ $(i = 1,2,\ldots,n)$, які є потроєними непарними.
25. Дано натуральне число *n*, дійсне число *x*. Серед чисел $e^{\cos(x^{2k})}\sin(x^{3k})$ $(k = 1,2,\ldots,n)$ знайти найближче до якого-небудь цілого.
26. Для $x_1 = y_1 = 1;\ x_i = x_{i-1} + \frac{y_{i-1}}{i^2},\ y_i = y_{i-1} + \frac{x_{i-1}}{i},\ i = 2,3,\ldots$ одержати x_n, y_n, де *n* – задане натуральне число більше за 1.
27. Дано натуральне число *m*. Знайти всі прості дільники цього числа.
28. Дано натуральне число *n*. Одержати всі натуральні числа, що менші за *n* і не мають із ним спільних натуральних дільників, крім 1.
29. Дано натуральні числа *m* і *n*. Одержати $\frac{m!+n!}{(m+n)!}$.
30. Дано натуральні числа *m* і *n* ($m \le n$). Знайти натуральне число між *m* і *n* з найбільшою сумою дільників.
31. Знайти 100 перших простих чисел.
32. Дано натуральні числа *m* і *n* ($m \le n$). Знайти всі прості числа між *m* і *n*.
33. Знайти найменше натуральне число *n*, що може бути подане двома різними способами як сума кубів двох натуральних чисел: $n = x^3 + y^3$ ($x \le y$).
34. Дано натуральне число *n*. Вирахувати добуток $f_0 f_1 \ldots f_n$, де

$$f_i = \frac{1}{i^2+1} + \frac{1}{i^2+2} + \ldots + \frac{1}{i^2+i+1}.$$

35. Дано натуральне число *n*. Обчислити

а) $\sum_{k=1}^{n} k(k+1)\ldots k^2$; *б*) $\sum_{k=1}^{n} (-1)^{k+1}(3k^2-1)!$; *в*) $\sum_{k=1}^{n} k^k$; *г*) $\sum_{k=1}^{n} \frac{1}{(k^2)!}$.

Дано дійсне число *x* і натуральне число *n*. Обчислити

а) $\sum_{i=1}^{n} \frac{(2i)!+|x|}{(i^2)!}$; *б*) $\sum_{p=1}^{n}\sum_{q=p}^{n} \frac{x+p}{q}(p+q)!$; *в*) $\frac{1}{n!}\sum_{k=1}^{n} (-1)^k \frac{x^k}{(k!+1)!}$.

36. Припустімо, Ви хочете дістати позику на k місяців у розмірі n, грн. Умови кредиту такі: процентна ставка становить p, %, на рік від загальної суми кредиту, причому вся сума процентних виплат відраховується із загальної суми кредиту відразу ж у момент його отримання. Погашення позики здійснюється щомісяця рівними частками, виходячи з номінальної вартості кредиту. Яка номінальна сума позички повинна бути нарахована і яка сума повинна погашатися щомісяця? Так, при $n = 775$, $k = 18$, $p = 15$ номінальна сума кредиту становитиме $m = \frac{p}{100} \cdot n \cdot \frac{k}{12} = 1000$ грн, виплата при одержанні кредиту дорівнює $m \cdot p \cdot \frac{k}{12} = 225$ грн, а щомісячні виплати будуть $\frac{1000}{18} = 55.56$ грн. Для обчислення номінальної вартості кредиту використайте цикл замість наведеної вище формули.
37. Варто врахувати те, що можлива ситуація, коли початкова виплата дорівнюватиме або перевищуватиме номінальну вартість кредиту.
38. Розв’язати задачу 36 при уточненні: якщо на запропонованих умовах надання позики неможливе, повинен бути знайдений найбільший можливий термін кредитування.
39. Розв’язати задачу 36, уточнивши її у такий спосіб: якщо на запропонованих умовах надання позики неможливе, повинна бути знайдена найбільша можлива річна процентна ставка.
40. Нехай деяка покупка вартістю n, грн., зроблена в кредит із процентною ставкою p, %, на рік і відсутністю початкового внеску. Виплати проводяться щомісяця фіксованою сумою в m, грн, до якої входить виплата відсотка за кредит у даному місяці, причому щомісячні відсотки за кредит нараховуються, виходячи з річної процентної ставки та залишку боргу на момент виплати (сума виплати в останній місяць може виявитися меншою за m). Скільки місяців буде потрібно для погашення боргу. Яку суму становитиме плата за кредит? Урахувати те, що щомісячна виплата повинна перевищувати оплату відсотків за кредит.

5. МАСИВИ І РЯДКИ

5.1. Масиви

5.1.1. Одновимірні масиви

У C# існує механізм, що дуже просто розв'язує проблему позначення великої кількості однорідних елементів. При цьому можна не обмежуватися однією змінною, в яку по черзі заноситься множина значень, або оголошувати велику кількість різнойменних змінних. Альтернативою є використання ***масиву*** – колекції (послідовності) визначеної кількості однойменних елементів того самого типу, яка дозволяє звертатися до першого, другого і всіх наступних елементів за значенням їх індексів.

Для оголошення так званого одновимірного масиву потрібно після вказівки типу елементів масиву (*базового типу* масиву) перед ім'ям змінної зазначити квадратні дужки:

```
тип_елементів[] ім'я;
```

де ім'я – ім'я масиву.

Наприклад:

```
double[] a;
```

При цьому оголошується змінна-масив, що має тип тип_елементів[], але відведення пам'яті під масив (під його елементи) не виконується. Сама змінна-масив є посиланням, і вона повинна бути ініціалізована. Відведення пам'яті під масив (його ініціалізація) здійснюється створенням екземпляра масиву за допомогою оператора **new** з використанням такого формату:

```
ім'яМасиву = new тип_елементів[кількість_елементів];
```

```
    textBox1.Text = "Сума " + F(out sum, a) +
                    " доданків дорівнює " + sum + '.';
    // Послідовність чисел як аргумент - масив параметрів.
    textBox1.Text += "\r\nСума " + F(out sum, 1.1, 2.2, -3) +
                     " доданків дорівнює " + sum + '.';
    // Одно число (вираз) як аргумент - масив параметрів.
    textBox1.Text += "\r\nСума " + F(out sum, 2 * Math.PI) +
                     " доданків дорівнює " + sum + '.';
    // Відсутність аргументу - масиву параметрів.
    textBox1.Text += "\r\nСума " + F(out sum) +
                     " доданків дорівнює " + sum + '.';
}
```

Метод `F` має два параметра, першим з яких є **`out`**-параметр типу **`double`**, а другим – масив параметрів, елементи якого також мають тип **`double`**, причому масив параметрів у відповідності з вимогами компілятора зазначений останнім. При виклику методу з масивом параметрів відповідний фактичний параметр створюється автоматично з кількістю елементів, яка визначається аргументом, а аргументом тут є все, що перелічено після першого фактичного параметра, який є обов'язковим (у даному випадку він наявний, причому один). Оскільки всередині методу масив `arguments` є звичайним масивом, доступ до нього і його елементів не має ніяких особливостей (у даному випадку використаний цикл **`foreach`**). Метод `F` ілюструє також використання **`out`**-параметра і повернення результату за допомогою інструкції **`return`**.

В опрацьовувачі події `Click` кнопки `button1` метод `F` викликається чотири рази для різних варіантів аргументу – для масиву, для списку виразів (у даному випадку константи у літеральному поданні), для одного виразу і для випадку відсутності аргументу. У коді також показано, як виконується виклик методу з **`out`**-параметром.

6.4.3. Рекурсивні методи

Кожен з методів може викликáти інший метод, у тому числі звертатися до самого себе. Метод, що звертається до самого себе, називається ***рекурсивним***. Рекурсія може бути і неявною, коли метод `First` викликає метод `Second`, а той, у свою чергу, викликає метод `First`.

Обов'язковим елементом в описі будь-якого рекурсивного процесу є деяка інструкція, що визначає умову завершення рекурсії; іноді

вона називається *опорною умовою*. У цій умові може бути задане якенебудь фіксоване значення, що обов'язково буде досягнуте в ході рекурсивного обчислення, дозволяючи тим самим організувати своєчасне зупинення процесу. Крім того, повинен бути другий елемент – спосіб вираження одного кроку розв'язання за допомогою іншого, простішого. Кількість рівнів вкладеності при цьому не обмежена і може бути досить великою.

Кожен виклик рекурсивного методу повинен наближати випадок, що зупиняє рекурсивні виклики, інакше виконання методу ніколи не припиниться через нескінченний ланцюжок рекурсивних викликів. У дійсності нескінченний ланцюжок рекурсивних викликів не може реалізуватися, оскільки відбувається аварійне завершення програми. Найпростішим способом гарантування виконання опорної умови є зменшення деякої додатної величини до досягнення нею деякого значення.

Особливістю реалізації рекурсивних методів є те, що в програмі існує тільки один екземпляр коду цього методу. На кожному рівні рекурсії здійснюється нове звертання до цього коду.

На рис. 6.2 на прикладі деякого методу `F` ілюструє виконання рекурсивного процесу з трьома рівнями рекурсивного виклику; правда, для простоти пояснення **вважається**, що на кожному рівні створюється новий екземпляр коду методу. Розглянемо послідовність дій, що виконується в цьому випадку.

Виклик методу `F` забезпечує початок виконання його коду. Уважаючи, що на першому рівні рекурсії опорна умова не виконується, маємо виконання інструкцій, які йдуть за опорною умовою, і новий виклик методу. На цьому виконання методу на першому рівні тимчасово переривається, і починається другий рівень рекурсії.

Другий рівень рекурсії аналогічний першому. Як і на першому рівні, на ньому спочатку перевіряється опорна умова (на рис. 6.2 вважається, що вона не виконується), після чого здійснюється ще один виклик методу `F` самого себе. Далі дії цього рівня також тимчасово перериваються з переходом до третього рівня рекурсії.

Уважаючи, що на третьому рівні рекурсії опорна умова виконана, маємо завершення третього рівня з поверненням на другий рівень у точку, де здійснилося тимчасове переривання його виконання. Далі завершуються обчислення на другому рівні рекурсії з наступним поверненням на перший рівень.

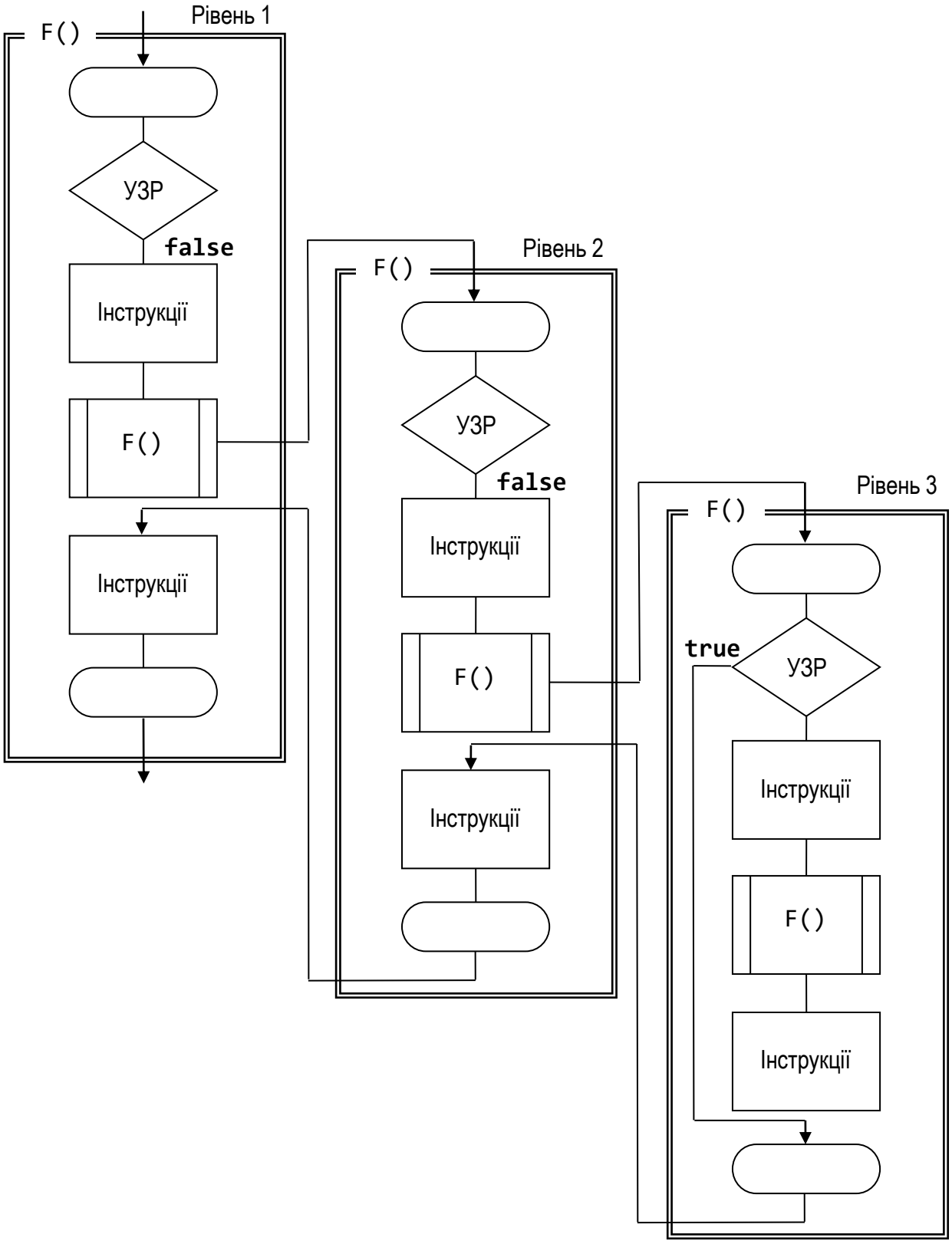


Рис. 6.2. Схема виконання рекурсивного процесу
(УЗР – умова завершення рекурсії)

Після завершення обчислень на першому рівні отримуємо остаточний результат з виходом з першого рівня рекурсивних викликів.

Звичайно, у випадку виконання опорної умови також можуть виконуватися деякі інструкції.

```
// Приклад 6.5.
// Рекурсивний метод обчислення факторіала:
// 0! = 1; k! = k * (k - 1)!.
```

Скористаємося формою з кнопкою `button1` та багаторядковим текстовим полем `textBox1` і визначимо в класі форми такі методи:

```
private void button1_Click(object sender, EventArgs e)
{
    int n = Convert.ToInt32(textBox1.Text);
    textBox1.Text = n + "! = " + RecursiveFactorial(n) + '.';
}

int RecursiveFactorial(int k)
{
    if (k == 0)                              // Опорна умова.
        return 1;                            // Якщо вона виконана.
    return k * RecursiveFactorial(k - 1);    // Якщо не виконана.
}
```

У наведеному тексті метод `RecursiveFactorial` після першого його виклику в методі `button1_Click` багаторазово звертається сам до себе зі значенням параметра `k`, яке зменшується від заданого при першому виклику початкового значення `n` до `0`. При досягненні параметром значення `0` виконується опорна умова і здійснюється вихід з методу. При цьому відбувається повернення в точку виклику методу на попередньому рівні з продовженням обчислення значення виразу `k * RecursiveFactorial(k - 1)` і наступним виходом з методу з поверненням на попередній рівень.

Рекурсивним може бути й **`void`**-метод.

```
// Приклад 6.6.
// Визначити множення цілих чисел через додавання:
// x * 0 = 0; x * y = x + x(y - 1) за умови, що y > 0.
```

Для тестування методу множення розмістимо на формі панель `panel1` з кнопкою `button1` і двома текстовими полями `textBox1` та

textBox2, над якими розмістимо відповідно мітки label1 та label2, у властивості Text яких запишемо підказки Уведіть x (для label1) і Уведіть y (для label2). Крім того, розмістимо на формі текстове поле textBox3 для виведення результату (у властивість Visible запишемо значення **false**). Визначимо також у класі форми два поля:

```
int x, y;
```

Рекурсивний метод множення може бути таким:

```
void Multiplication(int x, int y, out int product)
{
    if (y == 0)                                  // Опорна умова.
        product = 0;
    else
    {                                            // Рекурсивний виклик.
        Multiplication(x, y - 1, out product);
        product += x;
    }
}
```

Визначимо його у класі форми, а для його тестування скористаємося таким опрацьовувачем події Click кнопки button1:

```
private void button1_Click(object sender, EventArgs e)
{
    x = Convert.ToInt32(textBox1.Text);
    y = Convert.ToInt32(textBox2.Text);
    if (y < 0)
    {
        x = -x;
        y = -y;
    }

    int product;
    Multiplication(x, y, out product);
    panel1.Visible = false;
    textBox3.Visible = true;
    textBox3.Text = x + " * " + y + " = " + product;
}
```

У методі Multiplication для правильної організації рекурсивного процесу доводиться нарощувати змінну z після виходу з попереднього рівня рекурсії.

```
    // конструктор базового класу.
    public Two(int x, double y) : base(x)
    {
        this.y = y; // Ліворуч поле, праворуч параметр.
        this.F1();        // Доступ до власного методу.
        // Інструкції_2.
    }
    protected new void F1() // Перекриваючий метод.
    {
        base.F1();     // Доступ до перекритого методу.
        // Інструкції_3.
    }
}

public class Three : Two    // Похідний клас.
{
    protected double z;
    // Конструктор, що викликає інший конструктор
    // свого класу.
    public Three(int x, double y, int n) : this(x, y)
    {
        z = n;
        // Інструкції_4.
    }
    // Перевантажений конструктор, що викликає
    // конструктор базового класу.
    public Three(int x, double y) : base(x, y)
    {
    }
}
```

Виклик будь-якого конструктора завжди починається з виклику базового конструктора, навіть якщо базовим класом є клас **`object`** зі своїм конструктором, що є конструктором за умовчанням. Тільки після виконання конструктора базового класу виконується тіло конструктора похідного класу. Виходячи з цього, для наведеного вище ланцюжка класів при виклику конструктора `Three(int x, double y, int n)` послідовність дій буде така:

1. Вхід у конструктор `Three(int x, double y, int n)`.

1.1. Виклик конструктора `Three(int x, double y)`.
1.1.1. Виклик конструктора `Two(int x, double y)`.
1.1.1.1. Виклик конструктора `One(int n)`.
1.1.1.1.1. Виклик конструктора `Object`, тобто конструктора класу, що є базовим для класу `One` і повернення в конструктор `One(int n)`.
1.1.1.1.2. Ініціалізація поля `n` (`n = x;`).
1.1.1.1.3. Виконання методу `F1`, визначеного в класі `One`, і повернення в тіло конструктора `Two(int x, double y)`.
1.1.1.2. Ініціалізація поля `y` (тобто `this.y`) значенням фактичного параметра, що відповідає формальному параметра `y` конструктора `Two(int x, double y)`.
1.1.1.3. Виклик власного для класу `Two` методу `F1`.
1.1.1.3.1. Виклик і виконання методу `F1` базового класу `One` і повернення в метод `F1` класу `Two`.
1.1.1.3.2. Виконання інструкцій, позначених як *Інструкції_3*, і повернення в тіло конструктора `Two(int x, double y)`.
1.1.1.4. Виконання інструкцій, позначених як *Інструкції_2*, і повернення в тіло конструктора `Three(int x, double y)`.
1.1.2. Виконання тіла конструктора `Three(int x, double y)` – у даному випадку порожнього – і повернення в тіло конструктора `Three(int x, double y, int n)`.
1.2. Ініціалізація поля `z` (тобто `z = n;`).
1.3. Виконання інструкцій, позначених як *Інструкції_4*.
2. Вихід з конструктора `Three(int x, double y, int n)`.

Якщо у визначенні класу перед словом **`class`** помістити ключове слово **`sealed`**, то такий клас оголошується як запечатаний. Особливістю запечатаних класів, є те, що вони не можуть використовуватися як базові, обриваючи ланцюжок наслідування.

Так, для наведеного вище ланцюжка з трьох класів, якщо у визначенні класу `Two` замість рядка

```
public class Two : One
```

записати рядок

```
public sealed class Two : One
```

визначити клас Three як похідний від класу Two буде неможливо.

Розглянемо приклад, в якому використовується спадкування при роботі з класами.

```
// Приклад 6.15.
// Створити таку ієрархію спадкування класів: рівень 0 – клас
// "Точка на площині" з приватними полями, які визначають
// координати точки, і методом виведення координат; рівень 1 –
// клас "Відрізок на площині" з успадкованими полями, які
// визначають координати одного кінця відрізка, двома додатковими
// приватними полями, що визначають координати другого кінця
// відрізка, методами обчислення довжини відрізка і виведення,
// результату обчислення; рівень 2 – клас "Багатокутник", що має
// приватні поля, які визначають кількості вершин та самі
// вершини, й метод обчислення периметра.
```

Створимо форму, що містить кнопку button1 та текстове поле outputWindow, надавши його властивості Visible із значення **false**.

Крім того, розмістимо на формі розкривний список вибору (компонент ComboBox) із ім'ям comboBox1. Вибравши у вікні Properties у компонента ComboBox властивість Items і клацнувши мишею праворуч від цієї властивості над кнопкою, що містить три крапки, потрапимо у вікно редактора елементів списку, в якому наберемо три рядка:

```
Точка
Відрізок
Багатокутник
```

Над розкривним списком вибору помістимо мітку виведення, записавши в її властивість Text підказку Вибери тип об'єкта.

Визначимо такі три класи:

```
public class Point2D
{
    double x;
    double y;
    string message;
    public double X { get => x; set => x = value; }
    public double Y { get => y; set => y = value; }
    public string Message { get => message;
```

```
                            set => message = value; }

    public Point2D(double x, double y, string message) :
                                                    this(message)
    {
        X = x;
        Y = y;
    }

    public Point2D(string message)
    {
        Message = message;
    }

    public Point2D()
    {
    }

    public void Output(string message, TextBox outputWindow)
    {
        outputWindow.Text = message + "X = " + X + "; Y = " + Y;
    }
}

public class LineSegment : Point2D
{
    protected readonly double xFinish;
    protected readonly double yFinish;

    public LineSegment(double x1, double y1, double x2,
              double y2, string message) : base(x1, y1, message)
    {
        xFinish = x2;
        yFinish = y2;
    }

    public LineSegment() : base()
    {
    }

    public new void Output(string message, TextBox outputWindow)
```

```
    {
        outputWindow.Text = message + Calculate();
    }

    public double Calculate()
    {
        return LineSegmentLength(X, Y, xFinish, yFinish);
    }

    public double LineSegmentLength(
              double x, double y, double xFinish, double yFinish)
    {
        return Math.Sqrt(Math.Pow(x - xFinish, 2)
            + Math.Pow(y - yFinish, 2));
    }
}

public class Polygon : LineSegment
{
    readonly int vertexesNumber;
    readonly Point2D[] polygon;

    public Polygon(string message) : base()
    {
        vertexesNumber = Convert.ToInt32(
            Microsoft.VisualBasic.Interaction.InputBox(
            "Уведіть кількість вершин багатокутника",
            "Уведення кількості вершин багатокутника", "1"));
        polygon = new Point2D[vertexesNumber];
        for (int i = 0; i < vertexesNumber; i++)
        {
            polygon[i] = new Point2D();
            polygon[i].X = Convert.ToDouble(
                Microsoft.VisualBasic.Interaction.InputBox(
                "Координата X вершини " + (i + 1).ToString(),
                "Уведення координат вершин багатокутника", "0"));
            polygon[i].Y = Convert.ToDouble(
                Microsoft.VisualBasic.Interaction.InputBox(
                "Координата Y вершини " + (i + 1).ToString(),
                "Уведення координат вершин багатокутника", "0"));
        }
```

```
        Message = message;
    }

    public double Calculate()
    {
        double perimeter = 0;
        for (int i = 0, k = i + 1; i < vertexesNumber; i++)
        {
            perimeter += LineSegmentLength(polygon[i].X,
                polygon[i].Y, polygon[k].X, polygon[k].Y);
            k++;
            if (k == vertexesNumber)
                k = 0;
        }
        return perimeter;
    }
}
```

Створимо також наступний опрацьовувач події `Click` для кнопки `button1`:

```
private void button1_Click(object sender, EventArgs e)
{
    button1.Visible = false;
    comboBox1.Visible = false;
    label1.Visible = false;
    Point2D obj0;               // Екземпляр класу "Точка".
    LineSegment obj1;           // Екземпляр класу "Відрізок".
    Polygon obj2;               // Екземпляр класу "Багатокутник".
    switch (comboBox1.SelectedIndex)
    {
        case 0:     // Працюємо з точкою.
            this.Text = "Виведення координати точки";
            double pX, pY;
            pX = Convert.ToDouble(
                Microsoft.VisualBasic.Interaction.InputBox(
                    "Координата X точки :",
                    "Уведення координат точки", "0"));
            pY = Convert.ToDouble(
                Microsoft.VisualBasic.Interaction.InputBox(
                    "Координата Y точки :",
                    "Уведення координат точки", "0"));
```

```
            // Нижче створюємо екземпляр - "Точка".
            obj0 = new Point2D(pX, pY, "Координати точки: ");
            // Виведення координат.
            obj0.Output(obj0.Message, outputWindow);
            break;
        case 1:     // Працюємо з відрізком.
            this.Text = "Обчислення довжини відрізка";
            double p1X, p1Y, p2X, p2Y;
            p1X = Convert.ToDouble(
                Microsoft.VisualBasic.Interaction.InputBox(
                    "Координата X",
                    "Уведення координат початку відрізка", "0"));
            p1Y = Convert.ToDouble(
                Microsoft.VisualBasic.Interaction.InputBox(
                    "Координата Y",
                    "Уведення координат початку відрізка", "0"));
            p2X = Convert.ToDouble(
                Microsoft.VisualBasic.Interaction.InputBox(
                    "Координата X",
                    "Уведення координат кінця відрізка", "0"));
            p2Y = Convert.ToDouble(
                Microsoft.VisualBasic.Interaction.InputBox(
                    "Координата Y",
                    "Уведення координат кінця відрізка", "0"));
            // Нижче створюємо екземпляр - "Відрізок".
            obj1 = new LineSegment(p1X, p1Y, p2X, p2Y,
                                   "Довжина відрізка дорівнює ");
            // Виведення довжини.
            obj1.Output(obj1.Message, outputWindow);
            break;
        case 2:     // Працюємо з багатокутником.
            this.Text = "Обчислення периметра багатокутника";
            // Створюємо багатокутник.
            obj2 = new Polygon(
                                "Периметр багатокутника дорівнює ");
            // Виведення площі.
            obj2.Output(obj2.Message, outputWindow);
            break;
    }
    outputWindow.Visible = true;
}
```

Клас Point2D має два приватних поля x і y типу **double** для зберігання координат точки та додаткове поле message, що має тип **string**, для зберігання текстового повідомлення. Крім того, у класі визначено три властивості X, Y і Message, зв'язані з полями x, y і message відповідно. Тіло кожної з властивостей записано з використанням лямбда-виразів. Ініціалізація полів здійснюється конструктором з трьома параметрами, причому цей конструктор для ініціалізації поля message викликає за допомогою ключового слова **this** перевантажений конструктор з одним параметром типу **string**. Метод Output служить для виведення рядка, заданого його першим параметром, у компонент TextBox, заданий другим параметром, а також виведення в зазначений компонент координат точки.

Клас LineSegment, будучи нащадком класу Point2D, має успадковані поля x і y, що визначають координати одного кінця відрізка, і поле message. Ці поля доступні через відповідні властивості. Координати другого кінця відрізка визначають поля xFinish і yFinish, яким властивості у відповідність не поставлені. Основний конструктор має 5 параметрів і викликає конструктор базового класу Point2D для ініціалізації полів x, y та message. Клас має метод Output, що перекриває однойменний метод класу Point2D. Крім того, у цього класу є методи Calculate і LengthSegment. Слід відзначити, можна було б обійтись одним методом – два метода визначено тільки з метою ілюстрування можливостей класів. Зокрема, показано, що для звертання до методу LengthSegment усередині методу Calculate немає необхідності вказувати додаткову інформацію, оскільки кожен метод «знає» всі поля, методи й властивості свого класу.

Клас Polygon має успадковані поля, а також власні нові поля – vertexesNumber (кількість вершин багатокутника) і polygon (масив вершин багатокутника, кожен елемент якого має тип Point2D). Від класу LineSegment він успадковує методи LengthSegment і Output, маючи власний конструктор з одним параметром і власний метод Calculate, який перекриває однойменний метод його базового класу LineSegment. Конструктор формально звертається до конструктора без параметрів свого базового класу LineSegment, який у свою чергу звертається до конструктора свого прабатька Point2D. У тілі конструктора класу Polygon уводиться кількість вершин, створюється

масив точок (об'єктів класу Point2D) і заповнюються елементи цього масиву. При цьому використовується конструктор без параметрів класу Point2D.

Метод button1_Click служить для вибору варіанта розв'язуваної задачі, створюючи екземпляр одного з класів і звертаючись до того або іншого методу Output для виведення результату.

Вибір варіанта розв'язуваної задачі проводиться за допомогою компонента ComboBox, що являє собою розкривний список вибору, який служить для організації меню вибору варіанта і буде розглянутий пізніше (див. підрозділ 20.4). Цей компонент має властивість Items, призначену для зберігання кількох текстових рядків, один з яких вибирається стандартним для Windows методом (наприклад, кліком мишею). Рядки цієї властивості індексуються від 0, а індекс обраного (сфокусованого) рядка автоматично записується у властивість SelectedIndex. Значення цієї властивості використовується в перемикачі інструкції **switch**: 0 – Точка, 1 – Відрізок, 2 – Багатокутник.

При виборі пункту «Точка» виконується перемикання на роботу з класом «Точка». При цьому вводяться координати точки, а також створюється та ініціалізується екземпляр obj0 класу Point2D з наступним звертанням до методу Output цього класу для виведення результату.

При виборі пункту «Відрізок» забезпечується робота з класом «Відрізок». При цьому здійснюється введення координат кінців відрізка, створення та ініціалізація екземпляра obj1 класу LineSegment і звертання до методу Output цього класу, що виводить довжину відрізка, попередньо звернувшись до методу Calculate для її обчислення.

При виборі варіанта «Багатокутник» створюється й ініціалізує екземпляр obj2 класу Polygon і звертається до успадкованого від класу LineSegment методу Output для виведення результату (площі багатокутника). Оскільки в успадкованому методі Output проводиться звертання до методу Calculate, а в класі Polygon є однойменний метод, **ПЕРЕДБАЧАЄТЬСЯ**, що в цьому випадку буде здійснюватися звертання до методу Calculate класу Polygon, а не до однойменного методу класу LineSegment. У запропонованому варіанті реалізації методики опису двох методів Calculate звертання до методу Calculate класу Polygon не виконуватиметься, і площа багатокут-

ника буде виведена неправильно (точніше, вона взагалі **НЕ БУДЕ ОБЧИСЛЮВАТИСЯ**).

Щоб програма працювала правильно, слідом за заголовком методу `Calculate` в оголошенні класу `LineSegment` необхідно зазначити модифікатор **`virtual`**, а в оголошенні методу `Calculate` в оголошенні класу `Polygon` – службове слово **`override`** (див. підрозділ 6.11).

Особливістю змінних типу **`class`** є те, що їм можна присвоювати як значення відповідного типу, так і значення будь-якого похідного типу, тобто об'єкт дочірнього класу може передати записані в ньому значення об'єкта базового класу (але не навпаки).

Наприклад, для визначених у прикладі 6.15 змінних `obj0`, `obj1`, `obj2` правильними є такі інструкції:

```
obj0 = obj1;
obj0 = obj2;
obj1 = obj2;
```

При цьому фактично забезпечується передача посилання на екземпляр дочірнього класу.

Якщо в наведених вище інструкціях присвоювання поміняти місцями ліву й праву частини, то компілятор згенерує помилку.

Якщо екземпляр прабатька був ініціалізований екземпляром дочірнього класу, то, незважаючи на те, що він у цьому випадку посилається на ділянку пам'яті, займану спадкоємцем, звичайний доступ до полів, методів і властивостей може бути реалізований тільки для тих із них, які властиві батьківському класу. Для доступу до поля, оголошеного в дочірньому класі, необхідно тип екземпляра базового класу, з яким проводиться робота, привести (перетворити) до типу дочірнього, що його ініціалізував.

Змінимо, наприклад, в прикладі 6.15 в оголошенні класу `LineSegment` рівень доступу до поля `xFinish` на **`public`**, хоч це й протирічить одній з вимог об'єктно-орієнтованого програмування – поля не повинні бути загальнодоступними. Навіть у цьому випадку після виконання інструкції присвоювання

```
obj0 = obj1;
```

інструкція

```
double w = obj0.xFinish;
```

виявиться помилковою (повідомлення про помилку буде видане компілятором).

Для забезпечення доступу потрібно привести тип об'єкта `obj0` до типу `LineSegment`, тобто скористатися однією з інструкцій

```
double w = ((LineSegment)obj0).xFinish;
```

або

```
double w = (obj0 as LineSegment).xFinish;
```

У C# для визначено спеціальний ***оператор ідентифікації типу***, що задається службовим словом **`is`**. Цей оператор є бінарним, причому першим його операндом є екземпляр класу, а другим – ім'я типу. Результатом виконання оператора **`is`** є булеве значення **`true`**, якщо під час виконання програми екземпляр, заданий першим операндом, має тип, сумісний з типом, заданим як другий операнд оператора. У противному випадку (або якщо екземпляр класу, що перевіряється, не ініціалізований) результатом виконання оператора **`is`** є значення **`false`**. Перевірка може проводитися в такий спосіб:

```
if (obj0 is LineSegment) ...
```

Безпосередньо перетворення типу екземпляра класу в C# здійснюють за допомогою спеціального ***оператора перетворення типів***, що задається службовим словом **`as`**. Як і оператор **`is`**, цей оператор також бінарний, причому першим його операндом є екземпляр класу, а другим – ім'я типу, до якого перетворюється тип першого операнда.

Зазначимо, що спільне використання операторів **`is`** та **`as`** недоцільне, оскільки в цьому випадку буде виконуватися дворазова перевірка типу.

За аналогією з доступом до полів може бути забезпечений доступ екземпляра базового типу до нових методів і властивостей нащадка.

Наприклад, якщо в тілі методу `button1_Click` у прикладі 6.16 замість інструкції

```
obj1.Output(obj1.Message, outputWindow);
```

записати (у даному випадку абсолютно безглузду) послідовність інструкцій

12. ГРАФІЧНІ МОЖЛИВОСТІ C#

Графічні можливості C# визначаються насамперед чотирма спеціалізованими класами – `Graphics` (графіка), `Font` (шрифт), `Pen` (перо), `Pens` (перо) і `Brush` (пензель). Ці класи, як інші класи, що зв'язані з графічними можливостями C#, визначені в просторі імен `System.Drawing`.

12.1. Загальна характеристика класу Graphics і зв'язаних з ним структур і класів

Клас `Graphics` створює графічну поверхню, на якій можна малювати пером, пензлем, шрифтом, а також здійснювати виведення графічних зображень. Графічна поверхня являє собою прямокутник зі сторонами, паралельними межам екрану, і складається з окремих точок (інакше, пікселів), що розташовуються на однаковій відстані одна від одної по горизонталі й вертикалі. Положення пікселів визначається двома координатами – горизонтальною (координата `x`) і вертикальною (координата `y`). Початок системи координат – піксель з координатами (0, 0) – знаходиться у верхньому лівому куті графічної поверхні, а самі координати зростають зліва направо (координата `x`) і зверху вниз (координата `y`).

Працюючи з графічними об'єктами, варто **не забувати про необхідність попереднього їхнього створення**. У класі `Graphics` не визначено жодного конструктора для створення об'єктів. Отримати об'єкт цього класу можна, використавши метод `Control.CreateGraphics`

```
public System.Drawing.Graphics CreateGraphics ();
```

для об'єкта, який є спадкоємцем `System.Windows.Forms.Control`, або шляхом обробки події `Paint` деякого компонента з наступним до-

ступом до властивості Graphics параметра e опрацьовувача цієї події, який є об'єктом класу System.Windows.Forms.PaintEventArgs, що як свою властивість має властивість Graphics. Цей метод викликається кожен раз, коли необхідно використовувати об'єкт Graphics.

Наприклад:

```
Graphics g = this.CreateGraphics();
g.DrawLine(new Pen(Color.Red, 3), 0, 0, 100, 150);
```

По завершенні використання об'єкта Graphics потрібно викликати його метод

```
public void Dispose ();
```

При роботі з графікою досить часто буває необхідним використання деяких допоміжних структур і класів.

У просторі імен System.Drawing для задавання координат точки на двовимірній площині визначено дві структури – Point і PointF. У першій з них координати точки визначаються властивостями

```
public int X { get; set; }
```

і

```
public int Y { get; set; }
```

а в другій – властивостями

```
public float X { get; set; }
```

і

```
public float Y { get; set; }
```

Для опису прямокутних областей зі сторонами, паралельними осям координат у просторі імен System.Drawing також визначено дві структури – Rectangle й RectangleF.

Деякими властивостями структури Rectangle є такі:

```
public int X { get; set; }
```

– горизонтальна координата лівого верхнього кута прямокутника;

```
public int Y { get; set; }
```

– вертикальна координата лівого верхнього кута прямокутника;

```
public int Height { get; set; }
```

– висота прямокутника;

```
public int Width { get; set; }
```

```
public void DrawImage (System.Drawing.Image зображення,
System.Drawing.Rectangle прямокутник);
```

та

```
public void DrawImage (System.Drawing.Image зображення,
System.Drawing.RectangleF прямокутник);
```

малюють зображення в прямокутній області, що задається параметром прямокутник.

Інші методи забезпечують виведення частин зображення, виведення зображення з урахуванням його вихідного розміру тощо.

Наведений нижче приклад ілюструє використання методу DrawImage для виведення зображення в прямокутну область зі збереженням пропорцій.

```
// Приклад 12.5.
// У поточній папці знаходиться файл Baby.jpg. Вивести його в
// клієнтській частині форми з центруванням, збереженням
// пропорцій і задаванням максимально можливого розміру.
```

Помістимо на формі кнопку button1 і створимо для неї такий опрацьовувач події Click:

```
private void button1_Click(object sender, EventArgs e)
{
    button1.Visible = false;
    // Отримуємо об'єкт класу Graphics для форми.
    Graphics g = this.CreateGraphics();
    // Завантаження зображення з файлу.
    Image im = Image.FromFile("Baby.jpg");
    // Оголошуємо прямокутник для майбутнього малювання.
    RectangleF rectF;
    // Лівий верхній кут прямокутника.
    PointF p = new PointF(0, 0);
    // Визначаємо ширину прямокутника w рівній ширині клієнтської
    // області форми, а висоту h визначаємо відповідно висоті
    // клієнтської області форми та співвідношенню ширини
    // клієнтської області форми й ширини зображення.
    float w = this.ClientSize.Width,
    h = (float)this.ClientSize.Width / im.Size.Width *
        im.Size.Height;
    // Якщо прямокутник за висотою поміщається у форму,
```

```
    // плануємо його центрування по вертикалі.
    if (h < this.ClientSize.Height)
        p.Y = (this.ClientSize.Height - h) / 2;
    else // Якщо прямокутник не поміщається за вертикаллю,
    {    // встановлюємо його висоту h рівною висоті клієнтської
         // області форми, пропорційно змінюємо його ширину w
         // і виконуємо центрування за горизонталлю.
        h = this.ClientSize.Height;
        w = (float)this.ClientSize.Height / im.Size.Height *
            im.Size.Width;
        p.X = (this.ClientSize.Width - w) / 2;
    }
    // Створюємо прямокутник для малювання.
    rectF = new RectangleF(p.X, p.Y, w, h);
    // Малюємо зображення в прямокутнику на формі.
    g.DrawImage(im, rectF);
}
```

У наведеному програмному коді після створення об'єктів класів `Graphics` й `Image` і завантаження в останній зображення з файлу здійснюється розрахунок розмірів та положення прямокутника, в якому здійснюватиметься малювання зображення. Для цього співставляються розміри клієнтської області форми і розміри зображення, після чого ширина `w` прямокутника задається рівною висоті клієнтської області форми, а його висота `h` обчислюється у відповідності до ширини так, щоб розміри прямокутника були пропорційними розмірам зображення, тобто планується масштабування по горизонталі. Якщо отримані розміри дозволяють помістити зображення в клієнтську частину форми, то обчислюється майбутнє положення прямокутника таке, щоб виконати центрування по горизонталі. Якщо ж обчислена висота прямокутника більша за висоту клієнтської області форми, то положення і розміри прямокутника обчислюються для масштабування по горизонталі і центрування по вертикалі. Наприкінці коду створюється прямокутник для малювання зображення і здійснюється виведення в нього картинки. Дію застосунку ілюструє рис. 12.3.

Для виведення піктограм у класі `Graphics` визначено три методи, а саме:

```
public void DrawIcon (System.Drawing.Icon піктограма,
int x, int y);
```

– здійснює виведення піктограми в точці з координатами x і y;

```
public void DrawIcon (System.Drawing.Icon піктограма,
System.Drawing.Rectangle приймаючийПрямокутник);
```

– здійснює виведення піктограми в прямокутній області, що задається параметром приймаючийПрямокутник, з масштабуванням зображення за розмірами цієї області;

```
public void DrawIconUnstretched (System.Drawing.Icon
піктограма, System.Drawing.Rectangle приймаючийПрямокутник);
```

– виводить значок, заданий параметром піктограма, в прямокутній області, що задається параметром приймаючийПрямокутник, зі збереженням розмірів значка та його обрізанням, якщо розміри значка більші розмірів приймаючої області.

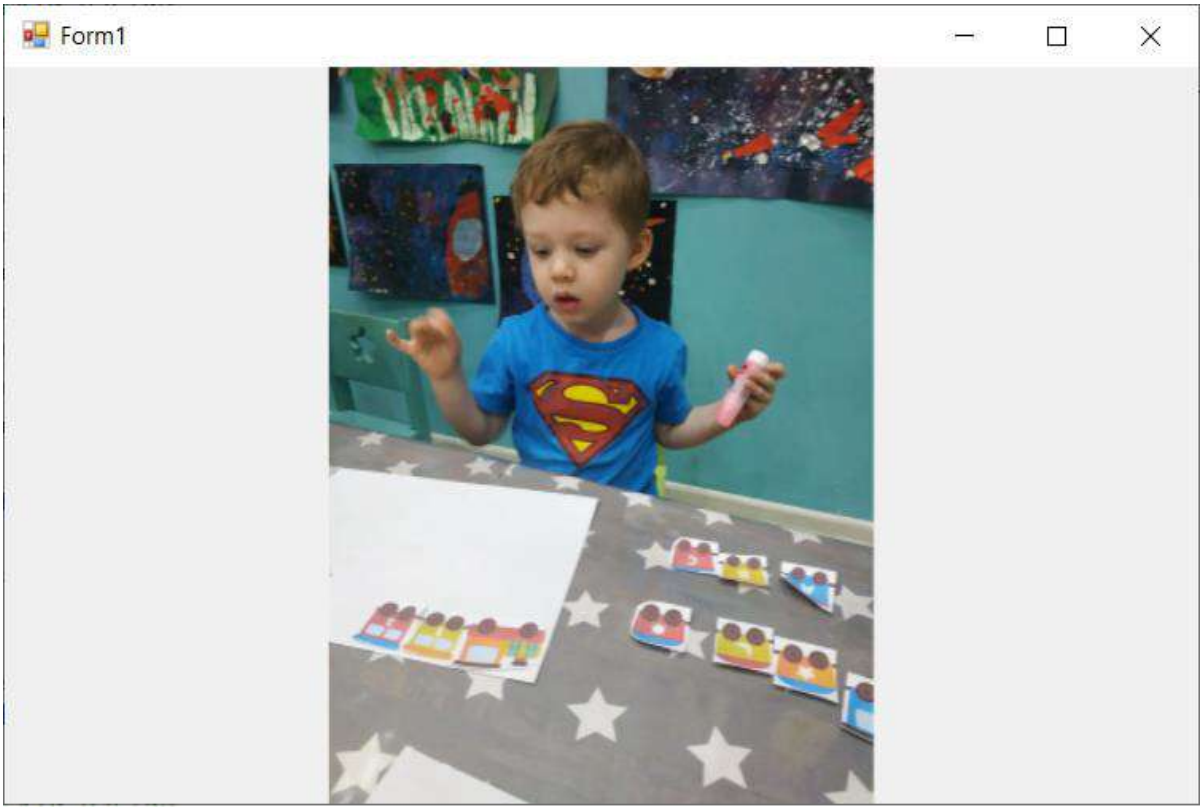


Рис. 12.3. Приклад виведення зображення з масштабуванням

Наведений нижче приклад демонструє можливість збереження зображення у форматах, для яких існують вбудовані конвертори. При цьому завантаження зображення з файлу здійснюється з використанням діалогового вікна відкриття файлу (див. підрозділ 21.2). При збереженні файлу для конвертованого файлу береться ім’я файлу, конвертування

якого здійснювалося, але задається нове розширення. Якщо такий файл у папці вже існує, то до його імені додається підрядок " - Copy" (багаторазово, поки не буде отримане допустиме ім'я).

```
// Приклад 12.6.
// Здійснити вибір файлу з деяким зображенням і виконати
// його збереження відповідно до вибраного формату.
```

Помістимо на порожній формі панель panel1 шириною, достатньою для розміщення на ній компонента ComboBox, і надаймо її властивості Dock значення Right, після чого на вільному місці форми розмістимо ще одну панель panel2 з наданням її властивості Dock значення Fill. На першій панелі розмістимо компонент comboBox1 і дві кнопки button1 й button2. У властивості Text цих трьох компонентів запишемо відповідно тексти BMP, Завантажити та Конвертувати. Крім того, виберемо на формі компонент comboBox1 і у вікні Properties виберемо властивість Items й після кліку над кнопкою ... у правій частині відповідного рядка наберемо у вікні редактора колекції п'ять рядків з текстами BMP, JPEG, PNG, GIF й TIFF. У вікні коду у визначенні класу форми додаймо визначення властивості для роботи з зображенням:

```
public Image Image { get; set; }
```

Тоді можна скористатися такими опрацьовувачами події Click кнопок button1 і button2:

```
void button1_Click(object sender, EventArgs e)
{
    // Визначаємо фільтр.
    openFileDialog1.Filter = "Файли зображень|*.bmp;*.jpg;" +
                 "*.jpeg;*.jpe;*.png;*.gif;*.tiff;*.jfif;" +
                 "*.jps;*.emf;*.wmf;*.eif;*.ico|Усі файли|*.*";
    // Перевірка умови відміни вибору.
    if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
        return;                          // Вихід з button1_Click.
    string fName = openFileDialog1.FileName;
    // Завантаження зображення з файлу.
    Image = Image.FromFile(fName);
    // Отримуємо об'єкт класу Graphics для panel2.
    Graphics g = panel2.CreateGraphics();
    // Малюємо зображення на панелі.
```

```
    System.Drawing.Drawing2D.GraphicsPath clipPath =
        new System.Drawing.Drawing2D.GraphicsPath();
    // Додаємо до об'єкта clipPath еліпс, що торкається меж
    // клієнтської області поля рисунка pictureBox.
    clipPath.AddEllipse(0, 0, pictureBox1.ClientSize.Width,
        pictureBox1.ClientSize.Height);
    // Визначаємо область відсічення графічної поверхні g
    // комбінуванням поточної області відсічення й об'єкта
    // clipPath (тут виконується вирахуванням з поточної області
    // відсічення об'єкта clipPath).
    g.SetClip(clipPath,
        System.Drawing.Drawing2D.CombineMode.Xor);
    // Зафарбовуємо нову область відсічення.
    g.FillRectangle(new SolidBrush(Color.Coral), 0, 0,
    pictureBox1.ClientSize.Width, pictureBox1.ClientSize.Height);
    // Отримуємо нову область відсічення заміною поточної області
    // об’єктом clipPath (еліпсом).
    g.SetClip(clipPath,
        System.Drawing.Drawing2D.CombineMode.Replace);
    // Малюємо еліпс на бітовому зображенні, що міститься на
    // графічній поверхні.
    g.DrawEllipse(new Pen(Color.Blue, 10), 0, 0,
    pictureBox1.ClientSize.Width, pictureBox1.ClientSize.Height);
    // Малюємо на бітовому зображенні текст.
    g.DrawString("C#", new Font("Arial", 250, FontStyle.Bold),
        Brushes.Aqua, new Point(pictureBox1.ClientSize.Width *
                9 / 11, pictureBox1.ClientSize.Height * 4 / 15));
    // Виконуємо збереження малюнка у файлі.
    pictureBox1.Image.Save("Picture.bmp",
        System.Drawing.Imaging.ImageFormat.Bmp);
}
```

Малюнок завантажується на графічну поверхню форми, формування області відсічення (частина прямокутника поза еліпсом) і її зафарбовування, формування нової області відсічення як частини бітового зображення всередині еліпса та малювання на бітовому зображенні еліпса і тексту. Слід відзначити, що з наведеного програмного коду слідує те, що еліпс креслився пером завтовшки 7 пікселів, але він матиме вдвічі меншу товщину, оскільки межа області відсічення проходить посередині лінії. Рис. 12.5, *а* ілюструє також те, що частина тексту, яка виходить за область відсічення не відбивається на малюнку.

а *б*

Рис. 12.5. Ілюстрації до прикладу 12.8: *а* – вихідне зображення; *б* – зображення, отримане після малювання на вихідному зображенні

Наступний приклад 12.9 ще раз ілюструє реалізацію малювання на бітовому зображенню, але на відміну від прикладу 12.8 у ньому як графічна поверхня використана поверхня мітки виведення. У прикладі також показано попікселене коригування бітового зображення.

```
// Приклад 12.9.
// Задача з прикладу 12.8 з виведенням зображення в мітку та
// попіксельним його коригуванням.
```

Рзмістимо на формі кнопку `button1` й мітку виведення `label1` і скористаємося такими опрацьовувачами події `Load` форми та `Click` кнопки:

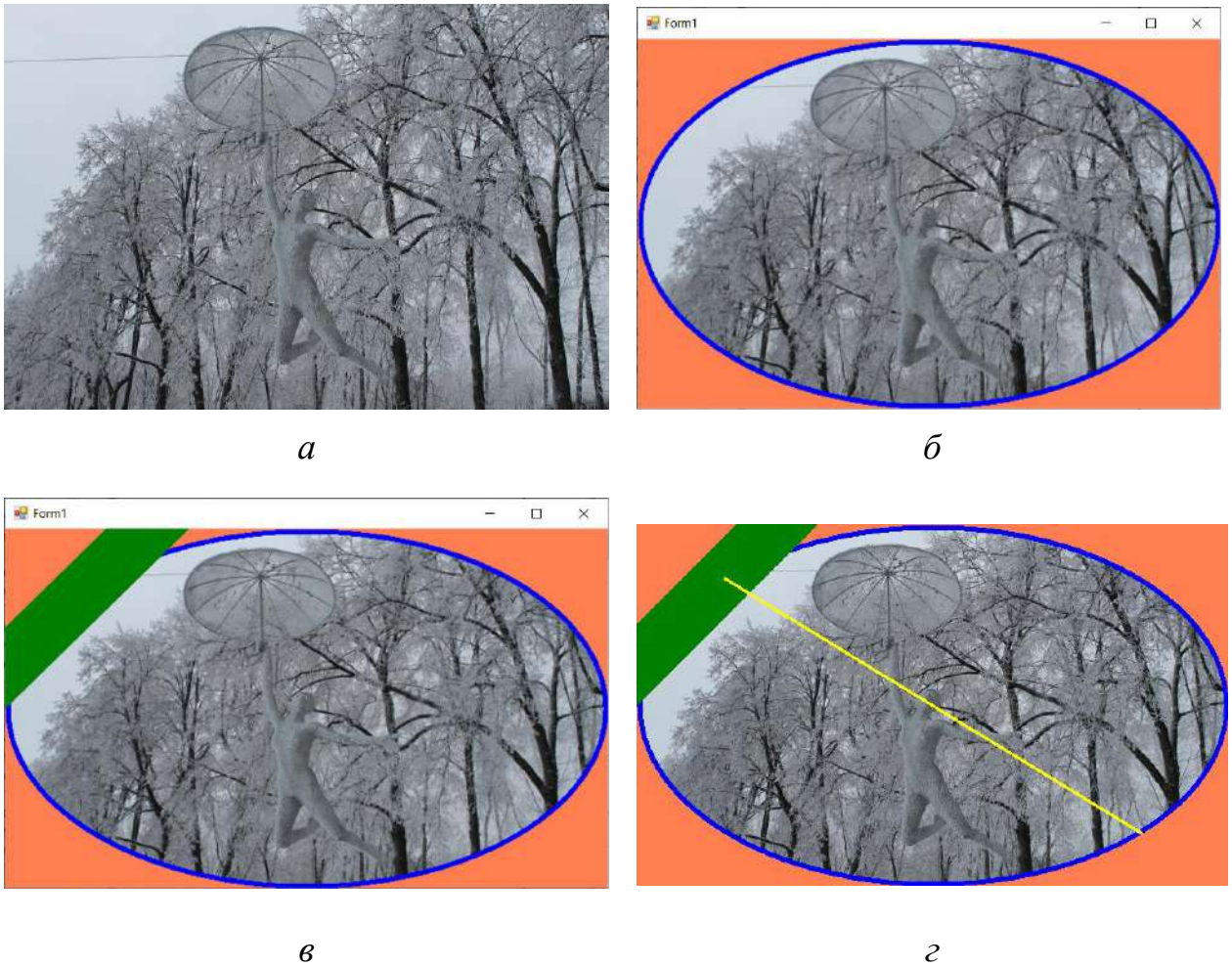
```
private void Form1_Load(object sender, EventArgs e)
{
    this.AutoScroll = true;
    label1.AutoSize = false;
    label1.Text = "";
}

private void button1_Click(object sender, EventArgs e)
{
    button1.Visible = false;
    label1.Location = new Point(0, 0);
    label1.Size = this.ClientSize;
    // Створюємо об'єкт image класу Image з файлу Picture.jpg.
    Image image = Image.FromFile("Photo.jpg");
```

```
    // Створюємо об’єкт bmp класу Bitmap з існуючого зображення.
    Bitmap bmp = new Bitmap(image, label1.Width, label1.Height);
    // Завантажуємо зображення у властивість Image мітки.
    label1.Image = bmp;
    Graphics g;
    // Створюємо графічну поверхню g з об’єкта label1.Image.
    g = Graphics.FromImage(label1.Image);
    // Визначаємо об'єкт clipPath.
    System.Drawing.Drawing2D.GraphicsPath clipPath =
        new System.Drawing.Drawing2D.GraphicsPath();
    // Додаємо до об'єкта clipPath еліпс, що торкається меж
    // клієнтської області мітки label1.
    clipPath.AddEllipse(0, 0, label1.ClientSize.Width,
        label1.ClientSize.Height);
    // Визначаємо область відсічення графічної поверхні g
    // комбінуванням поточної області відсічення й об'єкта
    // clipPath (тут виконується вирахуванням з поточної області
    // відсічення об'єкта clipPath).
    g.SetClip(clipPath,
        System.Drawing.Drawing2D.CombineMode.Xor);
    // Зафарбовуємо нову область відсічення.
    g.FillRectangle(new SolidBrush(Color.Coral), 0, 0,
        label1.ClientSize.Width, label1.ClientSize.Height);
    // Отримуємо нову область відсічення – еліпс.
    g.SetClip(clipPath,
        System.Drawing.Drawing2D.CombineMode.Replace);
    // Малюємо еліпс на бітовому зображенні, що міститься на
    // графічній поверхні.
    g.DrawEllipse(new Pen(Color.Blue, 10), 0, 0,
    label1.ClientSize.Width, label1.ClientSize.Height);
    int w = bmp.Width, h = bmp.Height;
    // Перефарбовуємо попіксельно смугу бітового зображення,
    // причому ця смуга проходить через область відсічення.
    for (int y = 0; y < (w < h ? w : h); y++)
        for (int x = 100 - y >= 0 ? 100 - y : 0;
             x < (w < h ? w : h) / 2 - y; x++)
            bmp.SetPixel(x, y, Color.Green);
    // Ілюстрування того, що навіть на перефарбованому бітовому
    // зображенні дійсна область відсічення.
    g.DrawLine(new Pen(Color.Yellow, 3), new Point(0, 0),
               new Point(label1.ClientSize.Width,
```

```
                label1.ClientSize.Height));
    // Виконуємо збереження малюнка у файлі.
    label1.Image.Save("ResultPhoto.bmp",
                      System.Drawing.Imaging.ImageFormat.Bmp);
}
```

На рис. 12.6, *а* наведено початкове зображення, що виводиться на поверхню мітки, а на рис. 12.6, *б* – вигляд форми з масштабованим за розмірами мітки зображенням й зафарбованою областю відсічення. Два інші рисунки ілюструють відповідно те, що дія області відсічення не розповсюджується на попіксельне коригування бітової поверхні (рис. 12.6, *в*), а також те, що навіть після коригування зображення область відсічення залишається дієвою (рис. 12.6, *г*).



а *б*

в *г*

Рис. 12.6. Ілюстрації до прикладу 12.9: *а* – вихідне зображення у файлі; *б* – форма з зображенням, отриманим після малювання на бітовому зображенні; *в* – форма з зображенням після попіксельної корекції; *г* – результуюче зображення, збережене у файлі

- ✓ `CenterImage (== 3)` – рисунок розміщується в центрі вікна `PictureBox`, причому він обрізається по краям, якщо його розміри більші за розміри вікна;
- ✓ `Zoom (== 4)` – розмір зображення збільшується або зменшується, підлаштовуючись під розміри `PictureBox`, зі збереженням пропорції.

Таким чином, наведений у прикладі 18.6 метод `button1_Click` можна дещо модифікувати:

```
private void button1_Click(object sender, EventArgs e)
{
    // Задаємо положення (лівий верхній кут клієнтської
    // області форми) і розміри компонента.
    pictureBox1.Location = new Point(0, 0);
    pictureBox1.Width = this.ClientSize.Width - panel1.Width;
    pictureBox1.Height = this.ClientSize.Height;
    // Налаштовуємо pictureBox1 так, щоб розмір зображення
    // збільшувався або зменшувався, підлаштовуючись під розміри
    // pictureBox1, зі збереженням пропорції.
    pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;
    // Те, що стосується компонента OpenFileDialog,
    // див. відповідний розділ.
    openFileDialog1.Filter =
        "Графічні файли|*.bmp;*.wmf;*.jpg;*.png;*.gif;*.ico";
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
        // Завантажуємо зображення у властивість Image
        // зі зчитуванням його з файлу.
        pictureBox1.Image =
                        Image.FromFile(openFileDialog1.FileName);
}
```

Наведений нижче рис. 18.2 ілюструє виведення зображення у компонент `PictureBox` для прикладу 18.6 з використанням наведеного останнім програмного коду методу `button1_Click`.

Крім компонета `PictureBox`, визначено ще один компонент для виведення зображень – компонент `ImageList`, з особливостями якого можна ознайомитися в довідковій системі.

Слід відзначити, що виведення зображення можливе в будь-який компонент, що має властивість `Image` (наприклад, у компоненти `Label`, `Button` та інші).

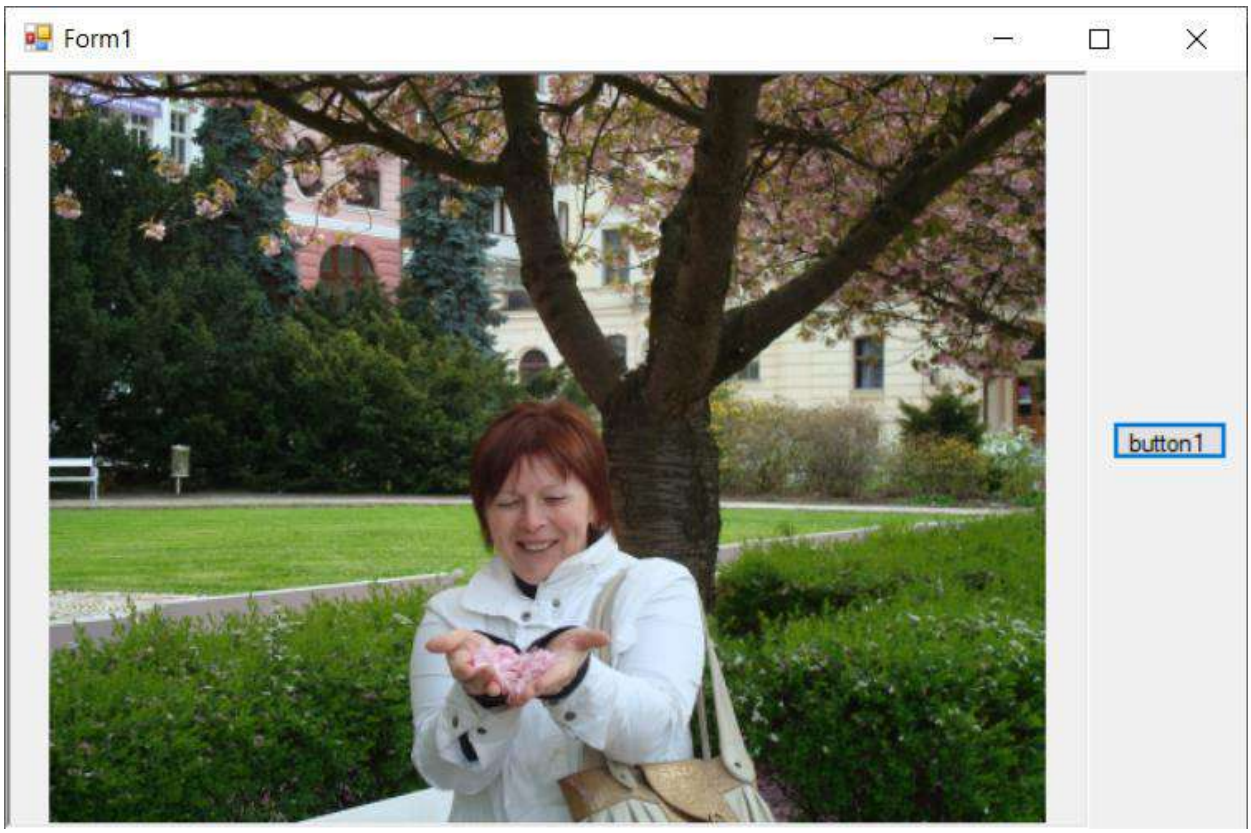


Рис. 18.2. Приклад виведення зображення в компонент `PictureBox`

18.4.2. Мітка виведення – компонент Label

Розглянутий в підрозділі 18.1.1 компонент `Label` може використовуватися і для виведення графічної інформації (див. приклад 12.9).

Аналогічно компоненту `PictureBox` завантаження зображення в цьому компоненті здійснюється у властивість

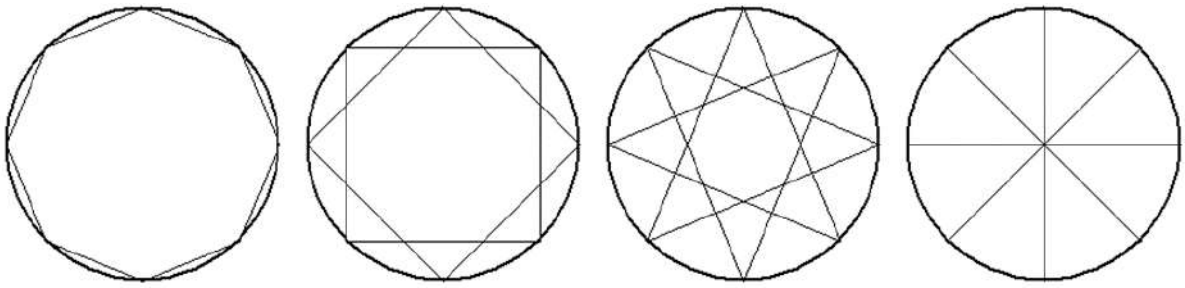
```
public System.Drawing.Image Image { get; set; }
```

Але відсутність властивості `SizeMode` не дозволяє здійснювати керування режимами змінення розмірів компонента і зображення на відміну від компонента `PictureBox`, в якого ця властивість наявна. При необхідності змінення розмірів мітки залежно від розмірів зображення (або навпаки) це виконується примусово при значенні **`true`** властивості `AutoSize` компонента `Label` (див. підрозділ 18.1.1). Якщо ж у компонента `Label` ця властивості має значення **`false`**, то розміри мітки автоматично змінюються у відповідності з розмірами зображення.

Відзначимо також, що зображення може виводитися і в інші компоненти, яким притаманна властивість `Image`.

Завдання для практичного відпрацьовування матеріалу

1. Створити застосунок, що демонструє роботу компонента `CheckBox`, а саме, за командою від кнопки виводити повідомлення про установку прапорця, розміщеного на формі.
2. Створити застосунок, аналогічний попередньому, який демонструє роботу окремого перемикача і перемикача в групі перемикачів.
3. Доповнити завдання 1 вимогою наявності групи перемикачів, що вкладена в іншу аналогічну групу.
4. Дано натуральне число *R*. Побудувати фігури, зображені на рис. 20.1, *а–г*. Фігури утворені колом радіусом *R* і вісьмома точками, що є вершинами вписаного в це коло правильного багатокутника і з'єднані між собою зазначеним способом. Для вибору фігури, що будується, створити групу перемикачів.



а *б* *в* *г*

Рис. 20.1. Фігури для побудови

5. Створити клас `LineSegm`, що реалізує задачу із завдання 1 розділу 12. Від класу `LineSegm` породити два нащадка – `SquarePattern` (візерунок з квадратів, див. завдання 4 з розділу 18) і `TrianglePattern` (аналогічний візерунок з трикутників). Дочірні класи повинні мати властивість, що визначає кількість рівнів вкладеності (тут рекомендується застосувати проміжний клас в ієрархії спадкування). Для класу `SquarePattern` визначити дочірній клас `CompoundSquarePattern`, що реалізує завдання 6 з розділу 18. Визначити аналогічний дочірній клас для `TrianglePattern`. Вибір рисунка виконувати за допомогою груп перемикачів.

СПИСОК ЛІТЕРАТУРИ

1. C# reference. URL: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/ (дата звернення: 04.04.2021).
2. Visual Studio tutorials | C#. URL: https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/?view=vs-2019 (дата звернення: 04.04.2021).
3. C# programming guide. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/ (дата звернення: 04.04.2021).
4. Шилдт Г. C# 4.0: полное руководство. Москва: Диалектика, 2018. 1056 с.
5. Троелсен Э., Джепикс Ф. Язык программирования C# 7 и платформы .NET и .NET Core. Москва: Диалектика-Вильямс, 2018. 1328 с.
6. Коноваленко І. В. Програмування мовою C# 6.0. Тернопіль: ТНТУ, 2016. 227 с.
7. Фленов М. Библия C#. Санкт-Петербург: БХВ-Петербург, 2020. 512 с.
8. Котов О. М. Язык C#: краткое описание и введение в технологии программирования : учебное пособие. Екатеринбург: Изд-во Урал. ун-та, 2014. 208 с.
9. Албахари Б., Албахари Д. C# 7.0. Справочник. Полное описание языка. Санкт-Петербург: Альфа-книга, 2018. 1026 с.
10. Васильев А. Н. Программирование на C# для начинающих. Основные сведения. Москва: Эксмо, 2018. 592 с.
11. Подбельский В. В. Язык C#. Базовый курс. Москва: Финансы и статистика, 2015. 408 с.
12. Подбельский В. В. Язык C#. Решение задач. Москва: Финансы и статистика, 2014. 296 с.
13. Шарп Д. Microsoft Visual C#. Подробное руководство. Санкт-Петербург: Питер, 2017. 848 с.

14. Пахомов Б. C# для начинающих. Санкт-Петербург: БХВ-Петербург, 2014. 432 с.
15. Евдокимов П. В. C# на примерах. Санкт-Петербург: Наука и техника, 2019. 322 с.
16. Павловская Т. А. C#. Программирование на языке высокого уровня. Санкт-Петербург: Питер, 2013. 432 с.
17. Павловская Т. А. Программирование на языке высокого уровня C#. Москва: Интуит, 2016. 245 с.
18. Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. Язык программирования C#. Классика Computers Science. Санкт-Петербург: Питер, 2012. 784 с.
19. Тюкачев Н. А., Хлебостроев В. Г. C#. Алгоритмы и структуры данных: Учебное пособие. Санкт-Петербург: Лань, 2017. 232 с.
20. Стиллмен Э., Грин Д. Изучаем C#. Санкт-Петербург: Питер, 2017. 816 с.
21. Дэвис С. Р., Сфер Ч. C# 2008 для «чайников». Москва: Диалектика, 2009. 585 с.
22. Мюллер Д. П., Семпф Б., Сфер Ч. C# для чайников. Москва: Диалектика, 2019. 608 с.
23. Шарп Дж. Microsoft Visual C#. Подробное руководство. Санкт-Петербург: Питер, 2017. 848 с.
24. Ишкова Э. Самоучитель C#. Начала программирования. Санкт-Петербург: Наука и техника, 2013. 496 с.
25. Зиборов В. В. Visual C# 2012 на примерах. Санкт-Петербург: БХВ-Петербург, 2013. 475 с.
26. Руководство по C# – Часть 1. URL: https://professorweb.ru/my/csharp/charp_theory/level1/index.php (дата звернення: 04.04.2021).
27. Руководство по C# – Часть 2. URL: https://professorweb.ru/my/csharp/charp_theory/level1/index1.php (дата звернення: 04.04.2021).
28. C# Учебник. URL: https://w3schoolsrus.github.io/cs/index.html#gsc.tab=0 (дата звернення: 04.04.2021).

29. Полное руководство по языку программирования C# 9.0 и платформе .NET 5. URL: https://metanit.com/sharp/tutorial/ (дата звернення: 04.04.2021).
30. Безменов, М. І. Турбо Паскаль 7.0 : навч. посіб. / М. І. Безменов. – Харків : НТУ «ХПІ»; Парус™, 2005. – 240 с.
31. Безменов, М. І. Основи програмування в середовищі Delphi : навч. посіб. / М. І. Безменов. – Харків : НТУ «ХПІ», 2010. – 608 с.
32. Безменов М. І. Вступ до інформатики : навч. посіб. / М. І. Безменов. – Харків : НТУ «ХПІ», 2014. – 168 с.
33. Безменов М. І. Збірник задач із програмування : збірник / М. І. Безменов. – Харків : НТУ «ХПІ», 2014. – 304 с.

Навчальне видання

БЕЗМЕНОВ Микола Іванович
БЕЗМЕНОВА Ольга Миколаївна
КАЛІНІН Денис Вікторович

ОСНОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ МОВОЮ C#

Навчальний посібник
для студентів навчально-наукового інституту комп'ютерних наук та інформаційних технологій

Відповідальний за випуск проф. Дорофєєв Ю. І.
Роботу до видання рекомендував проф. Гамаюн І. П.

Редактор М. П. Єфремова

Посібник видано за фінансової підтримки
ІТ-компанії «Cloud Works»

План 2021 р., поз. 78

Підписано до друку 10.01.2023. Формат 60×84 1/16. Папір офсетний.
Друк цифровий. Гарнітура Times New Roman. Ум. друк. арк. 37,6.
Наклад 100 прим. Зам. № 2/01. Ціна договірна.

Видавець і виготовлювач: ФОППанов А. М.
Свідоцтво серії ДК № 4847 від 06.02.2015 р.
м. Харків, вул. Жон Мироносиць, 10, оф. 6,
тел. +38(057)714-06-74, +38(050)976-32-87
copy@vlavke.com