Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра програмування та захисту інформації

ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт студентами
денної та заочної форми навчання спеціальностей
122 "Комп’ютерні науки" та
123 "Комп’ютерна інженерія"

Кропивницький 2024

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра програмування та захисту інформації

Мелешко Є. В., Козірова Н. Л.

ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт студентами
денної та заочної форми навчання спеціальностей
122 "Комп’ютерні науки" та
123 "Комп’ютерна інженерія"

Затверджено
на засіданні кафедри
кібербезпеки та
програмного
забезпечення
Протокол №12
від 19.03.2024

Кропивницький 2024

Візуальне програмування. Методичні вказівки до виконання лабораторних робіт студентами денної та заочної форми навчання спеціальностей 122 "Комп’ютерні науки" та 123 "Комп’ютерна інженерія". – Кропивницький: ЦНТУ, 2024. – 100 с.

Дані методичні вказівки адресовані майбутнім розроблювачам програмного забезпечення, що навчаються за спеціальностями 122 "Комп’ютерні науки" та 123 "Комп’ютерна інженерія". Вони включають в себе основи візуального програмування на прикладі середовища розробки програмного забезпечення Microsoft Visual Studio та ігрового рушія Unity з використанням мови програмування C#. Ці методичні вказівки можуть використовуватися також студентами інших спеціальностей. Вони містять основні теоретичні положення та практичні завдання, необхідні для засвоєння учбового матеріалу.

Схвалено на засіданні методичного семінару
кафедри програмування та захисту інформації
(Протокол № 5 від 20.02.2024)

Укладачі: Мелешко Єлизавета Владиславівна,
доктор технічних наук, професор
Козірова Наталія Леонідівна,
асистент

Рецензенти: Смірнов Олексій Анатолійович,
доктор технічних наук, професор
Минайленко Роман Миколайович,
кандидат технічних наук, доцент

ЗМІСТ

Вступ

Методичні вказівки до лабораторних робіт з дисципліни «Візуальне програмування» для студентів спеціальностей 122 "Комп’ютерні науки" та 123 "Комп’ютерна інженерія" містять в собі теоретичний матеріал, завдання та інструкції виконання до лабораторних робіт і призначені для студентів, що вивчають програмування, а також можуть бути корисними для школярів, вчителів та просто особистостей, що цікавляться інформаційними технологіями.

Також у цих методичних вказівках приклад оформлення звіту з лабораторної роботи, приклад оформлення лістингу програми, шкалу оцінювання успішності студентів та список рекомендованої літератури.

Дані методичні вказівки починають знайомити з основними засобами мови програмування C# для створення додатків у середовищі розробки Microsoft Visual Studio та ігровому рушії Unity. Приводиться багато практичних прикладів та ілюстрацій, які допомагають краще засвоїти викладений матеріал.

Лабораторна робота № 1

Тема: Знайомство з візуальним середовищем програмування Microsoft Visua Studio і мовою програмування C#

Мета: Навчитись створювати проекти у середовищі Visual Studio.

Теоретичні відомості

Платформа .NET Framework – це кероване середовище, що надає різноманітні служби виконуваним в ній програмам. Воно складається з двох основних компонентів: Середовища CLR (ядро виконання, яке обробляє виконувані програми), і бібліотеки класів.NET Framework (яка надає бібліотеку перевіреного коду, який можна використовувати декілька разів і який можна викликати з програм). Служби, які платформа .NET Framework надає працюючим програмам, включають наступне:

– **Керування пам'яттю.** У багатьох мовах програмування програмісти самостійно призначають і виділяють ресурси пам'яті і вирішують питання, пов'язані з часом життя об'єктів. У програмах платформи .NET Framework середовище CLR надає ці служби від імені програми.

– **Система загального типу.** У традиційних мовах програмування, базові типи визначаються компілятором, що ускладнює взаємодію між мовами. В платформі .NET Framework базові типи визначаються системою типу .NET Framework. При цьому використовуються одні й ті ж базові типи для всіх мов .NET Framework.

– **Розширена бібліотека класів.** Замість того щоб писати багато коду для виконання стандартних низькорівневих операцій програмування, розробники можуть використовувати легкодоступну бібліотеку типів і члени з бібліотеки класів .NET Framework.

– **Платформи і технології розробки.** Платформа .NET Framework включає бібліотеки для конкретних областей розробки додатків, наприклад ASP.NET для веб-додатків, ADO.NET для доступу до даних і Windows Communication Foundation для програм, орієнтованих на служби.

– **Взаємодія мов.** Мовні компілятори, націлені на платформу .NET Framework, дають проміжний код, званий мовою CIL (Common Intermediate Language), який, у свою чергу, компілюється під час виконання середовищем CLR. За допомогою цієї функції, підпрограми, написані однією мовою, доступні в інших мовах, а розробники можуть зосередитися на створенні додатків на улюбленому мовою або мовами.

– **Сумісність версій.** За рідкісними винятками, програми, які розробляються за допомогою платформи .NET Framework певної версії, можуть виконуватися без змін на більш пізньої версії.

– **Паралельне виконання.** Платформа .NET Framework допомагає у вирішенні конфліктів версій, дозволяючи інсталяцію декількох версій середовища CLR на одному комп'ютері. Це означає, що кілька версій програм також можуть співіснувати, і що програма може виконуватися на версії платформи .NET Framework, для якої вона була створена.

– **Налаштування для різних версій.** Орієнтуючись на переносиму бібліотеку класів платформи .NET Framework, розробники можуть створювати код, який працює на декількох платформах .NET Framework, наприклад .NET Framework, Silverlight, Windows Phone 7 або Xbox 360.

Створення програмного забезпечення у середовищі Microsoft Visual Studio

Розглянемо для початку можливості даного середовища на прикладі створення консольного додатку.

Оскільки консольні програми виконують введення і виведення даних через командний рядок, вони ідеально підходять для швидкого ознайомлення з можливостями мови та написання службових програм командного рядка.

Приклад 1.1. Консольний додаток Hello World!

1) Запустіть середовище Microsoft Visual Studio. У меню Файл виберіть команди Створити та Проект (рис. 1.1).

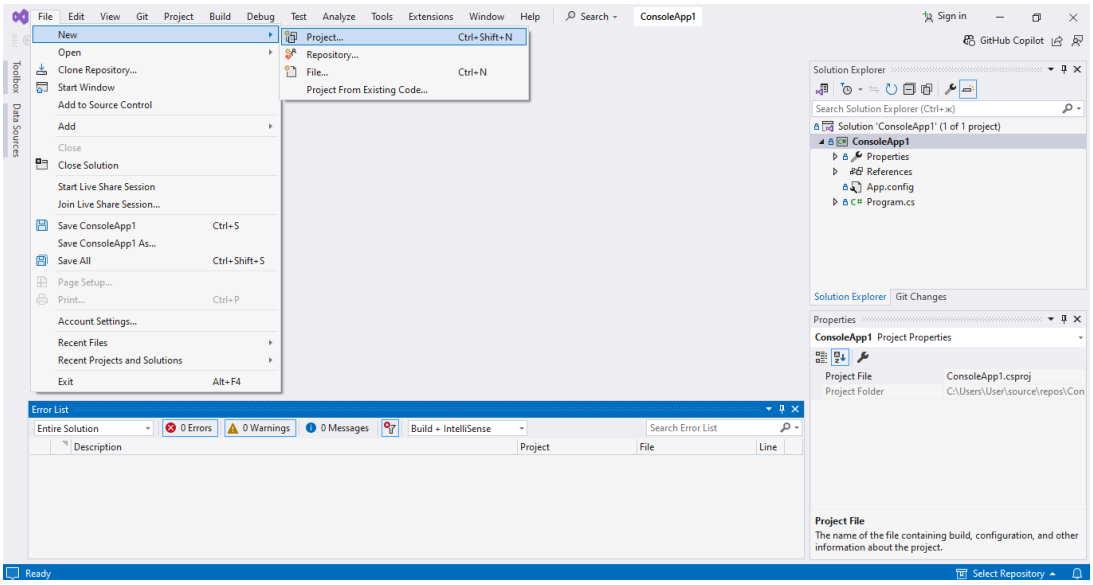


Рисунок 1.1 – Створення нового проекту у середовищі Visual Studio

2) Відкриється діалогове вікно Створення проекту. У цьому діалоговому вікні виводиться список різних типів додатків за замовчуванням, які можна створювати за допомогою Visual C#, експрес-випуск.

В якості типу проекту виберіть Консольну програму, натисніть «Next» (рис. 1.2), в наступному вікні відредагуйте ім'я додатка. Можна залишити розташування проекту за замовчуванням або вказати новий шлях на свій розсуд, після цього натисніть «Create».

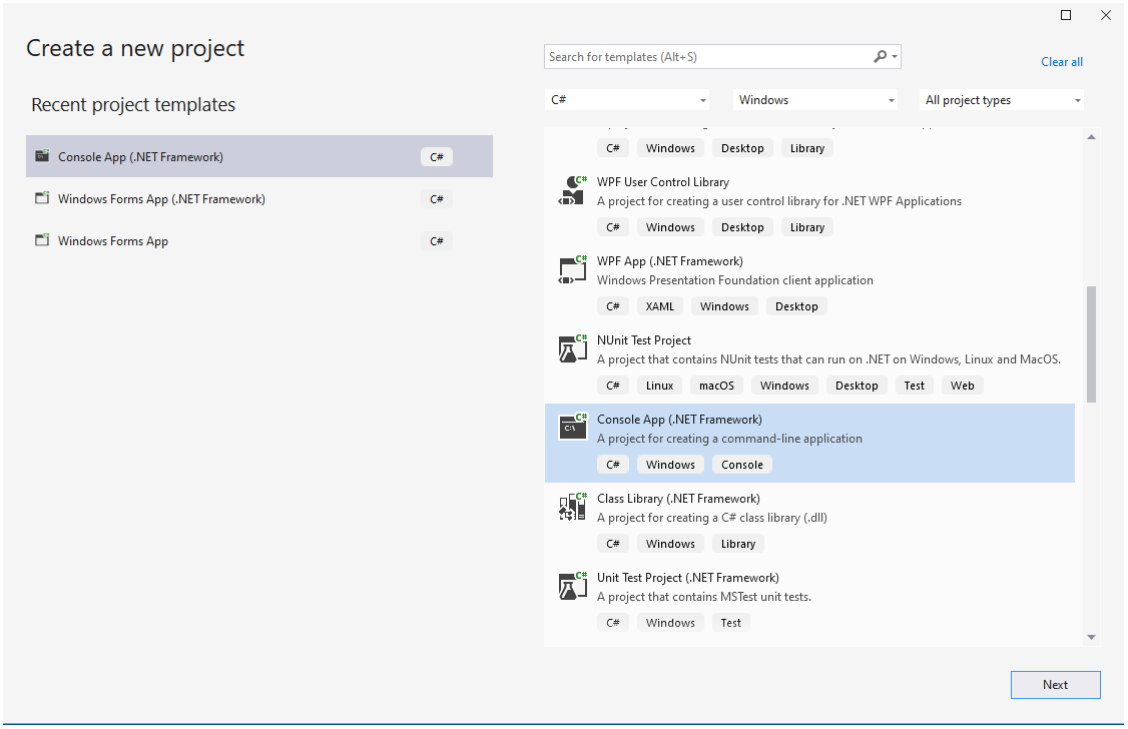


Рисунок 1.2 – Вибір типу проекту (Консольний додаток)

3) Натисніть кнопку ОК.

Visual C#, експрес-випуск створить нову папку для проекту з таким же ім'ям, як у проекту. Відкриється головне вікно Visual C#, експрес-випуск і панель коду для введення і зміни вихідного коду програми C#. (рис. 1.3)

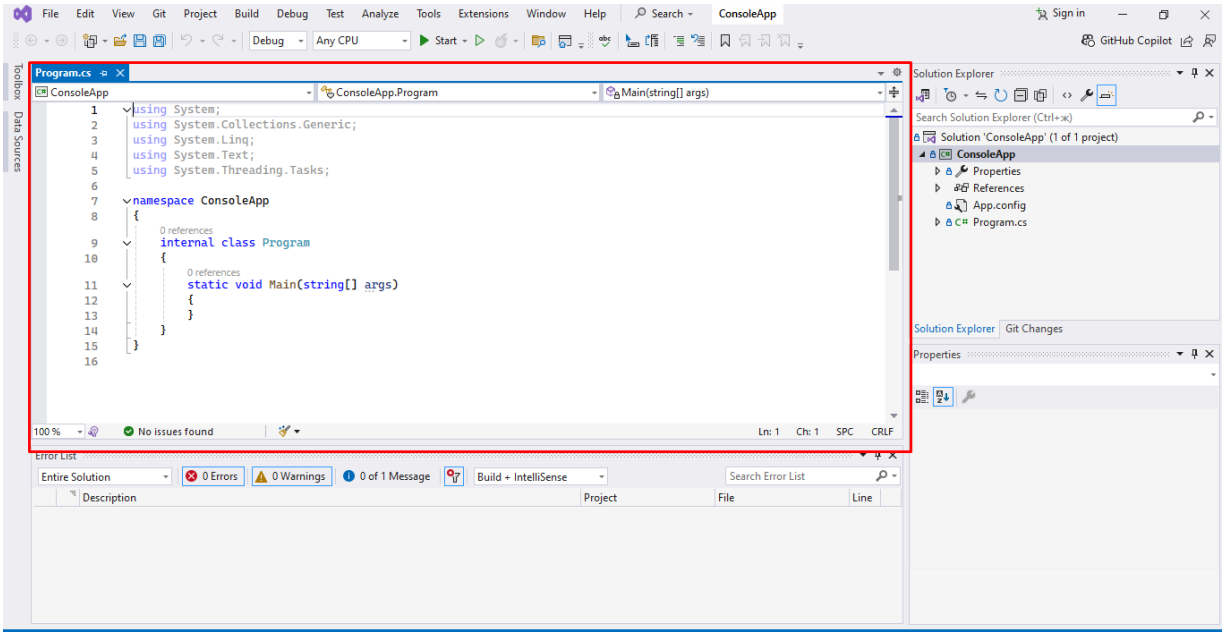


Рисунок 1.3 – Вікно редактора коду
(позначене на рисунку червоним прямокутником)

4) Переконайтеся, що відображається Оглядач рішень, клацнувши вкладку Оглядач рішень в правій частині екрана або значок Оглядач рішень в панелі інструментів (рис. 1.4).

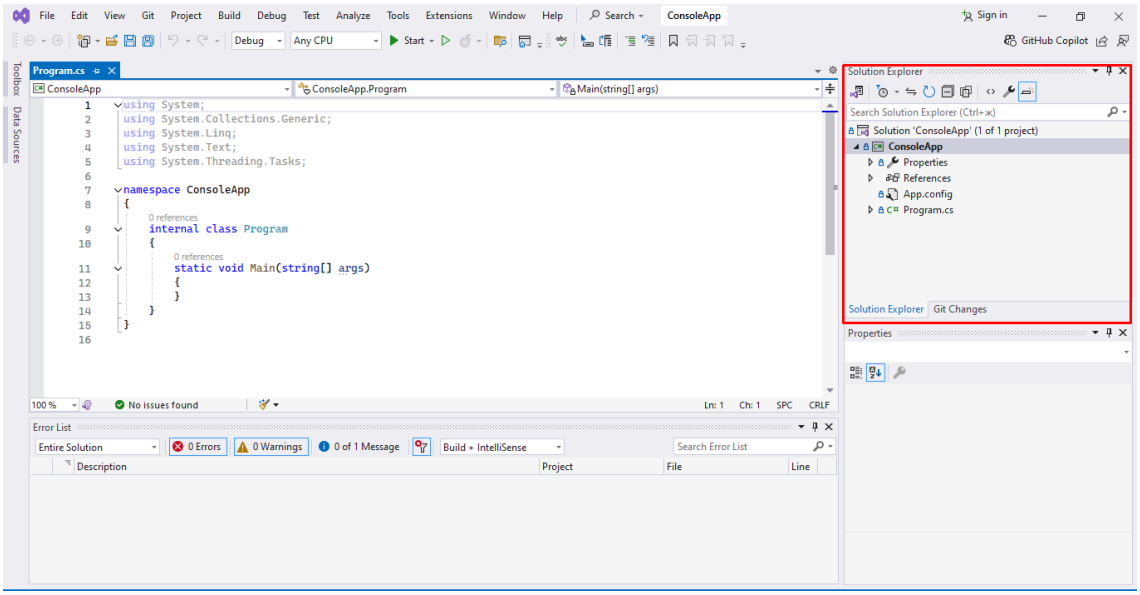


Рисунок 1.4 – Вікно оглядача рішень
(позначене на рисунку червоним прямокутником)

Панель Оглядач рішень є дуже корисною, на ній відображаються різні файли, які є частиною проекту. Найважливішим файлом проекту є "Program.cs", який містить вихідний код програми.

Важливо знати, як відкривати і приховувати вікна, подібні Оглядачеві рішень, це дозволить зберігати Visual C# в належному вигляді. Оглядач рішень за замовчуванням є видимим. Щоб приховати Оглядача рішень клацніть значок Автоприховування (значок канцелярської кнопки в рядку заголовка) або відкрийте меню Параметри в рядку заголовка Оглядача рішень і виберіть Автоприховування. Інші вікна, такі як Представлення класів і Властивості, також мають ці значки.

5) Введіть ім'я класу **Console** в редакторі коду.

Якщо оглядач рішень, як і раніше приховує панель Код, клацніть панель Код для його приховування. Тепер клацніть праву відкриту фігурну дужку ({) всередині методу Main і натисніть ВВЕДЕННЯ для переходу на новий рядок. Зверніть увагу, що редактор робить відступи автоматично.

Редактор коду завжди намагається форматувати введений код в стандартній, зручній для сприйняття структурі. Якщо код виглядає непривабливо, можна переформатувати весь документ, вибравши пункти Додатково та Форматувати документ в меню Правка або натиснувши клавіші CTRL + E, D.

При введенні імені класу або ключового слова C# є вибір: або закінчити введення слова самостійно, або дозволити зробити це вбудованому в панель Код інструменту IntelliSense. Наприклад, при введенні символу "c" з'явиться вспливаючий список слів, пропонованих IntelliSense, з якого можна вибрати слово, що набирається. У цьому випадку, слово "Console" поки ще не видно. Або прокрутіть вниз список, або продовжуйте вводити слово "console". Коли

слово "console" буде виділено в списку, натисніть клавішу ENTER, або TAB, або двічі клацніть його мишею, Console буде додано в код (рис. 1.5).

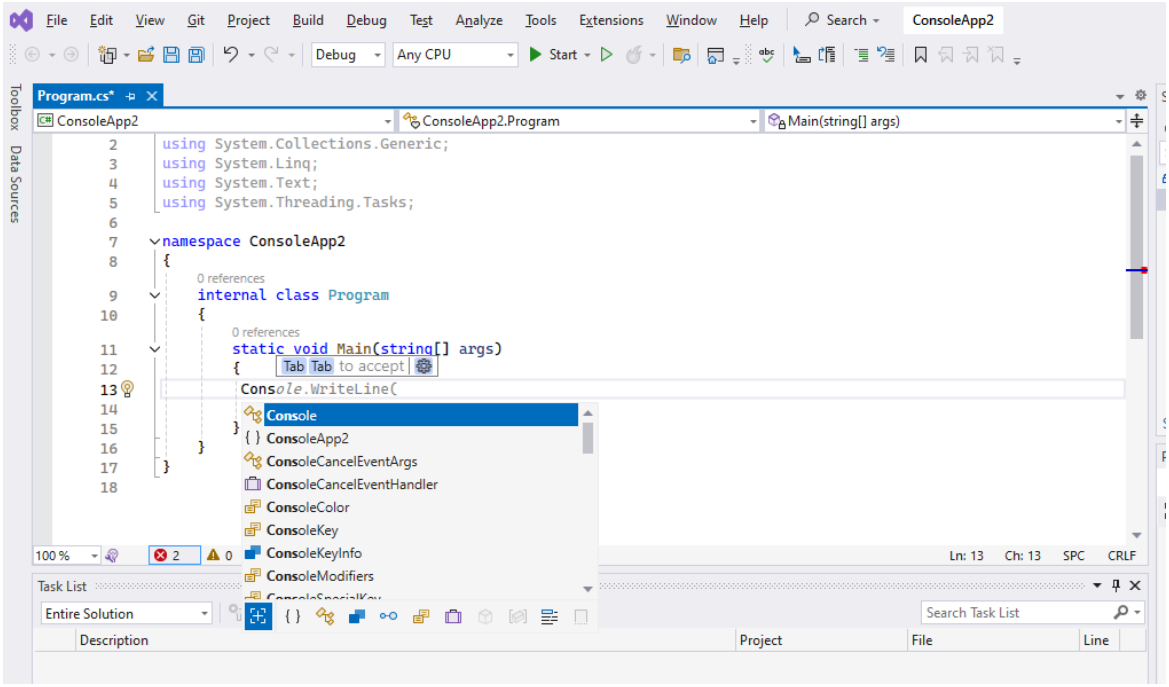


Рисунок 1.5 – Використання Intellisence

Перевагою використання IntelliSense є впевненість у правильності вибору регістра і написання слова. Користувач може або сам вводити код, або використовувати технологію IntelliSense.

6) Введіть точку і ім'я методу **WriteLine**.

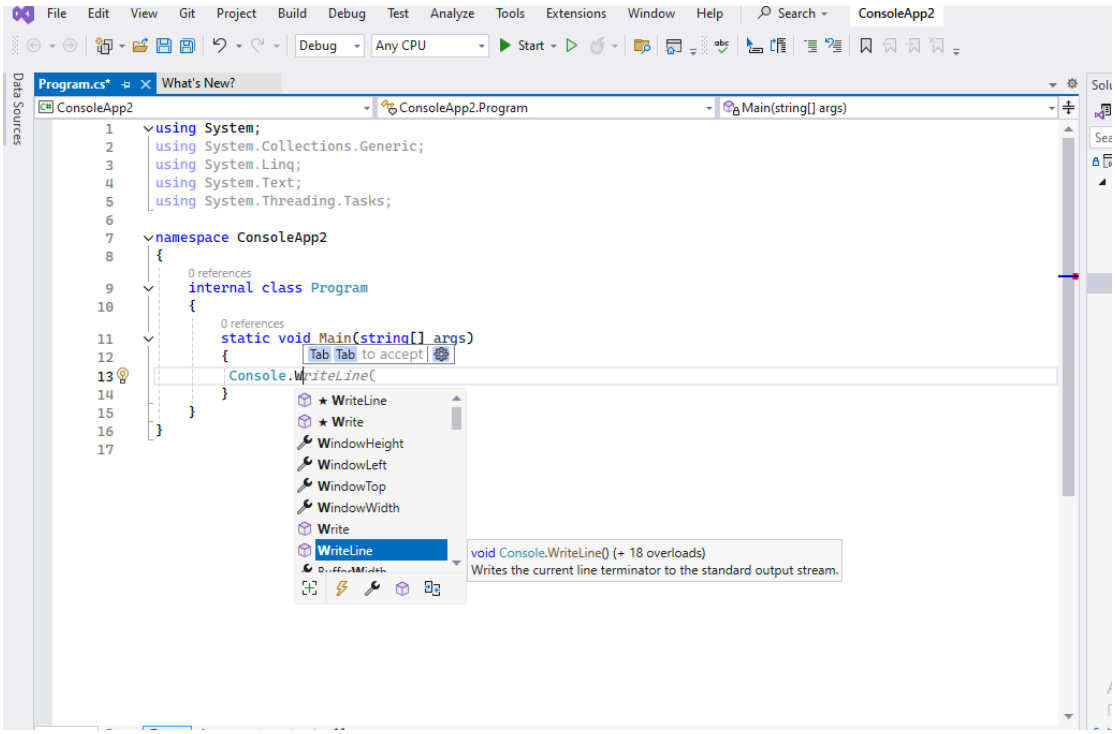


Рисунок 1.6 – Використання Intellisence для вибору методу класу

Як тільки користувач введе точку після Console, IntelliSense відобразить інший список. Список містить імена всіх можливих методів і властивостей, що є частиною класу Console. Необхідний метод WriteLine можна побачити вкінці списку. Або закінчіть введення слова WriteLine, або натисніть клавішу СТРІЛКА ВНИЗ для вибору, потім ENTER, або TAB, або двічі клацніть його мишею. Метод WriteLine буде додано до коду (рис. 1.6).

Введіть відкриваючу дужку. З'явиться повідомлення у вигляді спливаючої підказки з надписами методів, що є ще однією функцією IntelliSense. У цьому випадку буде відображено 19 різних надписів, які можна переглянути за допомогою клавіш СТРІЛКА ВГОРУ та СТРІЛКА ВНИЗ.

7) Введіть або вставте наступний код:

```
static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");

         // Зберегти вікно консолі відкритим до натиснення будь-якої клавіші
            Console.WriteLine("Press any key to exit.");
            Console.ReadKey();
        }
```

Останній рядок у програмі Console.ReadKey() призначений для призупинення виконання програми до натискання будь-якої клавіші. Якщо не додати цей рядок, вікно з командним рядком відразу ж никне і користувач не зможе побачити виведення результату виконання програми. Якщо створювана службова програма завжди буде використовуватися в консолі командного рядка, виклик методу Console.ReadKey() можна опустити.

8) Виконайте програму.

Тепер програма завершена, готова для компіляції і виконання. Для цього або натисніть клавішу F5, або скориставшись кнопкою Пуск на панелі інструментів (рис. 1.7).

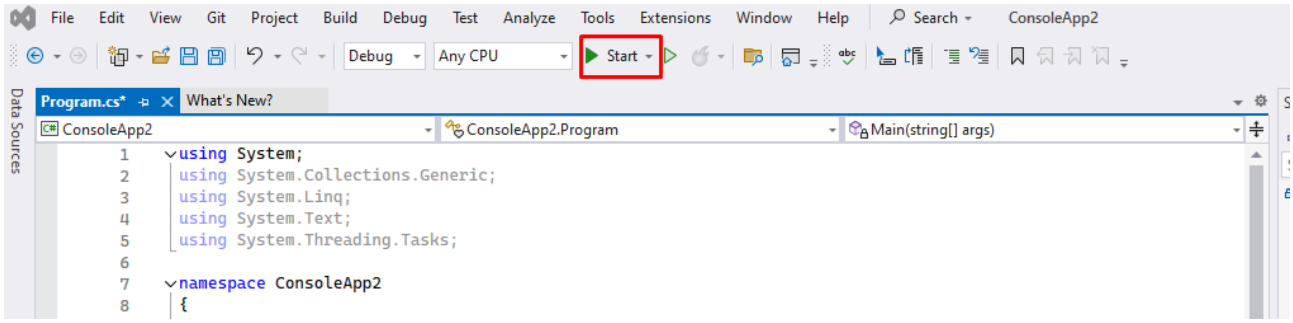


Рисунок 1.7 – Кнопка «Запуск» (виділена червоним прямокутником)

9) Після компіляції і виконання відкриється вікно консолі (рис. 1.8).

Рисунок 1.8 – Результат компіляції та виконання програми

Приклад 1.2. Windows-додаток простий калькулятор

Розглянемо приклад створення простого калькулятора (з графічним інтерфейсом користувача) в середовищі VisualStudio.

Послідовність дій для створення калькулятора:

1) Натисніть Файл → Створити проект... Відкриється вікно, а якому треба вибрати **Windows Forms Application** (рис. 1.9).

2) На формі, що відкриється після попередньої дії, слід розмістити 3 текстових поля (об'єкт TextBox), та 5 кнопок (Button). Об'єкти, які можна розмістити на формі знаходяться в панелі ToolBox (рис. 1.10). Панель ToolBox можна знайти в розділі View.

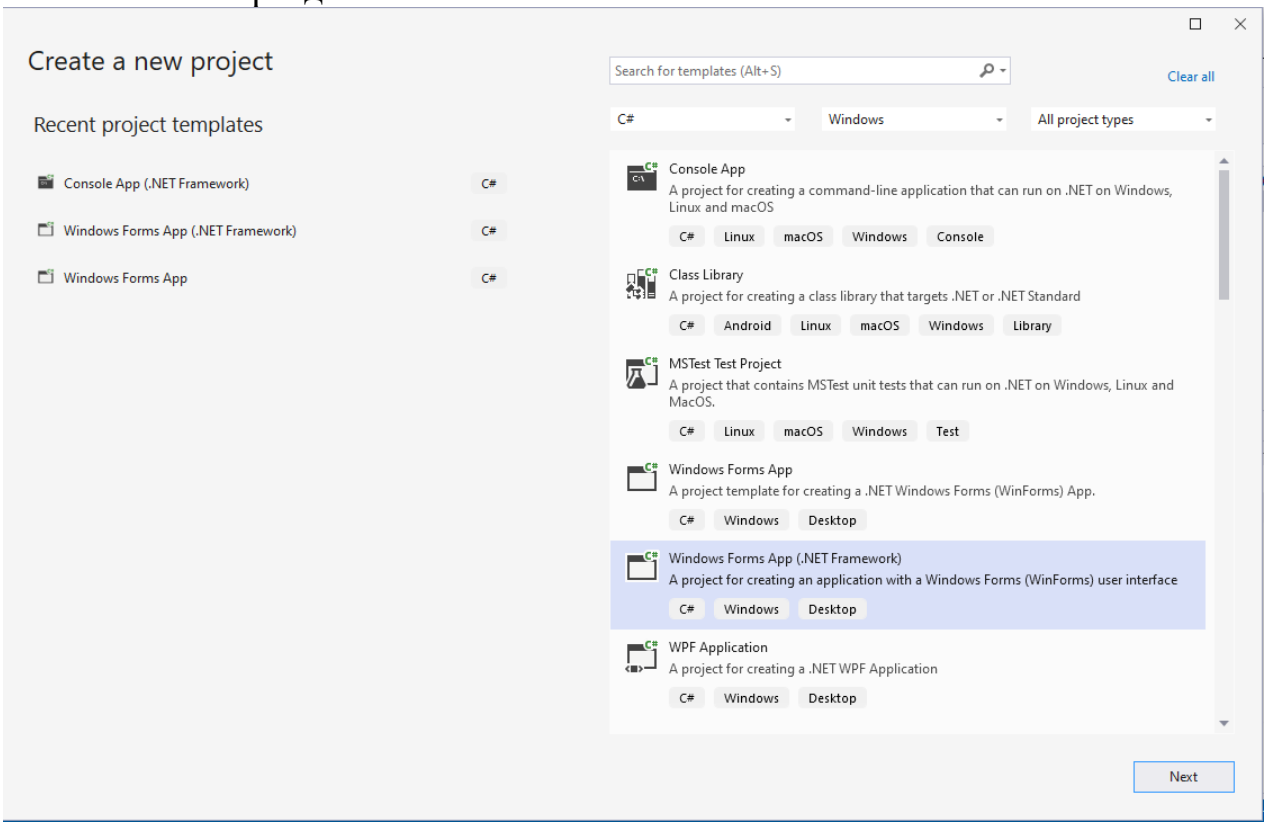


Рисунок 1.9 – Вибір типу проекту (Windows Forms)

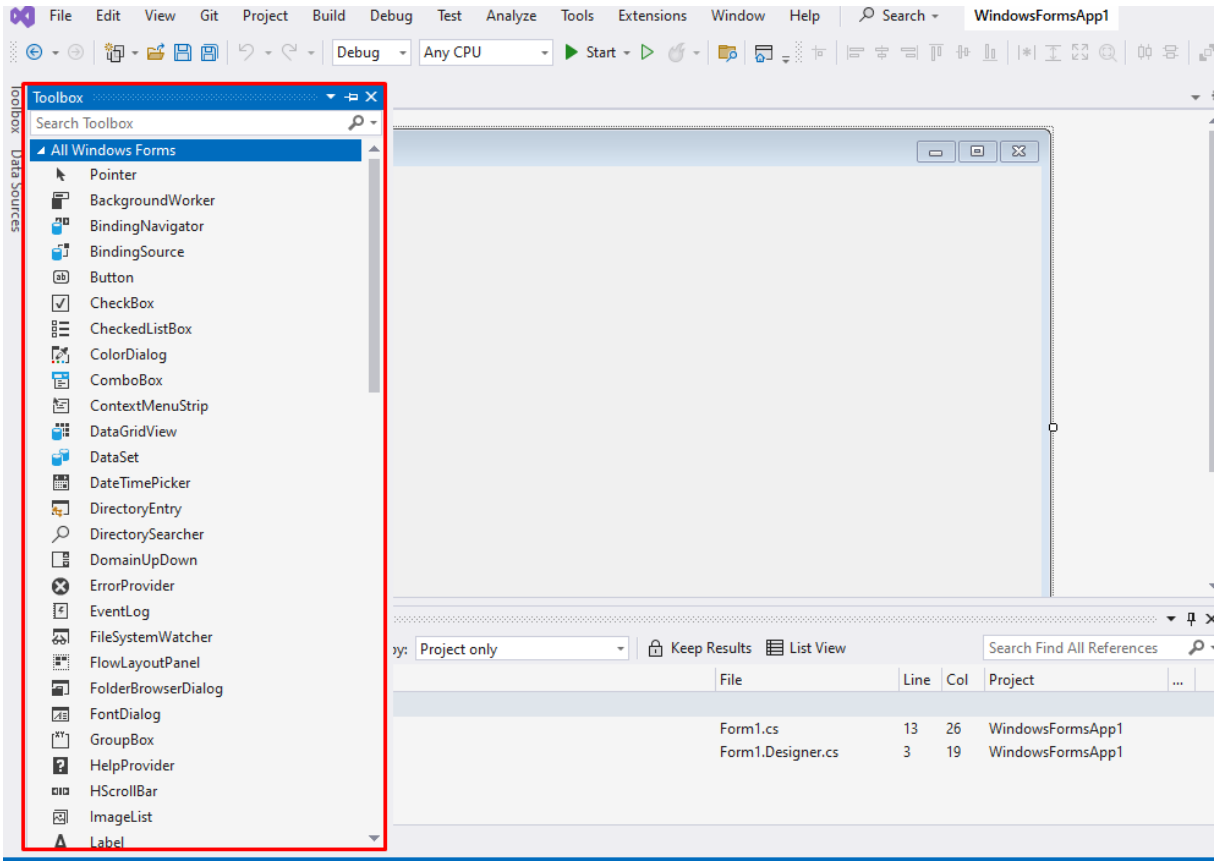


Рисунок 1.10 – Панель ToolBox *(виділена червоним прямокутником)*

3) У кнопок треба змінити властивість Text на наступні значення:

- У button1 властивість Text –"CE";
- У button2 властивість Text –"*";
- У button3 властивість Text –"/";
- У button4 властивість Text –"–";
- У button5 властивість Text –"+".

Властивості змінюються на панелі властивостей, приклад наведено на рис. 1.11.

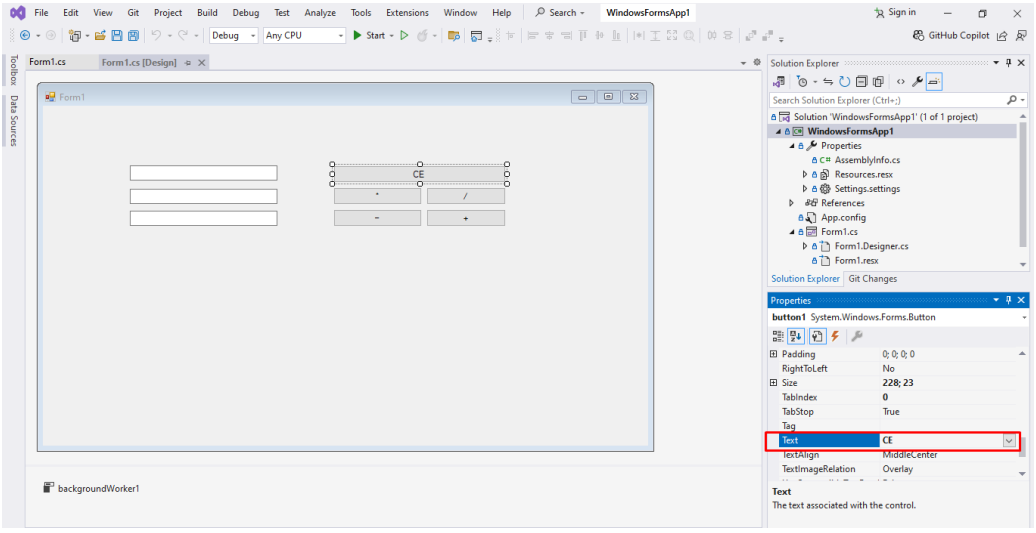


Рисунок 1.11 – Редагування властивості, що містить назву кнопки

Було змінено назви кнопок, їх розмір та назва вікна додатку. Після зазначених дій повинне вийти зображення, наведене на рис. 1.12.

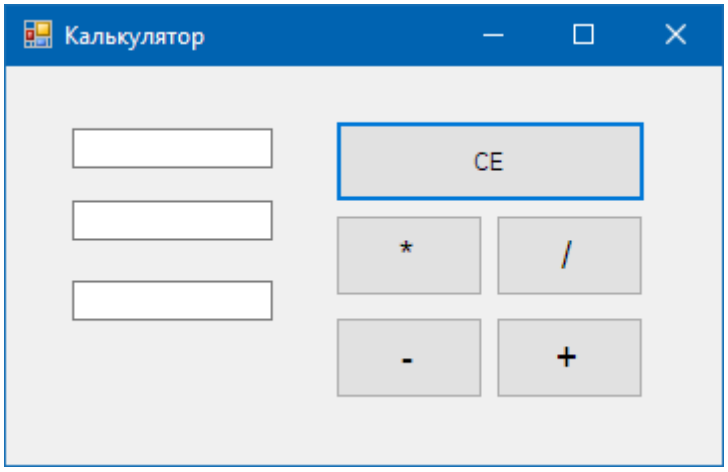


Рисунок 1.12 – Зовнішній вигляд програми Калькулятор

4) Клацніть два рази на створеній кнопці "*" (множення). Відкриється вікно Редактора коду, в якому курсор миші буде розташований всередині функції, що обробляє натиснення на кнопку "*". Додайте туди наступний код:

```
double result = Convert.ToDouble(textBox1.Text) * Convert.ToDouble(textBox2.Text);
```

В цьому коді до змінної `result` з типом `double` привласнюється значення текстових полів, які треба помножити між собою.

Щоб одержати значення текстового поля пишемо ім'я текстового поля (`textBox1`), і властивість `Text` (`textBox1.Text`).

Але чому ми не написали так: `textBox1.Text * textBox2.Text` ?

А тому що в текстовому полі зберігатися текст, а при множенні потрібно використовувати числа.

Тому за допомогою функції `Convert.ToDouble` ми міняємо тип змінної зі строкової в числову (із плаваючою точкою, тому що можливо доведеться множити дроби).

Далі результат, що зберігатися в змінній `result` записуємо в текстове поле 3, при цьому міняємо тип змінної назад з числового в строковий:

```
textBox3.Text = result.ToString();
```

5) Приклад обробки помилки введення:

```
private void button2_Click(objectsender, EventArgs e)
{
    // Якщо нічого не введено
    if (textBox1.Text == "")
       MessageBox.Show("Данні не введені");
    else
    {
       ...//здійснюємо множення
}
}
```

6) Для очистки текстових полів калькулятора застосуємо наступний код:

```
//очищаємо текстові поля (кнопка "CE")
private void button1_Click(object sender, EventArgs e)
   {
   textBox1.Text = "";
   textBox2.Text = "";
   textBox3.Text = "";
   textBox4.Text = "";
   }
```

Завдання до лабораторної роботи:

1. Створити програму "Калькулятор" на мові C# з графічним інтерфейсом, яка повинна виконувати чотири базові арифметичні дії (+, -, *, /), а також операції x^2, x! та обчислення остачі від ділення (%).

2. Створити програму "Розширений калькулятор" на мові C# з графічним інтерфейсом, у яку додайти математичні операції обрані самостійно.

Контрольнізапитання:

1) Опишіть особливості платформи .NetFramework Її переваги та недоліки в порівнянні з іншими платформами.

2) Які інші платформи для роботи з візуальними мовами програмування ви знаєте?

3) Як створити програму у середовищі Visual Studio?

Лабораторна робота № 2

Тема: Робота з масивами

Мета: одержання практичних навичок алгоритмізації й програмування обчислювальних процесів з використанням масивів.

Теоретичні відомості

Масиви

Масив – послідовна впорядкована сукупність елементів деякого типу, які адресуються за допомогою деякого індексу. Тип елементів масиву може бути будь-яким основним або користувацьким типом даних. Він називається базовим типом.

Масив відноситься до посилальних типів даних, тобто розташовується в динамічній області пам'яті, тому *створення масиву* починається з виділення пам'яті під його елементи. Елементами масиву можуть бути величини як значимих, так і посилальних типів (у тому числі масиви). Масив значимих типів зберігає значення, масив посилальних типів - посилання на елементи. Всім елементам при створенні масиву привласнюються значення за замовчуванням: нулі для значимих типів і null - для посилальних.

Кількість елементів у масиві *(розмірність)* не є частиною його типу, ця кількість задається при виділенні пам'яті й не може бути зміненою згодом. Розмірність може задаватися не тільки константою, але й виразом. Результат обчислення цього виразу повинен бути ненегативним, а його тип повинен мати неявне перетворення до int, uint, long або ulong.

Елементи масиву нумеруються з нуля, тому максимальний номер елемента завжди на одиницю менше розмірності. Для *звертання до елемента масиву* після ім'я масиву вказується номер елемента у квадратних дужках.

З елементом масиву можна робити все, що припустимо для змінних того ж типу. При роботі з масивом автоматично виконується контроль виходу за його границі: якщо значення індексу виходить за межі масиву, генерується виключення IndexOutOfRangeException.

Масиви одного типу можна привласнювати один одному. При цьому відбувається привласнення посилань, а не елементів, як і для будь-якого іншого об'єкта посилального типу.

У C# існують 3 розмірності масивів: одномірні, прямокутні та ступінчасті (невирівняні).

Одномірні масиви

Одномірні масиви використовуються в програмах найчастіше. При описі масив можна ініціалізувати, тобто привласнити його елементам початкові значення. Варіанти опису масиву:

```
базовий_тип [] ім'я;
базовий_тип [] ім'я = new тип [ розмірність ];
```

```
базовий_тип [] ім'я = { список_ініціалізаторів };
базовий_тип [] ім'я = new тип [] { список_ініціалізаторів };
базовий_тип [] ім'я = new тип [ розмірність ] { список_ініціалізаторів };
```

Розмірність - це кількість елементів масиву. Всі інструкції з виділення пам'яті формує компілятор до виконання програми. Внаслідок цього розмірність масиву може бути задана тільки константою або константним виразом.

Наприклад:

```
//одномірний масив цілочисельного типу з 5 елементів
int[] array = newint[5];

//одномірний масив речовинного типу з ініціалізацією елементів
float[] x = {64.3, 58.8, 98.8, 0.4};

//одномірний масив беззнакового цілочисельного типу з 2*M елементів
int M = 3;
double  b[2*M] = newdouble[] {3.0, 4.5, 6.1, -0.5, 7.8, 6.4};
```

Якщо кількість ініціалізуємих значень не збігається з розмірністю, виникає помилка компіляції.

Елементи масивів нумеруються з нуля, тому максимальний номер елемента завжди на одиницю менше розмірності. Для доступу до елемента масиву після його ім'я вказується номер елемента (індекс), у квадратних дужках:

```
ім'я_масиву [індекс];
```

Елемент масиву вважається змінною: він може одержувати значення (наприклад, в операторі присвоювання), а також брати участь у вираженнях.

Наприклад, для оголошених вище масивів звертання та використання будуть виглядати в такий спосіб:

```
//звертання до елементів масиву
Console.WriteLine(a[1]);

int i = 2;
Console.WriteLine(b[ i-1]);
Console.WriteLine(x[2*i]);

int z;

a[2] = -1;

z = a[0] + (b[i] + x[i+1]) / a[2*i+1];
c[0] = z - x[i]) / b[2*i+1];
```

Приклад 2.1. Розглянемо приклад знаходження суми позитивних елементів масиву. Завдання: знайти суму позитивних елементів і вивести на екран сам масив. Елементи масиву ввести із клавіатури.

Для спрощення введення чисел створимо метод ReadInt, а для спрощення виводу масиву на екран - метод PrintArray.

```
using System;
using System.Text;

namespace ConsoleApplication9
{
class Program
    {
        //метод для введення цілих чисел із клавіатури
staticint ReadInt(string prompt)
        {
                Console.Write(prompt);
int x = int.Parse(Console.ReadLine());
return x;
        }
        //метод для виведення масиву на екран
staticvoid PrintArray(int[] array)
        {
for (int i = 0; i < array.Length; i++)
            {
                Console.Write("{0,5} ", array[i]);
            }
            Console.WriteLine();
        }

staticvoid Main(string[] args)
        {
int N = ReadInt("Введіть розмірність масиву: ");

int[] a = newint[N];

            //введемо елементи масиву з клавіатури
for (int i = 0; i < N; i++)
            {
                try
                {
                    a[i] = ReadInt("Введіть " + (i+1).ToString() +
                         "-й елемент масиву: ");
                }
catch(FormatException)
                {
                    Console.WriteLine("Невірний формат числа!");
                }
            }

            //оголосимо змінну для зберігання суми елементів
int s = 0.0;

            //просумуємо позитивні елементи масиву
for (int i = 0; i < N; i++)
```

```
if (a[i] >0)

                    s += a[i];

            //виведемо на екран елементи масиву і їхню суму
            Console.WriteLine("Елементи масиву a:" );

            PrintArray(a);

            Console.WriteLine("Сума елементів a: {0,5}", s );
        }
    }
}
```

Часто виникає ситуація, коли кількість елементів масиву може варіюватися на етапі компіляції. Всі масиви у C# мають спільний базовий клас Array, визначений в просторі імен System. У ньому є декілька корисних методів, що спрощують роботу з масивами, наприклад методи одержання розмірності, сортування й пошуку. Так визначення розміру масиву використовується властивість Length:

```
ім'я_масиву.Length
```

При налагодженні програм, що використовують масиви, зручно мати можливість генерувати вихідні дані, задані випадковим чином. У бібліотеці C# на цей випадок є клас Random, визначений в просторі імен System.

Для одержання псевдовипадкової послідовності чисел необхідно спочатку створити екземпляр класу за допомогою конструктора, наприклад:

```
Random a = new Random();    // 1
Random b = new Random(1); //2
```

Є два види конструктора: конструктор без параметрів (оператор 1) використовує початкове значення генератора, обчислене на основі поточного часу. У цьому випадку щоразу створюється унікальна послідовність. Конструктор з параметром типу int (оператор 2) задає початкове значення генератора, що забезпечує можливість одержання однакових послідовностей чисел.

Для одержання чергового значення серії користуються методами, перерахованими у таблиці 2.1.

Таблиця 2.1. Основні методи класу System.Random

Назва	**Опис**
Next()	Повертає ціле позитивне число у всьому позитивному діапазоні типу int
Next(макс)	Повертає ціле позитивне число в діапазоні [0 , макс]

Назва	Опис
Next(хв, макс)	Повертає ціле позитивне число в діапазоні [хв, макс]
NextBytes(масив)	Повертає масив чисел у діапазоні [0, 255]
NextDouble()	Повертає речовинне позитивне число в діапазоні [0.1)

Приклад 2.2. Розглянемо приклад: знайти максимальний елемент у масиві речовинних чисел. Елементи масиву задаються випадковим чином у діапазоні [-100, 100].

Для спрощення заповнення масиву створимо метод InitRandom, а для спрощення виводу масиву на екран скористаємося методом PrintArray з попереднього прикладу.

```
using System;
using System.Text;

namespace ConsoleApplication1
{
class Program
    {
        //ініціалізація масиву
staticvoid InitRandom(double [] array)
        {
            Random gen = new Random();
for (int i = 0; i < array.Length; i++)
                array[i] = 200 * gen.NextDouble() - 100.0;
        }
        //виведення масиву на екран
staticvoid PrintArray(double[] array)
        {
foreach (double a in array)
            {
                Console.Write("{0,9:N5} ", a);
            }
            Console.WriteLine();
        }

staticvoid Main(string[] args)
        {
constint N = 10;

double[] a = newdouble[N];

            InitRandom(a);

            //знайдемо максимальний елемент масиву
            //привласнимо спочатку змінної значення
            //нульового елемента, а потім будемо
            //порівнювати це значення з наступними
            //починаючи з першого
double m = a[0];
for (int i = 1; i < N; i++)
if (m <= a[i])
                    m = a[i];
```

```
            //виведемо на екран елементи масиву і їхню суму
            Console.WriteLine("Елементи масиву a:" );

            PrintArray(a);

            Console.WriteLine("Максимальний елемент масиву a: {0,9:N5}",
                m );
        }
    }
}
```

Прямокутні масиви

Прямокутний масив має більше одного виміру. Найчастіше в програмах використовуються двовимірні масиви. Варіанти опису двовимірного масиву:

```
тип[,] ім'я;
тип[,] ім'я = new тип [ розмір_1, розмір _2 ];
тип[,] ім'я = { список_ініціалізаторів };
тип[,] ім'я = new тип [,] { список_ ініціалізаторів };
тип[,] ім'я = new тип [розмір _1, розмір _2 ] { список_ ініціалізаторів };
```

Приклади описів (один приклад для кожного варіанта опису):

```
int[ , ] a ; //1 елементів немає
int[ , ] b = new int[2, 3]; //2 елементи рівні 0
int[ , ] c = { { 1 , 2, 3 }, { 4, 5, 6 } }; // 3 new мається на увазі
int[ , ] c = new int[ , ] { { 1 , 2, 3 } , {4, 5, 6 } }; // 4 розмірність
обчислюється
int[ , ] d = new int [2, 3] { { 1, 2, 3 } , {4, 5, 6 } }; // 5 надлишковий опис
```

Для доступу до елемента прямокутного масиву вказуються всі його індекси.

При ініціалізації прямокутного масиву він представляється або як масив з масивів, при цьому кожний масив заключається у свої фігурні дужки.

Приклад 2.3. У цілочисельній матриці 3x4 визначити номери рядка й стовпця елемента, що дорівнює нулю. Масив задати цілими випадковими числами від -9 до 9.

```
using System;
using System.Text;

namespace ConsoleApplication9
{
class Program
    {
        //ініціалізація масиву
staticvoid InitRandom(int [,] array, int n, int m)
        {
            Random gen = new Random();
```

```
for (int i = 0; i < n; i++)
for (int j = 0; j < m; j++)
                    array[i, j] = gen.Next(-9, 9);
        }
        //виведення масиву на екран
staticvoid PrintArray(int[,] array, int n, int m)
        {
for (int i = 0; i < n; i++)
            {
for (int j = 0; j < m; j++)
                    Console.Write("{0,3} ", array[i, j]);
                Console.WriteLine();
            }
        }

staticvoid Main(string[] args)
        {
constint N = 3, M = 4;

int[,] a = newint[N, M];

            InitRandom(a, N, M);

            PrintArray(a, N, M);
            //знайдемо нульові елементи масиву
            //і виведемо їхні індекси на екран
for (int i = 0; i < N; i++)
for (int j = 0; j < M; j++)
if (a[i, j] == 0)
                Console.WriteLine("Елемент a["+i+", "+j+"] дорівнює нулю");
        }
    }
}
```

Східчасті масиви

У *східчастих масивах* кількість елементів у різних рядках може розрізнятися. У пам'яті східчастий масив зберігається інакше, ніж прямокутний: у вигляді декількох внутрішніх масивів, кожний з яких має свій розмір. Крім того, виділяється окрема область пам'яті для зберігання посилань на кожний із внутрішніх масивів.

Опис східчастого масиву:

```
тип[][] ім'я;
```

Під кожний з масивів, що складають східчастий масив, пам'ять потрібно виділяти явно, наприклад:

```
int [][] a = new int [3][]; // виділення пам'яті під посилання на три рядки
a[0] = new int[5];    // виділення пам'яті під 0-ий рядок (5 елементів)
a[1] = new int[3];    // виділення пам'яті під 1-ий рядок (3 елементи)
a[2] = new int[4];    // виділення пам'яті під 2-ий рядок (4 елементи)
```

Тут a[0], a[1] і a[2] - це окремі масиви, до яких можна звертатися за ім'ям (приклад наведений у наступному розділі). Інший спосіб виділення пам'яті:

```
int[][] a = { new int[5], new int[3], new int[4] };
```

До елемента східчастого масиву звертаються, вказуючи кожну розмірність у своїх квадратних дужках, наприклад:

```
a[1][2] a[i][j] a[j][i]
```

В іншому використання східчастих масивів не відрізняється від використання прямокутних. Невирівняні масиви зручно застосовувати, наприклад, для роботи із трикутними матрицями великого обсягу.

Завдання до лабораторної роботи:

Варіант 1

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти максимальний елемент масиву.
   - Обчислитисереднє арифметичне елементів масиву.
   - Вивести масив на екран у зворотному порядку.
2. Дано двовимірний масив розмірністю 4x6, заповнений цілими числами. Сформувати одномірний масив, кожний елемент якого дорівнює кількості елементів відповідного рядка, більших даного числа.

Варіант 2

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти мінімальний елемент.
   - Обчислити суму елементів масиву.
   - Вивести позитивні елементи на екран.
2. Дано матрицю розміром 5x4. Поміняти місцями перший рядок і рядок, у якому знаходиться перший нульовий елемент.

Варіант 3

1. Дано одномірний масив, що складається з N речовинних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.

– Знайти максимальний елемент.
– Обчислити середнє арифметичне позитивних елементів масиву.
– Вивести негативні елементи на екран у зворотному порядку.

2. Знайти суму двох матриць розміром nxm.

Варіант 4

1. Дано одномірний масив, що складається з N речовинних елементів.
   – Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   – Знайти мінімальний позитивний елемент.
   – Обчислити добуток не нульових елементів масиву.
   – Вивести ненульові елементи на екран у зворотному порядку.
2. Дано двовимірний масив розміром nxm, заповнений випадковими числами. Визначити, чи є в даному масиві стовпець, у якому рівна кількість позитивних і негативних елементів.

Варіант 5

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   – Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   – Знайти максимальний негативний елемент.
   – Обчислити суму негативних елементів масиву.
   – Вивести позитивні елементи на екран.
2. Дано матрицю A розмірністю nxm. Сформувати одномірний масив B, елементами якого є номери перших негативних елементів кожного рядка масиву A. (0 - негативний елемент відсутній).

Варіант 6

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   – Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   – Знайти максимальний позитивний елемент.
   – Обчислити суму елементів масиву.
   – Вивести ненульові елементи на екран у зворотному порядку.
2. Дано двовимірний масив розмірністю 5x6, заповнений цілими числами із клавіатури. Сформувати одномірний масив, кожний елемент якого дорівнює найбільшому по модулі елементу відповідного стовпця.

Варіант 7

1. Дано одномірний масив, що складається з N речовинних елементів.

– Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
– Знайти максимальний елемент.
– Обчислити середнє арифметичне негативних елементів масиву.
– Вивести масив на екран у зворотному порядку.

2. Знайти середнє арифметичне елементів кожного рядка матриці Q(l,m) і відняти його з елементів цього рядка.

Варіант 8

1. Дано одномірний масив, що складається з N речовинних елементів.
– Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
– Знайти мінімальний елемент.
– Обчислити добуток не нульових елементів масиву.
– Вивести позитивні елементи на екран у зворотному порядку.

2. Дано двовимірний масив розміром nxm, заповнений випадковими числами. Визначити, чи є в даному масиві рядок, що містить більше позитивних елементів, ніж негативних.

Варіант 9

1. Дано одномірний масив, що складається з N цілочисельних елементів.
– Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
– Знайти мінімальний позитивний елемент.
– Обчислити суму позитивних елементів масиву, кратних 3.
– Вивести не нульові елементи на екран.

2. Дано матрицю K(n,m). Сформувати одномірний масив L(m), елементами якого є суми елементів j-ого стовпця.

Варіант 10

– Дано одномірний масив, що складається з N цілочисельних елементів.
– Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
– Знайти максимальний позитивний елемент.
– Обчислити добуток елементів масиву.
– Вивести позитивні елементи на екран.

2. Матриця K(m,m) складається з нулів і одиниць. Знайти в ній номери рядків і стовпців, що не містять одиниці, або повідомити, що таких немає.

Варіант 11

1. Дано одномірний масив, що складається з N речовинних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти максимальний елемент.
   - Обчислити суму парних елементів масиву.
   - Вивести негативні елементи на екран у зворотному порядку.
2. Цілочисельний масив K(n,n) заповнити нулями й одиницями, розташувавши їх у шаховому порядку.

Варіант 12

1. Дано одномірний масив, що складається з N речовинних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти мінімальний негативний елемент.
   - Обчислити середнє арифметичне позитивних елементів масиву.
   - Вивести позитивні елементи на екран.
2. Дано матрицю A(n,m). Сформувати одномірний масив B(n), елементами якого є суми елементів i-ого рядка.

Варіант 13

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти максимальний негативний елемент.
   - Обчислити добуток негативних елементів масиву.
   - Вивести ненульові елементи на екран у зворотному порядку.
2. Дано двовимірний масив розмірністю 5x6, заповнений цілими числами із клавіатури. Сформувати одномірний масив, кожний елемент якого дорівнює добутку парних позитивних елементів відповідного стовпця.

Варіант 14

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти максимальний елемент.
   - Обчислити середнє арифметичне непарних елементів масиву.
   - Вивести негативні елементи на екран.

2. Дано двовимірний масив розміром 8x7, заповнений випадковим чином. Замінити всі елементи перших трьох стовпців на їхні квадрати, в інших стовпцях змінити знак кожного елемента на протилежний.

Варіант 15

1. Дано одномірний масив, що складається з N речовинних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти мінімальний позитивний елемент.
   - Обчислити суму парних елементів масиву.
   - Вивести масив на екран у зворотному порядку.

2. Дано матрицю розміром 8x7, заповнена випадковим чином. Поміняти місцями два середні рядки з першим і останнім.

Варіант 16

1. Дано одномірний масив, що складається з N речовинних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти мінімальний негативний елемент.
   - Обчислити добуток ненульових елементів масиву, кратних 3.
   - Вивести негативні елементи на екран у зворотному порядку.

2. Дано двовимірний масив розміром 5x6, заповнений випадковим чином. Замінити максимальний елемент кожного рядка на протилежний за знаком.

Варіант 17

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти максимальний негативний елемент.
   - Обчислити середнє арифметичне парних елементів масиву.
   - Вивести ненульові елементи на екран у зворотному порядку.

2. Визначити, чи є в даному масиві рядок, що складається тільки з негативних елементів.

Варіант 18

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   - Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   - Знайти мінімальний елемент.
   - Обчислити суму позитивних непарних елементів масиву.

– Вивести позитивні елементи на екран.

2. Дано матрицю розміром 4x5, заповнена випадковим чином. Поміняти місцями перший і останній стовпці.

Варіант 19

1. Дано одномірний масив, що складається з N речовинних елементів.
   – Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   – Знайти мінімальний позитивний елемент.
   – Обчислити добуток непарних елементів масиву.
   – Вивести негативні елементи на екран.
2. Дано двовимірний масив розмірністю 4x5, заповнений цілими числами із клавіатури. Сформувати одномірний масив, кожний елемент якого дорівнює кількості негативних елементів, кратних 3 або 5, що відповідає рядка.

Варіант 20

1. Дано одномірний масив, що складається з N речовинних елементів.
   – Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   – Знайти максимальний елемент.
   – Обчислити середнє арифметичне негативних елементів масиву.
   – Вивести позитивні елементи на екран у зворотному порядку.
2. У кожному рядку, заповненому випадковим чином, матриці розміром nxm поміняти місцями перший елемент і максимальний.

Варіант 21

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   – Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   – Знайти максимальний позитивний елемент.
   – Обчислити суму позитивних парних елементів масиву.
   – Вивести негативні елементи на екран у зворотному порядку.
2. Дано двовимірний масив розміром 6x7, заповнений випадковим чином. Поміняти місцями середні рядки.

Варіант 22

1. Дано одномірний масив, що складається з N цілочисельних елементів.
   – Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
   – Знайти мінімальний елемент.

– Обчислити добуток ненульових непарних елементів масиву.
– Вивести масив на екран у зворотному порядку.

2. Дано двовимірний масив розміром nxm, заповнений випадковим чином. Визначити, чи є в даному масиві рядок, у якому рівно два негативних елементи.

Варіант 23

1. Дано одномірний масив, що складається з N речовинних елементів.
– Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
– Знайти мінімальний позитивний елемент.
– Обчислити середнє арифметичне позитивних елементів масиву.
– Вивести ненульові елементи на екран у зворотному порядку.

2. У матриці Z(m,m) кожний елемент розділити на діагональний, розташований у тому ж стовпці.

Варіант 24

1. Дано одномірний масив, що складається з N речовинних елементів.
– Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
– Знайти максимальний негативний елемент.
– Обчислити середнє арифметичне непарних елементів масиву.
– Вивести негативні елементи на екран.

2. Визначити, чи є в даному масиві стовпець, що складається тільки з позитивних або нульових елементів.

Варіант 25

1.Дано одномірний масив, що складається з N цілочисельних елементів.
– Реалізувати можливості введення масиву із клавіатури або заповнення за допомогою функції Random.
– Знайти мінімальний негативний елемент.
– Обчислити суму негативних елементів масиву.
– Вивести позитивні елементи на екран.

2. Здійснити циклічний зсув матриці nxm на k елементів вправо або вниз (в залежності від введеного режиму). K може бути більшим кількості елементів в стовбці або рядку.

Контрольні питання:

1. Як визначити одномірний масив?
2. Як проініціалізувати одномірний масив?
3. Які варіанти оголошення з ініціалізацією ви знаєте?

4. Як звернутися до елемента масиву?
5. Як згенерувати випадкову величину?
6. Як оголосити прямокутний масив?
7. Як проініціалізувати прямокутний масив?
8. Як звернутися до елемента прямокутного масиву?
9. Як оголосити східчастий масив?
10. У чому різниця між прямокутним і східчастим масивом?
11. Як визначити розмір масиву, знаючи його ім'я?

Лабораторна робота № 3

Тема: Робота з рядками

Мета: Збереження та маніпулювання текстовими даними. Використання масивів та колекцій.

Теоретичні відомості

Рядок в C# представляє собою один або декілька символів, об’єднаних у группу та оголошених за допомогою ключового слова string, яке є прискореним методом мови C# для класу System.String. На відміну від масивів символів в C або C++, рядки в C# набагато простіші у використанні і менш схильні до помилок програмування.

Рядок в C# – це *об'єкт типу String* значенням якого є текст. Кожний символ у рядку займає 2 байти (в .NET за замовчуванням використовується Unicode).

Оголошення та ініціалізація рядків

Створити рядок можна декількома способами:

```
string s; //оголошення без ініціалізації
string s = null; //оголошення та ініціалізація значенням null
string s = "Hello, World!"; //оголошення та ініціалізація текстовим значенням
string s = new string(' ', 20); // конструктор створює рядок з 20 пробілів

char[] a = { '0', '0', '0' }; // масив для ініціалізації рядка
string s = new string(a); // створення з масиву символів
```

Робота з рядками. Методи класу String

Для рядків визначені наступні операції:

1) призначення =;
2) перевірка на рівність ==;
3) перевірка на нерівність !=;
4) звернення по індексу [];
5) конкатенація рядків +.

== Рядки рівні, якщо мають однакову кількість символів і збігаються посимвольно.

[] Звертатися до окремого елементу рядка за індексом можна лише для отримання значення, але не для його зміни. Це пов'язано з тим, що рядки типу string відносяться до так званих незмінних типів даних. Методи, що змінюють вміст рядка, насправді створюють нову копію рядка. Непотрібні "старі" копії автоматично видаляються при збірці сміття.

Об’єднання (конкатенація) рядків

Рядки можна об’єднувати між собою, як показано в наступному прикладі:

```
string s1 = "Рядок більше,";
string s2 = "ніж сума його символів.";

// Об'єднуючи S1 і S2, ми насправді створюємо новий
// об'єкт і зберігаємо його в s1, замінюючи посилання
// на початковий об’єкт посиланням на новий об’єкт

s1 += s2;

System.Console.WriteLine(s1);
// Результат: Рядок більше, ніж сума його символів.
```

Строкові об'єкти є незмінними: після створення їх не можна змінити. Методи, що працюють з рядками, повертають нові рядкові об'єкти. Тому з метою підвищення продуктивності великі обсяги роботи по об'єднанню рядків або інші операції слід виконувати в класі StringBuilder, як показано в далі в прикладах коду.

Таблиця 3.1. Деякі властивості і методи класу System.String

Назва	**Призначення**
Length	Властивість. Дозволяє отримати кількість символів в рядку.
Concat()	Дозволяє з'єднати декілька рядків або змінних типу object.
CompareTo()	Дозволяє порівняти два рядки. В разі рівності рядків результат виконання функції дорівнює нулю. При позитивному значенні функції більшим є рядок, для якого викликався метод.
Copy()	Створює нову копію існуючого рядка.
Format()	Застосовується для форматування рядка з використанням різних примітивів (рядків і числових даних) і підстановлювальних виразів вигляду {0}.
Insert()	Дозволяє вставити один рядок всередину існуючого.
Remove()	Видаляє символи в рядку.
Replace()	Замінює символи в рядку.
ToUpper()	Переводять всі символи рядка в верхній реєстр.
ToLower()	Переводять всі символи рядка в нижній реєстр.
Chars	Дозволяє отримати символ, що знаходиться в певній позиції рядка.
Join()	Створює рядок, сполучаючи задані рядки і розділяючи їх рядком-роздільником.
Replace()	Замінює один символ рядка іншим.

Назва	Призначення
Split()	Повертає масив рядків з елементами - підрядками основного рядка, між якими знаходяться символи-роздільники.
Substring()	Дозволяє отримати підрядок основного рядка, заданої довжини, що починається з певного символу.
Trim()	Видаляє пропуски або набір заданих символів на початку і кінці основного рядка.
ToCharArray()	Створює масив символів і вставляє в нього символи початкового рядка.

Метод ToString()

Всі вбудовані типи даних C# надають метод ToString, перетворюючий значення вказаної змінної в рядок. Цей метод може бути використаний для перетворення числових значень у рядки наступним чином:

```
int year = 1999;
string msg = "Єва народилася в " + year.ToString();
System.Console.WriteLine(msg);  // Результат: "Єва народилася в 1999"
```

Зміна регістра

Щоб змінити регістр літер у рядку (зробити їх великими або малими) слід використовувати ToUpper() або ToLower(), як показано в наступному прикладі.

```
string s6 = "Битва при Гастінгсі, 1066";
System.Console.WriteLine(s6.ToUpper());
// outputs "БИТВА ПРИ ГАСТІНГСІ 1066"
System.Console.WriteLine(s6.ToLower());
// outputs "битва при гастінгсі 1066"
```

Доступ до окремих знаків

До окремих знаків, що містяться в рядку, можна отримати доступ за допомогою таких методів як, наприклад, *Substring*, *Replace* та *IndexOf*.

Отримання підрядка методом Substring:

```
string s3 = "Visual C# Express";
System.Console.WriteLine(s3.Substring(7, 2));
// Результат: "C#"
```

Заміна підрядка методом Replace:

```
System.Console.WriteLine(s3.Replace("C#", "Basic"));
// Результат: "Visual Basic Express"
```

Одержання номера першого входження символу в рядок:

```
// Значення індексів починається з нуля
int index = s3.IndexOf("C");
```

```
// index = 7
```

Доступ до окремих знаків в рядку можливий за допомогою індексу, як показано в наступному прикладі.

```
string s5 = "Друк у зворотному напрямку";

for (int i = 0; i < s5.Length; i++)
{
System.Console.Write(s5[s5.Length - i - 1]);
}
// Результат: "укмярпан умонторовз у курД"
```

Escape-символи

Рядки можуть містити escape-знаки, такі як "\ n" (новий рядок) і "\ t" (табуляція). Рядок:

```
string columns = "Стовпчик 1\tСтовпчик 2\tСтовпчик 3";
//Результат: Стовпчик 1        Стовпчик 2        Стовпчик 3

string rows = "Рядок 1\r\nРядок 2\r\nРядок 3";
/* Результат:
Рядок 1
Рядок 2
Рядок 3
*/

string title = "\"The \u00C6olean Harp\", by Samuel Taylor Coleridge";
//Результат: "The Æolean Harp", by Samuel Taylor Coleridge
```

Якщо потрібно додати в рядок зворотну скісну риску, перед нею потрібно поставити ще одну зворотну скісну риску. Наступний рядок:

```
string fff = "\\\\My Documents\\";
```

еквівалентний рядку:

\\My Documents\

Символ @

Символ @ вказує, що при створенні рядків слід ігнорувати escape-знаки і переноси рядків. Наступнідварядки є ідентичними.

```
string p1 = "\\\\My Documents\\MyFiles\\";
string p2 = @"\\My Documents\MyFiles\";
```

Ще приклади застосування символу **@:**

```
string filePath = @"C:\Users\scoleridge\Documents\";
//Результат: C:\Users\scoleridge\Documents\

string quote = @"Her name was ""Sara.""";
//Результат: Her name was "Sara."
```

Таблиця 3.2. Escape-символів мови C#

Escape-Послідовність	Ім'я символу	Кодування Юнікоду
\’	Одинарні лапки	0×0027
\"	Подвійні лапки	0×0022
\\	Зворотна коса риса	0x005C
\0	Null	0×0000
\a	ALERT	0×0007
\b	Backspace	0×0008
\f	FORM FEED	0x000C
\n	Новий рядок	0x000A
\r	Повернення каретки	0x000D
\t	Горизонтальна табуляція	0×0009
\U	Escape-Послідовність Юнікоду для пар символів-заступників.	\Unnnnnnnn
\u	Escape-Послідовність Юнікоду	\u0041 = "A"
\v	Вертикальна табуляція	0x000B
\x	Escape-Послідовність Юнікоду аналогічна "\u", за винятком рядків зі змінною довжиною.	\x0041 = "A"

Клас StringBuilder та його методи

Іноді слід уникати ситуацій, коли в результаті виконання операції створюється новий рядок, оскільки це пов'язано з додатковими витратами пам'яті та інших ресурсів комп'ютера при виконанні операції.

C# містить спеціальний клас StringBuilder, використовуючи який можна уникнути створення копій рядків при їх обробці. Всі зміни, що вносяться до об'єкту даного класу, негайно відображаються в ньому, що ефективніше, ніж робота з копіями рядка.

Основною операцією, яка найчастіше використовується класом **StringBuilder**, є операція додавання до рядка вмісту. Для цього існує метод **Append**. Наступний код додає один рядок до іншого і виводить результат на консоль. При цьому змінюється оригінал рядка, копія не створюється:

```
StringBuilder sb = new StringBuilder("Наступного тижня у нас модульній контроль");
sb.Append(", потрібно захистити лабатораторні роботи");
Console.Write(sb);
```

Окрім додавання клас StringBuilder містить інші методи, найбільш значимі з яких наведені нижче. Після того, як усі необхідні дії, пов'язані з

обробкою рядка, були виконані, необхідно викликати метод ToString() для перетворення вмісту об'єкту в звичайний тип даних string.

Таблиця 3.3. Деякі методи класу StringBuilder

Назва	Призначення
Append	Додавання заданого рядка в кінець рядка об'єкту.
AppendFormat	Додавання заданого форматованого рядка (рядка, що містить управляючі символи) в кінець рядка об'єкту.
CopyTo	Копіювання символів заданого сегменту рядка в задані комірки масиву символів.
Insert	Додавання рядка в задану позицію рядка об'єкту.
Remove	Видалення заданої кількості символів з рядка об'єкту
Replace	Заміна заданого символу або рядка об'єкту на інший заданий символ або рядок.

При інтенсивній роботі з рядками рекомендується використовувати клас **StringBuilder**, оскільки це дозволяє зменшити накладні витрати, пов'язані із створенням копії рядка при виконанні кожної операції.

Приклад роботи з класом StringBuilder:

```
string question = "hOW DOES mICROSOFTwORD DEAL WITH THE cAPSlOCK KEY?";

//запис змінної типу String в змінну типу StringBuilder
System.Text.StringBuilder sb = new System.Text.StringBuilder(question);

for (int j = 0; j <sb.Length; j++)
{
if (System.Char.IsLower(sb[j]) == true)
sb[j] = System.Char.ToUpper(sb[j]);
elseif (System.Char.IsUpper(sb[j]) == true)
sb[j] = System.Char.ToLower(sb[j]);
}
// Збереження відредагованого рядка
string corrected = sb.ToString();
System.Console.WriteLine(corrected);
// Результат: Howdoes Microsoft Word deal with the CapsLock key?
```

Приклад 3.1

Розглянемо роботу з рядками в C# на прикладі простого шифрування тексту – так званої *літореї.*

Літорея (від littera) – тайнопис, рід шифрованого листа, що використовувався в давньоруській рукописній літературі. Відома літорея двох типів: *проста* та *мудра.* Проста, інакше називається тарабарською грамотою, полягає в наступному: поставивши приголосні букви у два ряди, у порядку:

б	**в**	**г**	**д**	**ж**	**з**	**к**	**л**	**м**	**н**
щ	**ш**	**ч**	**ц**	**х**	**ф**	**т**	**с**	**р**	**п**

записують у шифрованому листі верхні букви замість нижніх і навпаки – нижні замість верхніх, причому голосні залишаються без зміни; так, наприклад, *лсошамь* = *словник* і т.п.

Мудра літорея припускає більш складні правила підстановки. У різних варіантах, що дійшли до нас, використовуються підстановки цілих груп букв, а також числові комбінації: кожній приголосній букві ставиться у відповідність число, а потім відбуваються арифметичні дії над отриманою послідовністю чисел (наприклад, до всіх чисел додавалася деяка константа - ключ).

По своїй суті Літорея є шифром простої заміни, що легко дешифрується сучасними методами.

Створимо Windows-додаток з графічним інтерфейсом користувача, що реалізує алгоритм шифрування простою літореєю.

Для реалізації на C# роботи з текстом використовуємо елемент керування **richTextBox** – поле форматованого тексту.

Приклад сирцевого коду, що реалізує просту літорею:

```
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;

//Програма для кодування тексту
//методом простої літореї
public partial class Form1 : Form
{
  const string lit1 = "АБВГДЕЄЖЗИІЇЙКЛМ";
  const string lit2 = "НОПРСТУФХЦЧШЩЬЮЯ";
  public Form1()
  {
    InitializeComponent();
  }
  //Обробка натиснення на кнопку "шифрувати"
  private void btnGo_Click(object sender, EventArgs e)
  {
    //вхідний текст:
    string text = rtxSource.Text;

    //літорейний текст:
    string litorea = "";

    //довжина тексту:
    int len = text.Length;

    //шифруємо - заміняємо букви парними:
    for (int i= 0; i < len; ++i){
```

```
            //чергова буква:
            char chr= text[i];
            chr = char.ToUpper(chr);
            int n = lit1.IndexOf(chr);
            if (n > -1) // верхня буква
                //заміняємо нижньою:
                litorea += lit2[n];
            else
            {
                n = lit2.IndexOf(chr);
                if (n > -1) //нижня буква
                    //заміняємо верхньою:
                    litorea += lit1[n];
                else //буква Е або інший символ:
                    litorea = litorea + chr;
            }
        }
         //виводимо зашифрований текст у другому текстовому полі:
         rtxTarget.AppendText(litorea);
      }
      //очистити текстові вікна
      private void btnClear_Click(object sender, EventArgs e)
      {
         rtxSource.Clear();
         rtxTarget.Clear();
      }
   }
```

Приклад скриншоту програми наведено на рис. 3.1.

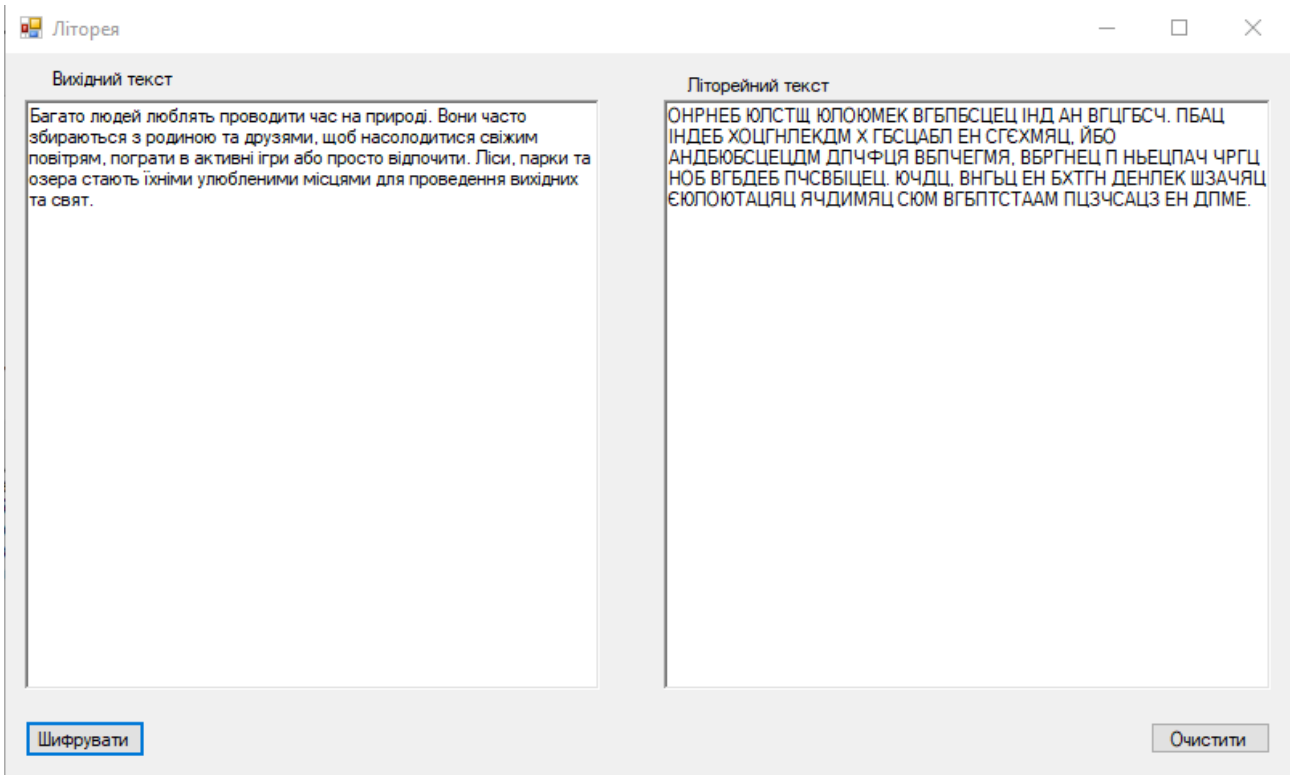


Рисунок 3.1 – Вікно програми «Літорея»,
що здійснює шифрування тексту

Завдання до лабораторної роботи:

Варіант 1

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує всі слова в тексті у зворотному порядку.

Варіант 2

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує з 1-го тільки ті слова у яких є літера «р».

Варіант 3

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує з 1-го тільки ті слова, що починаються на голосну букву.

Варіант 4

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст українськими літерами, в 2-ге програма записує транслітерацію тексту в латинський алфавіт.

Варіант 5

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує з 1-го тільки ті речення, які не містять ком та двокрапок.

Варіант 5

Написати програму, у якій є текстове поле richTextBox та textBox. В поле richTextBox користувач вводить текст, а в поле textBox програма записує число, що відповідає кількості слів в тексті з 5-ти букв.

Варіант 6

Написати програму, у якій є два поля richTextBox та одне– textBox. В 1-ше поле richTextBox користувач вводить текст, в поле textBox вводить склад. В 2-ге поле richTextBox виводяться всі слова з цим складом.

Варіант 7

Написати програму, у якій є поле richTextBox та 2 поля textBox. В richTextBox користувач вводить текст, в поля textBox – слово з тексту та слово для заміни. Заміна робиться по натисненні кнопки «Замінити».

Варіант 8

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує найдовше слово у тексті та число його повторів у тексті.

Варіант 9

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує спочатку всі окличні речення, потім питальні, потім всі інші.

Варіант 7

Написати програму, у якій є поле richTextBox та 2 поля textBox. В richTextBox вводиться текст, а в I поле textBox – слово для пошуку. В II поле textBox виводиться скільки разів дане слово зустрічається в тексті.

Варіант 8

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує всі речення в тексті у зворотному порядку.

Варіант 9

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує введений текст міняючи у всіх словах першу і останню букви місцями.

Варіант 10

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує речення з введеного тексту, в якому найчастіше зустрічається буква «О».

Варіант 11

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує лише ті речення, в яких є цифри та кількість таких речень.

Варіант 12

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує лише ті речення, які починаються на слово з однієї букви.

Варіант 13

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує лише ті слова, які починаються і закінчуються на однакову літеру.

Варіант 14

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує замість букв – їх номер в алфавіті, а замість цифр – їх назву.

Варіант 15

Написати програму, у якій є два текстових поля (richTextBox). В 1 текстове поле користувач вводить текст, у 2-ге програма записує лише ті слова, кількість літер в яких – просте число.

Контрольні питання:

1) Як ініціалізувати рядкову змінну?
2) Які ви знаєте методи для роботи з рядками?
3) За допомогою яких операторів можна змінити регістр літер у рядку?
4) Що таке кодування Unicode?
5) Які засоби роботи з окремими символами надає C#?
6) Яке основне обмеження має клас string?
7) Як визначити кількість символів у рядку?
8) Чим відрізняються класи String і StringBuilder?
9) Як об'єднати два об'єкти StringBuilder?
10) Чому іноді для роботи з рядками краще використовувати клас StringBuilder?

Лабораторна робота № 4

Тема: Робота з графікою

Мета: Навчитися реалізувати необхідні класи графічних об'єктів

Теоретичні відомості

Графіка Windows Forms

У платформі .NET Framework реалізована вдосконалена система графічного програмування - GDI+. Вона розташовується в бібліотеці System.Drawing.dll.

Малювати на контексті пристрою дозволяє клас System.Drawing.Graphics.

Поняття контекст пристрою в Windows є концептуальним. Як графічний контекст можуть виступати: вікна, принтер, плоттер, графічний файл і т.д.

Узагальнене малювання GDI+ також включає в себе наступні основні поняття:

– колір (Color),

– перо (Pen) для малювання ліній,

– пензель (Brush) для заливання областей,

– шрифт (Font) для відображення тексту,

– точка (Point),

– розмір (Size),

– прямокутник (Rectangle) для задавання прямокутних областей відсікань,

– контур (GraphicsPath).

Наприклад, наступний код намалює лінію червоним пером шириною в один піксель:

```
Graphics g = ...//Одержання об'єкта Graphics.
g.DrawLine(Pens.Red, 10, 10, 200, 100);
```

Цей код буде однаково виконуватися для будь-якого графічного контексту, будь то принтер, форма або графічний файл. Наступні приклади дозволяють переконатися в незалежності цієї системи від пристроїв.

Приклад простої програми для малювання на pictureBox:

```
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 WindowsFormsApplication1
{
    public partial class Form1 : Form
```

```
    {
        public Form1()
        {
            InitializeComponent();
        }


// кнопка «Намалювати»

        private void button1_Click(object sender, EventArgs e)
        {
            Graphics g = pictureBox1.CreateGraphics();
            g.DrawLine(Pens.Red, 200, 200, 100, 20);
            g.DrawEllipse(Pens.Blue, 50, 50, 100, 150);
            g.DrawRectangle(Pens.Green, 200, 100, 102, 100);


         }
    }
}
```

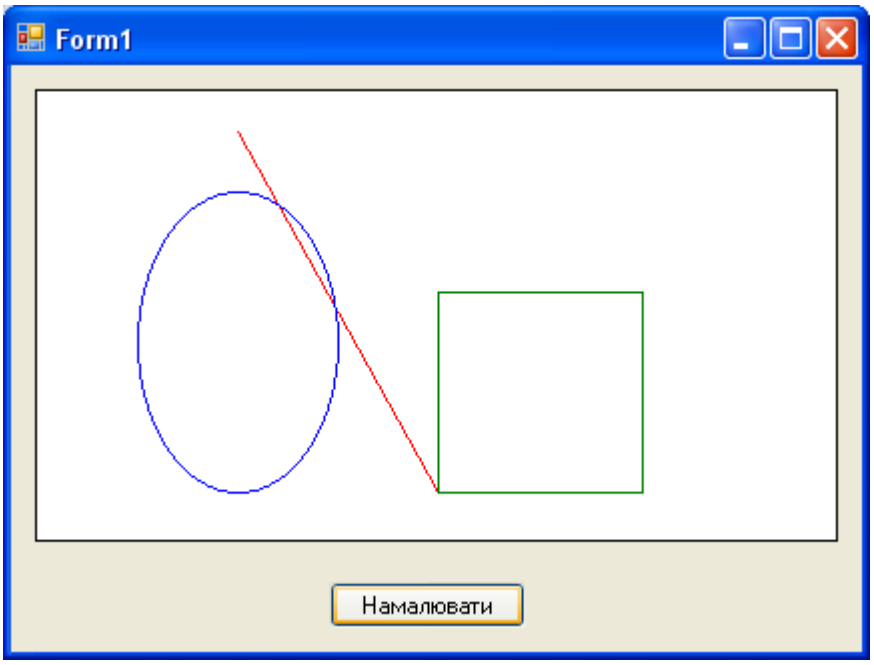


Рисунок 4.1 – Приклад скриншоту простої програми для виведення графіки

Колір

Кольори в Windows Forms засновані на моделі ARGB (red-green-blue). Кольори визначаються однобайтовими значеннями червоного, зеленого й синього. Альфа-канал визначає прозорість кольору. Значення alpha лежать у діапазоні від 0 для прозорого й до 255 для непрозорого. Для задання кольору необхідно вказати його складові або скористатися визначеними кольорами.

Таблиця 4.1. Приклади задання кольору

Приклад	Пояснення
`Color c = Color.FromArgb(127, 0, 255, 0);`	Зелений напівпрозорий колір

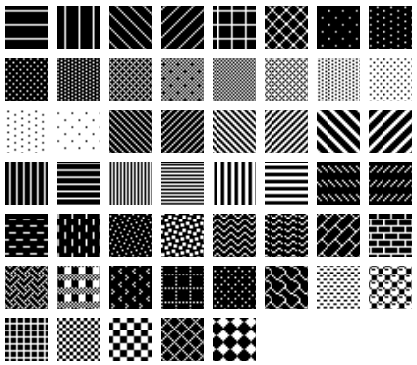
`Color c1 = Color.Green;`
`Color c2 = Color.Aqua;`
`...`
	Визначені кольори.
	Усього 141

`Color c1 = SystemColors.ActiveBorder;`
`Color c2 = SystemColors.MenuText;`
`...`
	Системні кольори.
	Усього 33

Системні кольори можна змінити в панелі керування Windows.

Пензель

Таблиця 4.2. Приклади заливання областей пензлем

Приклад	Пояснення
`Brush brush = new SolidBrush(Color.Red);`	Суцільний червоний пензель
`Brush brush1 = new LinearGradientBrush(point1, point2, c1, c2);`	Градієнтний пензель на основі двох точок і двох кольорів
`Brush brush2 = new HatchBrush(hs, c1, c2);`	Виконує штрихування заданими двома кольорами. Змінна hs має тип перерахування HatchStyle: 

Перо

Таблиця 4.3. Приклади малювання ліній або кривих пером

Приклад	Пояснення
`Pen pen = new Pen(c1);`	Перо зеленого кольору товщиною 1 піксель
`Pen pen1 = new Pen(c1, 5);`	Перо зеленого кольору товщиною 5 пікселів
`Pen pen2 = new Pen(brush1);`	Перо на основі градієнтного пензля

Шрифт

Таблиця 4.4. Приклади виведення тексту (основні властивості Name і Size)

Приклад	Пояснення
`Font font = new Font("Arial", 10);`	Шрифт Arial розміром 10

Точка, Розмір, Прямокутник

Точку на площині характеризує структура Point. Її основні властивості X і Y.

Структура Size призначена для завдання розміру. Вона містить властивості для висоти (Height) і ширини (Width).

Структуру Rectangle можна розглядати як об'єднання точки й розміру.

Методи класу Graphics для малювання

Функції малювання починаються із префікса Draw або Fill. Методи Draw малюють прямі й криві; методи Fill служать для заливання областей.

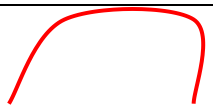
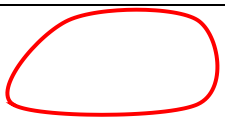
Нехай задані опорні точки, пензель, перо, шрифт:

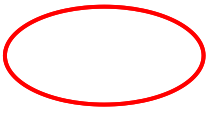
```
Pen pen = new Pen(Color.Red, 2);
int startAngle = 0;    //Початковий кут у градусах для малювання дуг.
int sweepAngle = -135; //Кінцевий кут для малювання дуг.
int x = 5, y = 5, width = 100, height = 50;
Rectangle rect = new Rectangle(x, y, width, height);
Point point = new Point(x, y);
Point pt1 = new Point(7, 50), pt2 = new Point(35, 7);
Point pt3 = new Point(100, 7), pt4 = new Point(100, 50);
Point[] points = new Point[] { pt1, pt2, pt3, pt4 };
Image image = Image.FromFile("H:\\PrintHS.png");//Завантаження зображення з файлу.
string s = "Текст";
Brush brush = new SolidBrush(Color.DarkGray);
Font font = new Font("Times New Roman", 14);
```

Таблиця 4.5. Приклади малювання фігур та деяких інших об’єктів

Приклад	Результат

Дуга
`g.DrawArc(pen, rect, startAngle, sweepAngle);`
або
`g.DrawArc(pen, x, y, width, height, startAngle, sweepAngle);`
Позитивний напрямок виміру дуги - за годинниковою стрілкою
	

Сплайн Безье `g.DrawBezier(pen, pt1, pt2, pt3, pt4);`	
Канонічний сплайн `g.DrawCurve(pen, points);`	
Замкнутий канонічний сплайн `g.DrawClosedCurve(pen, points);`	

Еліпс
`g.DrawEllipse(pen, rect);`
або
`g.DrawEllipse(pen, x, y, width, height);`
	

Зображення
`g.DrawImage(image, point);`
або
`g.DrawImage(image, x, y);`
	

Приклад	Результат

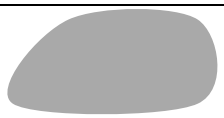
Лінія
`g.DrawLine(pen, pt1, pt3);`
або
`g.DrawLine(pen, pt1.X, pt1.Y, pt3.X, pt3.Y);`
	

Зчленовані лінії `g.DrawLines(pen, points);`	

Сектор
`g.DrawPie(pen, rect, startAngle, sweepAngle);`
або
`g.DrawPie(pen, x, y, width, height, startAngle, sweepAngle);`
	

Полігон `g.DrawPolygon(pen, points);`	

Прямокутник
`g.DrawRectangle(pen, rect);`
або
`g.DrawRectangle(pen, x, y, width, height);`
	

Текст `g.DrawString(s, font, brush, x, y);`	Текст
Заливання замкнутого канонічного сплайна `g.FillClosedCurve(brush, points);`	

Заливання еліпса
`g.FillEllipse(brush, rect);`
або
`g.FillEllipse(brush, x, y, width, height);`
	

Заливання сектора
`g.FillPie(brush, rect, startAngle, sweepAngle);`
або
`g.FillPie(brush, x, y, width, height, startAngle, sweepAngle);`
	

Заливання полігона `g.FillPolygon(brush, points);`	

Заливання прямокутника
`g.FillRectangle(brush, rect);`
або
`g.FillRectangle(brush, x, y, width, height);`
	

Контур

Контур - набір точок координат, що описує сукупність прямих і кривих. У контур можна додавати прямі й криві, використовуючи методи із префіксом Add. Ці методи аналогічні методам із префіксом Draw класу Graphics.

Таблиця 4.6. Приклади застосування контуру

Приклад	Результат

Відображення контуру
`GraphicsPath path = new GraphicsPath();`
`path.AddLine(pt1, pt2);`
`path.AddLine(pt3, pt4);`
`g.DrawPath(pen, path);`
	

Заливання контуру `g.FillPath(brush, path);`	

Завдання до лабораторної роботи:

Намалювати у вікні Windows-додатку, методами бібліотеки System.Drawing.dll, наступні об’єкти:

- прямокутник, залитий суцільним пензлем;
- еліпс, залитий суцільним пензлем із границею;
- коло, залите суцільним пензлем із границею;
- будиночок, залитий суцільною кистю із границею (висота даху не менше 50 пікселів);
- заштрихований будиночок;
- коло, зафарбоване чорно-білими квадратами;
- квітку, що складається з жовтого кола та блакитних еліпсів;
- конверт, залитий суцільною кистю із границею (висота кришки конверта не менше 30 пікселів, відношення ширини до висоти 5 до 3, ширина кратна 5);
- заштрихований конверт.

Контрольні питання:

1. В який бібліотеці C# розташовані засоби роботи з графікою?
2. Які основні поняття включає в себе малювання в C#?
3. Для чого служить інструмент Пензель?
4. Яким чином можна намалювати прямокутник?
5. Яким чином можна намалювати еліпс?

Лабораторна робота № 5

Тема: Робота з файлами

Мета: Дослідити файлове введення/виведення.

Теоретичні відомості

System.IO містить всі необхідні класи, методи і властивості для маніпуляцій з каталогами та файлами (У таблиці 5.1 наведено основні класи).

Таблиця 5.1. Класи в System.IO

Клас	Призначення
BinaryReader та Writer	Читання і запис простих типів даних
Directory, File, DirectoryInfo та FileInfo	Створення, видалення та переміщення файлів і директорій. Отримання докладної інформації про файли, за допомогою властивостей, визначених у цих класах.
FileStream	Доступ до файлів потоковим способом
MemoryStream	Доступ до даних що зберігаються в пам'яті
StreamWriter та StreamReader	Читання і запис текстової інформації
StringReader та StringWriter	Читання і запис текстової інформації з рядкового буфера

Робота з класами DirectoryInfo і FileInfo

Класи DirectoryInfo і FileInfo успадковані від FileSystemInfo, який є абстрактним. Це означає, що Ви не можете успадкувати від нього свій клас, але можете використовувати властивості, визначені у ньому. Таблиця 5.2 перераховує його властивості та методи.

Таблиця 5.2. Властивості та методи класу FileSystemInfo

Властивість	Значення
Attributes	Повертає атрибути файлу у вигляді значень перерахування FileAttributes
CreationTime	Повертає час створення файлу
Exists	Перевіряє чи є файл директорією чи ні
Extension	Повертає розширення файлу
LastAccessTime	Повертає час останнього доступу до файлу
FullName	Повертає повний шлях до файлу
LastWriteTime	Повертає час останньої зміни файлу
Name	Повертає ім'я даного файлу
Delete()	Видаляє файл. Будьте обережні при використанні цього методу.

Клас DirectoryInfo містить методи для створення, переміщення і видалення каталогів. Щоб використовувати вищенаведені властивості, необхідно створити об'єкт класу DirectoryInfo як показано в прикладі:

```
DirectoryInfo dir1 = new DirectoryInfo(@"F:\WINNT");
```

Після цього вже можна переглянути властивості директорії за допомогою об'єкта dir1, як показано у фрагменті коду:

```
Console.WriteLine("Повне ім’я : {0}", dir1.FullName);
Console.WriteLine("Атрибути : {0}", dir1.Attributes.ToString());
```

Можна також використовувати значення перерахування FileAttributes. Вони наведені в таблиці 5.3.

Таблиця 5.3. СписокFileAttributes

Властивість	Значення
Archive	Повертає Архівний статус файлу
Compressed	Дозволяє дізнатися стиснутий файл чи ні
Directory	Показує чи є файл директорією чи ні
Encrypted	Показує закодований файл чи ні
Hidden	Показує прихований файл чи ні
Offline	Показує, що дані відсутні
ReadOnly	Показує чи є файл тільки для читання
System	Показує, чи є файл системним (можливо файл в директорії Windows)

Робота з файлами в директорії

Припустимо, ви хочете отримати список всіх файлів з розширенням BMP в папці F: \ Pictures. Для цього можна використовувати наступний код:

```
DirectoryInfodir = new DirectoryInfo(@"F:\WINNT");
FileInfo[] bmpfiles = dir.GetFiles("*.bmp);
Console.WriteLine("Загальна кількість Bmp файлів", bmpfiles.Length);
Foreach( FileInfo f inbmpfiles)
{
Console.WriteLine("Ім’я : {0}", f.Name);
Console.WriteLine("Довжина файлу : {0}", f.Length);
Console.WriteLine("Час творення : {0}", f.CreationTime);
Console.WriteLine("Атрибути файлу : {0}",
f.Attributes.ToString());
}
```

Створення підкаталогів

Наступний фрагмент коду описує як можна створити піддиректорію MySub вдиректорії Sub:

```
DirectoryInfodir = newDirectoryInfo(@"F:\WINNT");
try
```

```
{
dir.CreateSubdirectory("Sub");
dir.CreateSubdirectory(@"Sub\MySub");
}
catch(IOException e)
{
Console.WriteLine(e.Message);
}
```

Створення файлів за допомогою класу FileInfo

Клас FileInfo дозволяє створювати нові файли, одержувати інформацію, видаляти і переміщати їх. У цьому класі також є методи для відкриття, читання і запису в файл. У наступному прикладі показано, як можна створити текстовий файл і отримати доступ до його інформації (часу його створення, повне ім'я, і так далі):

```
FileInfofi = new FileInfo(@"F:\Myprogram.txt");
FileStreamfstr = fi.Create();
Console.WriteLine("Час створення: {0}",f.CreationTime);
Console.WriteLine("Повне ім’я: {0}",f.FullName);
Console.WriteLine("Атрибути файлу: {0}",f.Attributes.ToString());
//Видалення файлу Myprogram.txt.
Console.WriteLine("Натисніть будь-яку клавішу, щоб видалити файл");
Console.Read();
fstr.Close();
fi.Delete();
```

Опис методу Open()

У класі FileInfo є метод під назвою Open() за допомогою якого можна створювати файли, підставляючи в параметри значення перерахувань FileMode і FileAccess. Наступний фрагмент коду показує, як це робиться:

```
FileInfo f = new FileInfo("c:\myfile.txt");
FileStream s = f.Open(FileMode.OpenorWrite, FileAccess.Read);
```

Після цього, використовуючи об'єкт 's', можна читати і записувати в файл. У перевантаженому методі Open() можна тільки читати з файлу. Для запису в файл необхідно в параметрах відкриття використовувати значення FileAccess.ReadWrite. Таблиці 5.4 і 5.5 містять можливі значення FileMode і FileAccess.

Таблиця 5.4. Значення FilrMode

Значення	Призначення
Append	Для відкриття файлу і додавання даних. Використовується спільно зі значенням FileAccess.Write.
Create	Для створення нового файлу. Якщо файл вже існує, то він затирається.
CreateNew	Для створення нового файлу. Якщо файл існує, то виникає виключення IOException.

Значення	Призначення
Open	Для відкриття існуючого файлу
OpenOrCreate	Для відкриття існуючого або створення нового файлу. Якщо файл не існує, то буде створено новий.
Truncate	Для урізання існуючого файлу

Таблиця 5.5. Значення FilrAccess

Значення	Призначення
Read	Для читання (отримання) даних з файлу
ReadWrite	Для запису в або читання з файлу
Write	Для запису даних у файл

Запис в текстовий файл за допомогою класу StreamWriter

Текстові дані або будь-яку іншу інформацію можна записати у файл використовуючи метод CreateText() в класі FileInfo. Однак попередньо необхідно отримати валідний StreamWriter. Саме StreamWrite забезпечує необхідну функціональність для запису у файл. Наступний приклад ілюструє це:

```
FileInfo f = new FileInfo("Mytext.txt")
StreamWriter w = f.CreateText();
w.WriteLine("Це з");
w.WriteLine("Глава 6");
w.WriteLine("C# Модуль");
w.Write(w.NewLine);
w.WriteLine("Дякуємо за очікування");
w.Close();
```

Читання з текстового файлу

Для читання з текстового файлу можна скористатися класом StreamReader. Для цього необхідно вказати ім'я файлу в статичному методі OpenText() класу File. Наступний приклад зчитує вміст файлу, яке було записано в попередньому прикладі:

```
Console.WriteLine("Читання вмісту з файлу");
StreamReader s = File.OpenText("Mytext.txt");
stringread = null;
while ((read = s.ReadLine()) != null)
{
Console.WriteLine(read);
}
s.Close();
```

Робота з різними кодуваннями

За замовчуванням в .NET всі текстові дані в кодуванні UTF8, але часто потрібно зчитати текстовий файл, збережений в іншому кодуванні, допустимо в

WIN1251. У такому випадку якщо у файлі був український текст в кодуванні WIN1251, при зчитуванні його в UTF8 ми отримаємо нечитаємі дані. Для того щоб переводити рядки з одного кодування в інше існує клас Encoding з простору імен System.Text, завдяки якому знаючи вихідну кодування ми можемо привести текстові дані до потрібного кодування. Розглянемо наступний приклад, наприклад, ми хочемо конвертувати зчитаний рядок в кодуванні WIN1251 в кодування DOS (866), для цього можна використовувати наступний код:

```
classProgram
    {
privatestaticstring in1251;
privatestaticreadonlyEncoding enc1251 = Encoding.GetEncoding(1251);
privatestaticreadonlyEncoding enc866 = Encoding.GetEncoding(866);
staticvoidMain(string[] args)
{
            //....
            //тут якимось чином отримуємо дані в in1251

byte[] sourceBytes = enc1251.GetBytes(in1251);
stringoutputString = enc866.GetString(sourceBytes);
            //даліробимоте, щонеобхіднозотриманимрядком
//....
        }
    }
```

У sourceBytes ми отримали вхідний рядок у вигляді масиву байт, які далі можна так само за допомогою Encoding зберегти в рядок з потрібним кодуванням. У випадку, коли ми читаємо текст з файлу з уже відомим кодуванням, справи йдуть ще простіше. При створенні екземпляра StreamReader можна явно вказати кодування джерела, код буде виглядати наступним чином:

```
using (varsr = new StreamReader("Mytext.txt", Encoding.GetEncoding(1251)))
     {
stringread = null;
while ((read = sr.ReadLine()) != null)
 Console.WriteLine(read);
     }
```

При записі у файл із належним кодуванням для StreamWriter так само можна вказати кодування, в якому буде текст.

Приклад роботи з файлами в додатках Windows Forms

Розмістіть на форму програми 4 кнопки:

- button1, у її властивість Text запишіть «Створити файл».
- button2, у її властивість Text запишіть «Видалити файл».
- button3, у її властивість Text запишіть «Записати текст у файл».

– button4, у її властивість Text запишіть «Прочитати файл».

Додайте на форму текстове поле textBox, встановіть його властивість Multiline в true (ввімкнення багаторядкового режиму)

Приклад форми, яка повинна вийти, зображений на рис. 5.1.

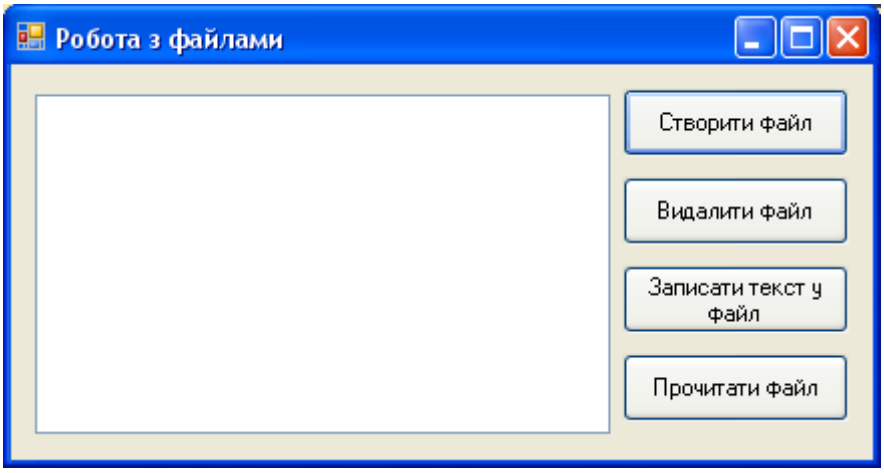


Рисунок 5.1 – Приклад програми для роботи з файлами

Приклад коду програми:

```
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;
using System.IO; //Підключаємо простір імен для роботи з файлами

namespace Files
{
publicpartialclass Form1 : Form
    {
public Form1()
        {
            InitializeComponent();
            FileInfo file = new FileInfo("book.txt");
        }


// Створення файлів
privatevoid button1_Click(object sender, EventArgs e)
        {
            FileInfo file = new FileInfo("book.txt");
if (file.Exists == false) //Якщо файл не існує
            {
                file.Create(); //Створюємо
            }
else MessageBox.Show("Файл уже існує!");
```

```
        }

    //Видалення файлу
    privatevoid button2_Click(object sender, EventArgs e)
        {
            FileInfo file = new FileInfo("book.txt");
    if (file.Exists == true) //Якщо файл існує
            {
                file.Delete(); //Видаляємо
            }
    else MessageBox.Show("Такого файлу не існує!");
        }

    // Запис тексту у файл
    privatevoid button3_Click(object sender, EventArgs e)
        {
            StreamWriter write_text;  //Клас для запису у файл
            FileInfo file = new FileInfo("book.txt");
            write_text = file.AppendText(); //Дописуємо дані у файл, якщо
файлу не існує він створиться
            write_text.WriteLine(textBox1.Text); //Записуємо у файл текст
із текстового поля
            write_text.Close(); // Закриваємо файл
        }

    // Читання з файлу
    privatevoid button4_Click(object sender, EventArgs e)
        {
            StreamReader streamReader = new StreamReader("book.txt");
//Відкриваємо файл для читання
    string str = ""; //Повідомляємо змінну, у яку будемо записувати текст із
файлу

    while (!streamReader.EndOfStream) //Цикл триватиме поки не буде досягнутий
кінець файлу
            {
                str += streamReader.ReadLine(); //У змінну str по рядках
записуємо вміст файлу
            }

            textBox1.Text = str;
        }
    }
}
```

Завдання до лабораторної роботи:

Варіант 1

Створити файл, заповнити його випадковими числами. Вивести вміст файлу на екран.

Варіант 2

Створити файл, заповнити його випадковими цілими числами. Обчислити середнє арифметичне чисел, що перебувають у файлі, та дописати це значення в кінець файлу.

Варіант 3

Потрібно створити файл заповнений цілими числами, у якому значення кожного елемента x_{i+1} дорівнює $(x_i)^2$ і квадрати всіх чисел не перевершують n.

Варіант 4

Створити два файли заповнені відсортованими за зростанням цілими числами. Створити третій файл та записати в нього числа з першого та другого. Числа в третьому файлі також повинні розташовуватися за зростанням.

Варіант 5

Створити файл, заповнити його цілими числами. Знайти найбільше число у файлі вивести його на екран та записати у кінець файлу.

Варіант 6

Створити файл, записати в нього рядки, що містять прізвища власників і номери їхніх телефонів. Реалізувати можливість пошуку номера телефона у файлі за прізвищем власника (користувач вводить у перше текстове поле на формі прізвище, програма виводить в друге текстове поле знайдений у файлі номер телефону, якщо прізвище не знайдено, виводиться повідомлення про відсутність номеру телефону).

Варіант 7

Створити файл, заповнити його випадковими числами. Створити дві копії даного файлу. Одну з копій видалити. В іншій копії знайти мінімальне число та вивести його на екран.

Варіант 8

Створити програму для підрахунку кількості чисел у файлі, менших середнього арифметичного всіх елементів цього файлу.

Варіант 9

Створити програму, що міняє місцями перший і останній символи у вказаному файлі.

Варіант 10

Створити текстовий файл. Записати в нього текст, введений у текстове поле форми програми. Створити другий текстовий файл, у який записати

кількість букв «А», «Б» та «В» у вказаному файлі. Вивести на екран максимальне число з другого файлу та до якої букви воно відноситься.

Варіант 11

Створити файл, заповнити його випадковими дійсними числами, обчислити суму елементів цього файлу та вивести її на екран.

Варіант 12

Створити файл, заповнити його випадковими буквами. Вказати в одному текстовому полі форми програми букву, а в трогуму цифру. Створити другий файл, у якому кожний елемент, що являється вказаною буквою замінити на вказану цифру.

Варіант 13

Створити текстовий файл. Заповнити його буквами та цифрами. Прочитати створений файл і знайти суму цифр, що зустрічаються в ньому.

Варіант 14

Створити файл заповнений випадковими позитивними та негативними числами. Прочитати вміст файлу та створити ще два файли, в перший записати всі позитивні числа, в другий – всі негативні.

Варіант 15

Створити текстовий файл. Заповнити його текстом. Прочитати створений файл і вивести його вміст на екран задом наперед.

Контрольні запитання:

1) В якому просторі імен зберігаються методи для роботи з файлами та каталогами?
2) Який метод необхідно використати для створення нового файлу та генерування виключення, якщо такий файл вже існує?
3) Для чого використовуються класи StringReader та StringWriter?
4) За допомогою якого класу можна отримати доступ до даних, що зберігаються в пам’яті?

Лабораторна робота № 6

Тема: Особливості об'єктно-орієнтованого програмування в C#

Мета: Одержання практичних навичок по створенню й використанню класів у мові C#.

Теоретичні відомості

Класи

Клас є типом даних, визначеним користувачем. Він повинен являти собою одну логічну сутність, наприклад, бути моделлю реального об'єкта або процесу. *Елементами* класу є *дані* й *функції,* призначені для їхньої обробки.

Опис класу містить ключове слово class, за яким слідує його *ім'я,* а далі у фігурних дужках - *тіло* класу, тобто список його елементів. Крім того, для класу можна задати його базові класи (предки) і ряд необов'язкових атрибутів і специфікаторів, що визначають різні характеристики класу:

```
[ атрибути ] [ специфікатори ] class ім'я_класу [ : предки ]
{
    тіло-класу
}
```

Ім'я_класу задається програмістом за загальними правилами C#. *Тіло класу* – це список описів його елементів, поміщений у фігурні дужки. Список може бути порожнім, якщо клас не містить жодного елемента.

Специфікатори визначають властивості класу, а також доступність класу для інших елементів програми.

У даній лабораторній роботі ми будемо розглядати класи, які описуються в просторі імен безпосередньо. Для таких класів допускаються тільки два специфікатори: public і internal. За замовчуванням, тобто якщо жоден специфікатор доступу не зазначений, мається на увазі специфікатор internal.

Клас є узагальненим поняттям, що визначає характеристики й поводження деякої множини конкретних об'єктів цього класу, що називаються *екземплярами* або *об'єктами класу.*

Об'єкти створюються явним або неявним чином, тобто або програмістом, або системою. Програміст створює екземпляр класу за допомогою операції new, наприклад:

```
Demo a = new Demo();// створення екземпляра класу Demo
Demo b = new Demo();//створення іншого екземпляра класу Demo
```

Клас відноситься до посилальних типів даних, пам'ять під які виділяється в купі. Таким чином, змінні a і b зберігають не самі об'єкти, а посилання на об'єкти, тобто їхні адреси. Якщо достатній для зберігання об'єкта обсяг пам'яті виділити не вдалося, операція new генерує виключення OutOfMemoryException.

Механізм виконання присвоювання однаковий для величин будь-якого типу, як посилального, так і значимого, однак результати розрізняються. При присвоюванні значень копіюються значення, а при присвоюванні посилань - посилання, тому після присвоювання одного об'єкта іншому ми отримаємо два посилання, що вказують на одну й ту саму область пам'яті.

Аналогічна ситуація з операцією перевірки на рівність. Величини значимого типу рівні, якщо рівні їхні значення. Величини посилального типу (екземпляри класу в цьому випадку) рівні, якщо вони посилаються на ті самі дані.

Для кожного об'єкта при його створенні в пам'яті виділяється окрема область, у якій зберігаються ці дані. Крім того, у класі можуть бути присутніми *статичні елементи,* які існують у єдиному екземплярі для всіх об'єктів класу. Часто статичні дані називають *даними класу,* а інші - *даними екземпляра.*

Функціональні елементи класу не тиражуються, тобто завжди зберігаються в єдиному екземплярі. Для роботи з даними класу використовуються *методи класу (статичні методи),* для роботи з даними екземпляра – *методи екземпляра,* або просто *методи.*

Поля й методи є основними елементами класу. Крім того, у класі можна задавати інші елементи. Нижче наведені всі можливі елементи класу:

– *Константи* класу зберігають незмінні значення, пов'язані із класом.

– *Поля* містять дані класу.

– *Методи* реалізують обчислення або інші дії, виконувані класом або екземпляром.

– *Властивості* визначають характеристики класу в сукупності зі способами їхнього завдання й одержання, тобто методами запису й читання.

– *Конструктори* реалізують дії по ініціалізації екземплярів або класу в цілому.

– *Деструктори* визначають дії, які необхідно виконати до того, як об'єкт буде знищений.

– *Індексатори* забезпечують можливість доступу до елементів класу за їхнім порядковим номером.

– *Операції* задають дії з об'єктами за допомогою знаків операцій.

– *Події* визначають повідомлення, які може генерувати клас.

– *Типи* – це типи даних, внутрішні стосовно класу.

Ключове слово this

Кожний об'єкт містить свій екземпляр полів класу. Методи перебувають у пам'яті в єдиному екземплярі й використовуються всіма об'єктами спільно, тому необхідно забезпечити роботу методів нестатичних екземплярів з полями саме того об'єкта, для якого вони були викликані. Для цього в будь-який нестатичний метод автоматично передається прихований параметр this, у якому зберігається посилання на екземпляр, що викликав функцію.

У явному вигляді параметр this застосовується для того, щоб повернути з методу посилання на викликавший об'єкт, а також для ідентифікації поля у випадку, якщо його ім'я збігається з ім'ям параметра методу, наприклад:

```
class Demo
{
   double y;
   public Demo T() // метод повертає посилання на екземпляр
   {
return this;
   }
public void Sety( double y)
   {
      this.y = y; // полю y привласнюється значення параметра y
   }
```

Конструктори

Конструктор призначений для ініціалізації об'єкта. Він викликається автоматично при створенні об'єкта класу за допомогою операції new. Ім'я конструктора збігається з ім'ям класу. Нижче перераховані властивості конструкторів:

– Конструктор *не повертає значення,* навіть типу void.

– Клас може мати *декілька конструкторів* з різними параметрами для різних видів ініціалізації.

– Якщо програміст не вказав жодного конструктора або якісь поля не були ініціалізовані, полям значимих типів привласнюється нуль, полям посилальних типів - значення null.

– Конструктор, викликаний без параметрів, називається *конструктором за замовчуванням.*

Можна задавати початкові значення полів класу при описі класу. Це зручно в тому випадку, коли для всіх екземплярів класу початкові значення якогось поля однакові. Якщо ж при створенні об'єктів потрібно привласнювати полю різні значення, це варто робити в конструкторі. У наступному прикладі в клас Demo доданий конструктор, а поля зроблені закритими (непотрібні в цей момент елементи опущені). У програмі створюються два об'єкти з різними значеннями полів.

Лістинг 6.1. Клас із конструктором

```
using System;
namespace ConsoleApplication1
{
class Demo
    {
public Demo(int a, double y) // конструктор з параметрами
        {
this.a = a;
this.y = y;
```

```
        }
publicdouble Gety() // метод одержання поля y
        {
return y;
        }
int a;
double y;
    }

class Classl
    {
staticvoid Main()
        {
Demo a = new Demo(300, 0.002); // виклик конструктора
Console.WriteLine(a.Gety());   // результат: 0.002
Demo b = new Demo(1, 5.71);    // виклик конструктора
Console.WriteLine(b.Gety());   // результат: 5.71
        }
    }
}
```

Часто буває зручно задати в класі *декілька конструкторів,* щоб забезпечити можливість ініціалізації об'єктів різними способами.

Всі конструктори повинні мати різні сигнатури.

Якщо один з конструкторів виконує які-небудь дії, а інший повинен робити те ж саме плюс ще що-небудь, зручно *викликати перший конструктор із другого.* Для цього використовується вже згадане ключове слово this в іншому контексті, наприклад:

```
class Demo
{
    public Demo( int a ) // конструктор 1
    {
        this.a = a;
    }
    public Demo(int a, double y) : this( a ) // виклик конструктора 1
    {
        this.y = y;
    }
    ...
}
```

Конструкція, що перебуває після двокрапки, називається *ініціалізатором,* тобто тим кодом, що виконується до початку виконання тіла конструктора.

Конструктор будь-якого класу, якщо не зазначений ініціалізатор, автоматично викликає конструктор свого предка. Це можна зробити і явно за допомогою ключового слова base, що позначає конструктор базового класу.

Конструктор базового класу викликається явно в тих випадках, коли йому потрібно передати параметри.

Властивості

Властивості служать для організації доступу до полів класу. Як правило, властивість пов'язана із закритим полем класу й визначає методи його одержання й встановлення. Синтаксис властивості:

```
[ атрибути ] [ специфікатори ] тип ім'я_властивості
{
    [ [ атрибути ] [ специфікатори ] get код_доступу ]
    [ [ атрибути ] [ специфікатори ] set код_доступу ]
}
```

Значення специфікаторів для властивостей і методів аналогічні. Найчастіше властивості оголошуються як відкриті (зі специфікатором public), оскільки вони входять в інтерфейс об'єкта.

Код доступу являє собою блоки операторів, які виконуються при одержанні (get) або встановленні (set) властивості. Може бути відсутньою або частина get, або set, але не обидві одночасно.

Якщо відсутня частина set, властивість доступна тільки для читання (read-only), якщо відсутня частина get, властивість доступна тільки для запису (write-only).

Специфікатори доступу для окремої частини повинні задавати або такий же, або більш обмежений доступ, ніж специфікатор доступу для властивості в цілому. Наприклад, якщо властивість описана як public, його частини можуть мати будь-який специфікатор доступу, а якщо властивість має доступ protected internal, його частини можуть оголошуватися як internal, protected або private.

Приклад опису властивостей:

```
public class Button: Control
{
    private string caption; // закрите поле, з яким зв'язана властивість
    public string Caption { // властивість
        get { // спосіб одержання властивості
            return caption;
        }
        set { // спосіб встановлення властивості
            if (caption != value) {
                caption = value;
            }
        }
    }
    ...
}
```

Метод запису звичайно містить дії по перевірці допустимості встановлюваного значення, метод читання може містити, наприклад, підтримку лічильника звертань до поля.

У програмі властивість виглядає як поле класу, наприклад:

```
Button ok = new Button();
ok.Caption = "OK";     // викликається метод встановлення властивості
string s = ok.Caption; // викликається метод одержання властивості
```

При звертанні до властивості автоматично викликаються зазначені в ньому методи читання й встановлення.

Синтаксично читання й запис властивості виглядають майже як методи. Метод get повинен містити оператор return, що повертає вираз, для типу якого повинне існувати неявне перетворення до типу властивості. У методі set використовується параметр зі стандартним ім'ям value, що містить встановлюване значення.

Властивість може не зв'язуватися з полем. Фактично, вона описує один або два методи, які здійснюють деякі дії над даними того ж типу, що й властивість. На відміну від відкритих полів, *властивості забезпечують розділення між внутрішнім станом об'єкта і його інтерфейсом* і, таким чином, спрощують внесення змін у клас.

Приклад створення класу

Лістинг 6.2. Клас Man

```
using System;
namespace ConsoleApplication1
{
class Man
    {
public Man()
        {
this.name = "John Doe";
this.height = 180;
this.age = 20;
        }

public Man( string name ) : this()
        {
this.name = name;
        }

public Man( int height, int age, string name )
        {
this.name = name;
this.height = height;
this.age = age;
        }

publicstring GetName()
        {
return name;
        }

publicint GetHeight()
        {
```

```
    return height;
            }

    publicvoid SetHeight(int height)
            {
    this.height = height;
            }

    publicint Age
            {
                get
                {
    return age;
                }
                set
                {
                    age = value;
                }
            }

    publicvoid Passport()
            {
    Console.WriteLine("Ім'я: {0} \t Зріст = {1} Вік= {2} ", name, height,
age );
            }

    // закриті поля
    string name;
    int height, age;
        }

    class Classl
        {
    staticvoid Main()
            {
    Man X = new Man();
                X.Passport();
    Man Ivan = new Man( "Іван" );
                Ivan.Passport();
    Man Masha = new Man( 201, 18, "Маша" );
                Masha.Passport();
                Ivan.Age = 21;
                Ivan.Passport();
            }
        }
    }
```

Результат роботи програми:

```
Ім'я: John Doe    Зріст = 180 Вік = 20
Ім'я: Іван        Зріст = 180 Вік = 20
Ім'я: Маша        Зріст = 201 Вік = 18
Ім'я: Іван        Зріст = 180 Вік = 21
```

У класі три закриті поля (name, height і age), чотири методи (GetName, GetHeight, SetHeight і Passport), одна властивість (Age) і три конструктори, що дозволяють задати при створенні об'єкта жодного, один або три параметри.

Завдання до лабораторної роботи:

Описати клас, заданий за варіантом, та реалізувати роботу з ним. Кожний розроблювальний клас повинен, як правило, містити наступні елементи: приховані поля, конструктори з параметрами й без параметрів, методи, властивості. Методи й властивості повинні забезпечувати несуперечливий, повний, мінімальний і зручний інтерфейс класу. У програмі повинна виконуватися перевірка всіх розроблених елементів класу.

Варіант 1

Описати клас, який реалізує десятковий лічильник, що може збільшувати або зменшувати своє значення на одиницю в заданому діапазоні. Передбачити ініціалізацію лічильника значеннями за замовчуванням і довільними значеннями. Лічильник має два методи: збільшення й зменшення, - і властивість, що дозволяє одержати його поточний стан. При виході за межі діапазону викидаються виключення.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 2

Описати клас, що реалізує шістандцятковий лічильник, що може збільшувати або зменшувати своє значення на одиницю в заданому діапазоні. Передбачити ініціалізацію лічильника значеннями за замовчуванням і довільними значеннями. Лічильник має два методи: збільшення й зменшення, - і властивість, що дозволяє одержати його поточний стан. При виході за межі діапазону викидаються виключення.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 3

Описати клас, що представляє трикутник. Передбачити методи для створення об'єктів, переміщення на площини, зміни розмірів і обертання на заданий кут. Описати властивості для одержання стану об'єкта. При неможливості побудови трикутника викидається виключення.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 4

Побудувати опис класу, що містить інформацію про поштову адресу організації. Передбачити можливість роздільної зміни складових частин адреси

й перевірки допустимості значень, що вводяться. У випадку неприпустимих значень полів викидаються виключення.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 5

Скласти опис класу для представлення комплексних чисел. Забезпечити виконання операцій додавання, віднімання й множення комплексних чисел.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 6

Скласти опис класу для вектора, заданого координатами його кінців у тривимірному просторі. Забезпечити операції додавання й віднімання векторів з одержанням нового вектора (суми або різниці), обчислення скалярного добутку двох векторів, довжини вектора, косинуса кута між векторами.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 7

Скласти опис класу прямокутників зі сторонами, паралельними осям координат. Передбачити можливість переміщення прямокутників на площини, зміну розмірів, побудова найменшого прямокутника, що містить два заданих прямокутники, і прямокутника, що є спільною частиною (перетином) двох прямокутників.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 8

Скласти опис класу для представлення дати. Передбачити можливості встановлення дати й зміни її окремих полів (рік, місяць, день) з перевіркою допустимості значень, що вводяться. У випадку неприпустимих значень полів викидаються виключення. Створити методи зміни дати на задану кількість днів, місяців і років.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 9

Скласти опис класу для представлення часу. Передбачити можливості встановлення часу й зміни його окремих полів (година, хвилина, секунда) з перевіркою допустимості значень, що вводяться. У випадку неприпустимих значень полів викидаються виключення. Створити методи зміни часу на задану кількість годин, хвилин і секунд.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 10

Скласти опис класу багаточлена виду $ax^2 + bx + c$. Передбачити методи, що реалізують:

– обчислення значення багаточлена для заданого аргументу;

– операцію додавання, віднімання й множення багаточленів з одержанням нового об'єкта-багаточлена;

– виведення на екран опису багаточлена.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 11

Описати клас, що представляє трикутник. Передбачити методи для створення об'єктів, обчислення площі, периметра й точки перетину медіан. Описати властивості для одержання стану об'єкта. При неможливості побудови трикутника викидається виключення.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 12

Описати клас, що представляє коло. Передбачити методи для створення об'єктів, обчислення площі кола, довжини окружності й перевірки влучення заданої точки усередину кола. Описати властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 13

Описати клас для роботи з рядком, що дозволяє зберігати тільки двійкове число й виконувати з ним арифметичні операції. Передбачити ініціалізацію з перевіркою допустимості значень. У випадку неприпустимих значень викидаються виключення.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 14

Описати клас дробних (раціональних) чисел, що є відношенням двох цілих чисел. Передбачити методи додавання, віднімання, множення й ділення дробів.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 15

Описати клас «файл», що містить відомості про ім'я, дату створення й довжині файлу. Передбачити ініціалізацію з перевіркою допустимості значень полів. У випадку неприпустимих значень полів викидаються виключення.

Описати метод додавання інформації в кінець файлу й властивості для одержання стану файлу.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 16

Описати клас «кімната», що містить відомості про метраж, висоту стель і кількість вікон. Передбачити ініціалізацію з перевіркою допустимості значень полів. У випадку неприпустимих значень полів викидаються виключення. Описати методи обчислення площі й обсягу кімнати й властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 17

Описати клас, що представляє нелінійне рівняння виду:

$ax - \cos(x) = 0$.

Описати метод, що обчислює рішення цього рівняння на заданому інтервалі методом ділення навпіл і викидаючий виключення у випадку відсутності кореня. Описати властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 18

Описати клас, що представляє квадратне рівняння виду:

$ax^2 + bx + c = 0$.

Описати метод, що обчислює рішення цього рівняння й викидає виключення у випадку відсутності коренів. Описати властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класу.

Варіант 19

Описати клас «процесор», що містить відомості про марку, тактову частоту, обсяг кешу й вартості. Передбачити ініціалізацію з перевіркою допустимості значень полів. У випадку неприпустимих значень полів викидаються виключення. Описати властивості для одержання стану об'єкта.

Описати клас «материнська плата», що включає клас «процесор» і обсяг встановленої оперативної пам'яті. Передбачити ініціалізацію з перевіркою допустимості значень поля обсягу пам'яті. У випадку неприпустимих значень поля викидається виключення. Описати властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класів.

Варіант 20

Описати клас «кольорова точка». Для точки задаються координати й колір. Колір описується за допомогою трьох складових (червоний, зелений, синій). Передбачити різні методи ініціалізації об'єкта з перевіркою допустимості значень. Припустимим діапазоном для кожної складової є [0, 255]. У випадку неприпустимих значень полів викидаються виключення. Описати властивості для одержання стану об'єкта й метод зміни кольору.

Написати програму, що демонструє всі розроблені елементи класу.

Контрольні питання:

1. Як визначити клас?
2. Як створити екземпляр класу?
3. Які елементи можуть входити до складу класу?
4. Як звернутися до елемента класу?
5. Для чого використовується ключове слово this?
6. Для чого застосовуються конструктори?
7. Як визначається, який з конструкторів буде викликаний при створенні об'єкта?
8. Що таке статичні методи класу?

Лабораторна робота № 7

Тема: Інтерфейс користувача та елементи керування

Мета: Навчитися створювати динамічні додатки, що працюють за принципами керування подіями

Теоретичні відомості

Створення динамічних додатків на мові C#

Використовуючи елемент управління pictureBox, створимо просту гру "Перегони жуків". Суть гри полягає в наступному: перемагає той, чий жук прибіжить до фінішу першим. Для гри потрібні два рисунки жучків, один буде червоним, інший – синім. Рисунки можна намалювати, наприклад, в редакторі Paint, або завантажити до себе в папку із проектом представлені тут рисунки:

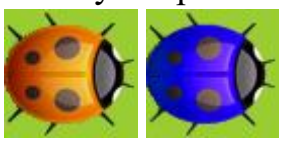


Рисунок 7.1 – Зображення жуків для гри «Перегони жуків»

Створивши новий проект, варто розташувати на формі наступні елементи управління (рис. 7.2): 1) textBox1, textBox2; 2)label1, label2; 3)button1, button2; 4) panel1; 5) pictureBox1, pictureBox2; 6)menuStrip1; 7) timer1.

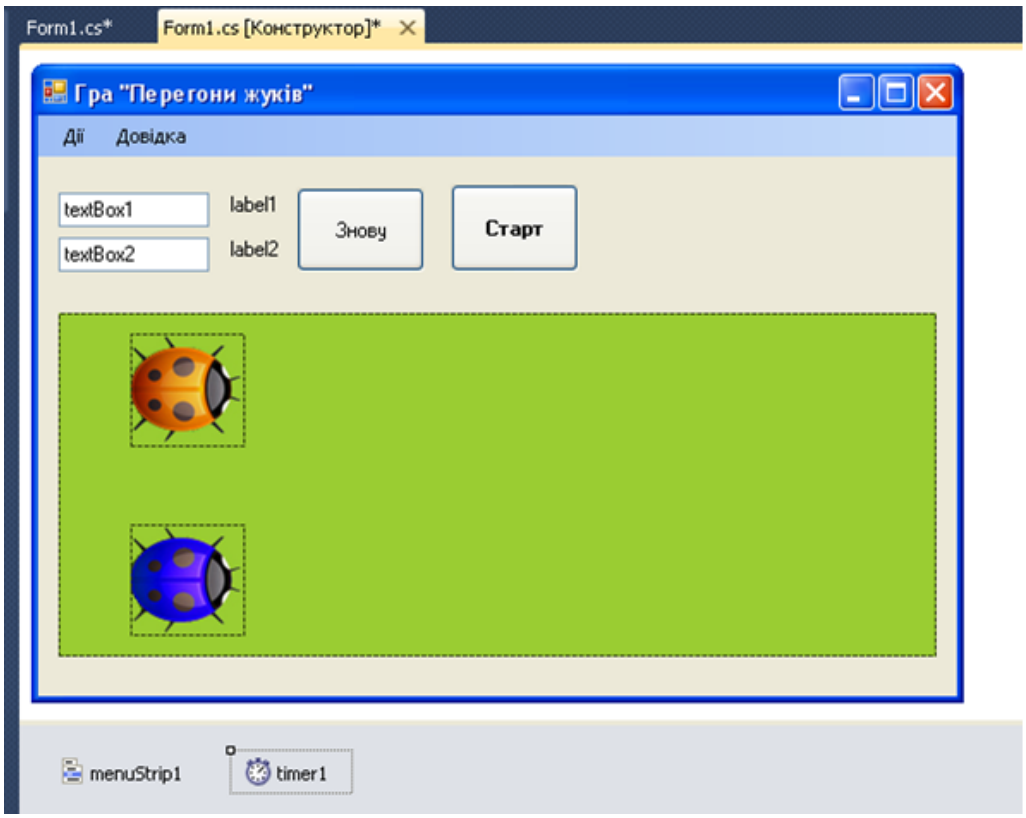


Рисунок 7.2 – Створення інтерфейсу користувача для гри «Перегони жуків»

В об'єкти pictureBox1 і pictureBox2 необхідно завантажити рисунки жуків, для цього спочатку натисніть на кнопку «…» біля властивості Image цих елементів, а потім на кнопку Import на панелі ресурсів.

Коли рисунки будуть завантажені, треба привласнити панелі panel1 у властивості BackColor такий же колір, як і тло рисунків жуків.

Для визначення координати фінішу треба провести наступні обчислення:

```
Finish = panel1.Size.Width - pictureBox1.Width;
```

`panel1.Size.Width` - ширина панелі panel1,

`pictureBox1.Width` - ширину рисунку жука.

Отримана цифра буде означати фінішну координату.

Подвійним клацанням на виділеній формі варто створити подію й прописати в ній початкові налаштування нашої програми:

```
private void Form1_Load(object sender, EventArgs e)
  {
    textBox1.Text = "Гравець 1";
    textBox2.Text = "Гравець 2";
    label1.Text = "0";
    label2.Text = "0";
    pictureBox1.Left = 1;
    pictureBox2.Left = 1;
     Finish = panel1.Size.Width - pictureBox1.Width;
  }
```

Кнопці button1 у властивості Text варто привласнити значення "СТАРТ", а button2 значення "Знову".

Запустимо додаток на виконання; форма повинна прийняти вигляд, показаний на рис. 7.3:

Рисунок 7.3 – Інтерфейс користувача гри «Перегони жуків»

Далі, створимо в коді програми декілька глобальних змінних:

```
public partial class Form1 : Form
  {
   int flag1;
   int x1, x2;
   int Finish;
   public Form1()
  {
 ...
```

Змінна flag1 - призначена для контролю ігрової ситуації, якщо flag1 дорівнює 0, то жуки стоять на місці, а якщо flag1 не дорівнює 0, то жуки починають рухатися. Змінні x1 і x2 призначені для зберігання горизонтальних координат положення жуків на біговій доріжці.

Створимо подію для кнопки "Знову", ця подія очищає всі змінні й встановлює жуків у стартову позицію.

```
private void button2_Click(object sender, EventArgs e)
  {
   label1.Text = "0";
   label2.Text = "0";
   x1 = 1; x2 = 1;
   pictureBox1.Left = x1;
   pictureBox2.Left = x2;
   flag1 = 0;
  }
```

Для кнопки "СТАРТ" теж створимо подію, вона буде дуже простою, потрібно привласнити змінній flag1 будь-яке значення не рівне 0.

```
private void button1_Click(object sender, EventArgs e)
  {
    flag1 = 1;
  }
```

Тепер залишилося створити подію, що буде переміщати жуків. Для цього знадобиться компонент таймер timer1. Встановимо компонент на формі й зробимо деякі настроювання.

Рисунок 7.4 – Компонент таймер timer1

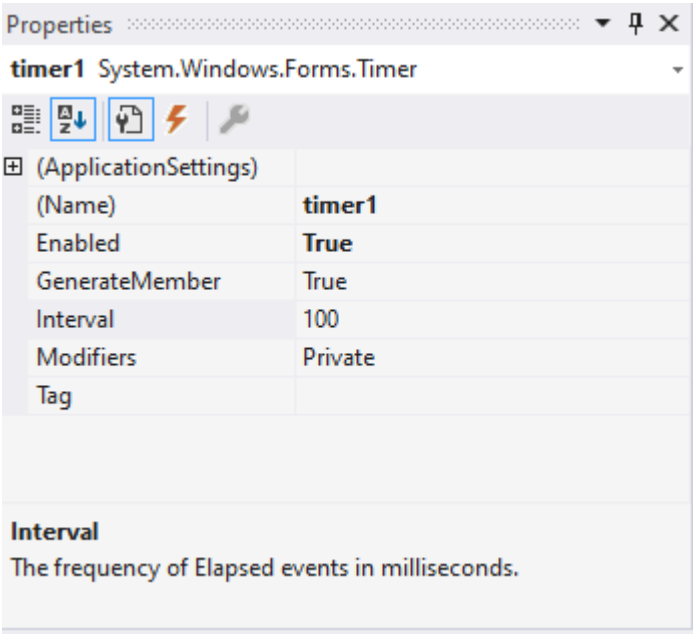


Рисунок 7.5 – Налаштування властивостей елементу керування «Таймер»

Властивість GenerateMember і Enabled повиннібути True. Властивість Interval дорівнює 100 - це один тік таймера в секунду. Якщо жуки будуть рухатися повільно, то це значення можна зменшити, наприклад до 50.

Для того щоб визначити скільком секундам відповідає значення задане у властивості Interval – розділіть це число на 1000. Напр., якщо Interval=5000, то дії вказані в обробнику таймера будуть відбуватися з інтервалом в 5 секунд.

Подвійним клацанням по іконці timer1 створимо подію руху жучків:

```
private void timer1_Tick(object sender, EventArgs e)
  {
   if (flag1 != 0) // Якщо дано старт :
   {
   Random a = new Random(); // Включаємо генератор випадкових чисел
 // У змінну count записуємо випадкове число в діапазоні від 0 до 8
   int count = a.Next(8);
 // Нарощуємо значення координати x1 на випадкове число count
   x1 += count;
 // Виводимо значення пройденого шляху для першого гравця
   label1.Text = Convert.ToString(x1);
// Зміщаємо першого жучка на випадкову величину
   pictureBox1.Left = x1;
 // Створюємо випадкове число для другого гравця й повторюємо все те ж саме
   count = a.Next(8);
   x2 += count;
   label2.Text = Convert.ToString(x2);
   pictureBox2.Left = x2;
  }
 // Перевіряємо, якийізгравцівдійшовдофінішуйзупиняємопроцес
  if ((x1 >= Finish) || (x2 >= Finish))
  {
   flag1 = 0;
  }
 }
```

Запустимо програму на виконання й, натиснувши кнопку "СТАРТ" зіграємо кілька партій, щоб переконатися, що гравці приходять до фінішу зі змінним успіхом.

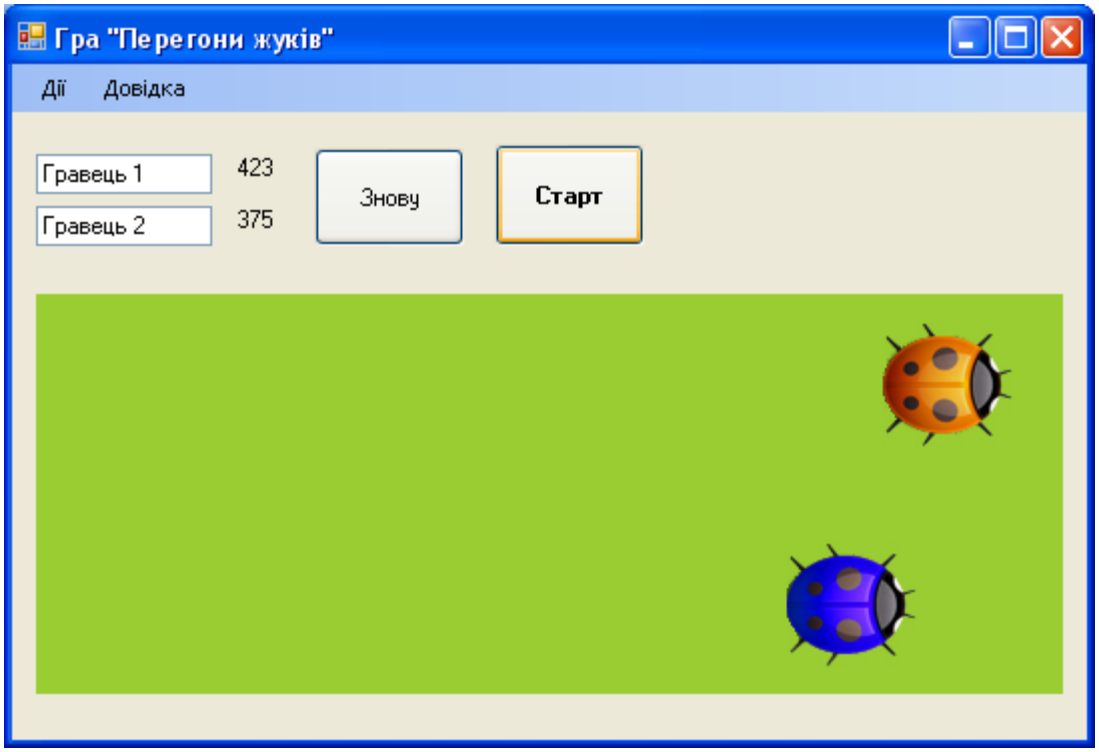


Рисунок 7.5 – Результати роботи програми «Перегони жуків»

Завдання до лабораторної роботи:

1. Реалізуйте описану в теоретичній частині гру «Перегони жуків». Додайте в програму поле, в якому буде виводитися ім'я переможця. Реалізуйте можливість управління програмою через меню користувача та створіть вікно довідки про програму.

2. Виконайте одне з наступних завдань за своїм варіантом (використовуючи елемент управління Timer):

1) Зобразити червоне коло, що випадковим чином пересувається по білому квадрату у вікні програми при натисненні кнопки «Старт» та зупиняється при натисненні кнопки «Стоп».

2) Виводити на екран питання та варіанти відповідей на нього, давати певний інтервал часу на відповідь. Якщо користувач відповів невірно, або не відповів по завершенню заданого інтервалу на поточне питання, віднімати 3 бали, якщо відповів вірно додавати 6 балів; після чого виводити наступне питання. Виводити проміжну та остаточну кількість балів на екран. Зробити 5 питань по 3 варіанти відповіді по мові програмування C#.

3) Виводити на екран N точок, що повинні рухатися по вікну випадковим чином, періодично (через інтервали часу тривалістю M) визначати скільки точок потрапило у ділянку вікна відокремлену квадратом nxn. Використати 2 різні елементи Timer.

4) Виводити у вікні програми через кожні 3с 6 різних та 2 однакові випадкові зображення, якщо користувач встигатиме послідовно натиснути на 2 однакові зображення, то нараховувати йому 2 бали, якщо лише на одне з них то 1 бал. По завершенні 1хв зупинити гру та вивести загальну кількість балів.

5) Зобразити на екрані таймер у форматі *години:хвилини:секунди*, що починає відлік при натисненні кнопки «Старт», зупиняється при натисненні кнопки «Стоп» та обнуляється при натисненні кнопки «Обнулити».

6) Вивести у вікні зображення світлофора (три кола), світлофор кожні N секунд повинен змінювати колір (червоний, жовтий, зелений). Релізувати функцію встановлення користувачем значення N та вибору відтінків кольорів сфітлофора.

7) Виводити на екран графік функції y = k*sin(x) з деякою затримкою (користувач повинен мати можливість змінювати значення затримки), щоб було видно як він послідовно промальовується. При зміні користувачем коефіцієнту k, перемальовувати графік.

8) Через випадкові інтервали часу в проміжку від 3 до 8 секунд виводити на екран різні зображення тварин. Внизу вікна повинні міститися кнопки з назвами всіх використаних в програмі зображень. Якщо користувач встигатиме натиснути на відповідну поточному зображенню кнопку – нараховувати йому 10 балів.

9) Виводити у вікні фігуру, що рухається по ньому випадковим чином, та кожні 6 секунд змінює форму та колір.

10) Користувач вводить у текстове поле речення, програма змінює кожні 3с порядок слів у реченні випадковим чином (також є кнопка «старт/стоп»).

Контрольні запитання:

1) Які елементи керування необхідно було використати в даній лабораторній роботі? Що вони дозволяють реалізувати та які властивості мають?

2) Для чого використовується елемент керування Timer? Навіщо виконувати періодично певні дії?

3) Опишіть алгоритм Вашої програми.

Лабораторна робота № 8

Тема: Обробка виключних ситуацій

Мета: Навчитися реалізовувати обробку виключних ситуацій

Теоретичні відомості

Виключна ситуація (або виключення) – це ситуація, в результаті якої генерується помилка, а робота програми переривається.

Виняткові ситуації можуть виникати у процесі роботи будь-якої програми. Наприклад, користувач може розділити в калькуляторі 5 на 0 або відкрити за допомогою WinAMP-у текстовий файл. Якщо не враховувати подібні ситуації при розробці, то у кінцевого користувача будуть виникати зайві труднощі при користуванні з вашим програмним забезпеченням. Виникнення помилок неминуче. Запобігти небажаним наслідкам помилок можна тільки за допомогою правильного проектування інтерфейсу користувача.

У більшості мов програмування є можливість обробки виняткових ситуацій. І в кожній мові ця техніка своя. В **.NET** є теж своя техніка обробки виняткових ситуацій - **SEH** (*Structured Exception Handling*). Зручність даної техніки в тому, що вона єдина для всіх **.NET-мов**.

Ще однією особливістю виняткових ситуацій в **.NET** є те, що будь-яке виключення – це об'єкт, похідний від класу **System.Exception**. У даного класу є декілька дуже корисних властивостей, які допоможуть одержати докладну інформацію про виниклу ситуацію:

Message - властивість, що повертає опис виниклої помилки.

Source - властивість, що повертає ім'я об'єкта або додатка, в якому виникла помилка.

StackTrace - властивість, що повертає послідовність викликів, які призвели до помилки.

Для того, щоб виявляти виключення в C# застосовується наступна конструкція:

```
        try
        {
            //Тут міститься код, що може викликати помилку (а може й не
викликати)
        }
        catch ()
        {
            //Тут міститься код, що буде обробляти помилку
        }
        finally
        {
            //Тут міститься код, що буде виконаний незалежно
            //від того, відбулася помилка чи ні
        }
```

Дана конструкція складається із трьох блоків: **try**, **catch** і **finally**. Останній блок (*finally*) є необов'язковим. Зазвичай його використовують для коректного завершення програми. Наприклад, якщо програма працює з базою даних, і в процесі роботи відбулася помилка, то перш ніж вийти необхідно відключитися від бази даних, щоб не зіпсувати дані. Також є безліч інших випадків, коли потрібно використовувати блок **finally**.

Приклад 8.1. Обробка виключення, яке виникає при діленні на нуль в програмі, що виконує ділення **X** на **Y**.

```
static void Main(string[] args)
        {
            int x = 15;
            int y = 5;
            Console.WriteLine(x / y);
}
```

У результаті роботи програми на екран буде виведене число 3. А тепер привласнимо **Y** значення нуль. Якщо скомпілювати дану програму й запустити (не в **Visual Studio**, а самостійно), то на екран буде виведене повідомлення про помилку:

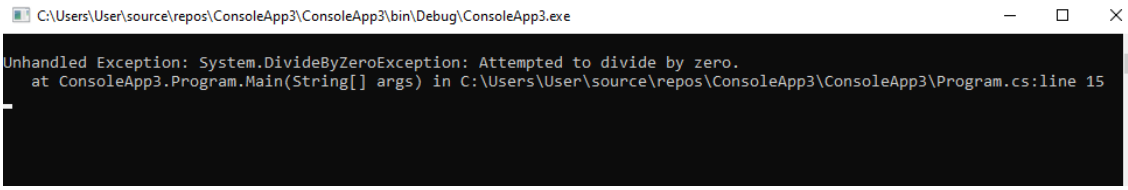


Рисунок 8.1 – Повідомлення про помилку

Щоб користувач вашої програми не побачив подібне, мало змістовне для нього, повідомлення, потрібно обробити цю виняткову ситуацію. У просторі імен **System** визначена множина готових виключень на різні помилки. Їх потрібно знати й користуватися ними. Можна також створювати й свої типи виключень, коли не одне з наявних не підходить для поставленої задачі. Ділення на 0 поширена помилка, тому для неї є своє виключення, що має назву **DivideByZeroException**. Назви інших виключень можна подивитися в довідці **Visual Studio**. Скористаємося цим виключенням й конструкцією **try/catch**, щоб обробити появу даної помилки.

```
static void Main(string[] args)
        {
            int x = 15;
            int y = 0;
```

```
            try
            {
                Console.WriteLine(x / y);
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Message: " + e.Message);
                Console.WriteLine("Source: " + e.Source);
                Console.WriteLine("StackTrace: " + e.StackTrace);
    }
            Console.ReadLine();
        }
```

Тепер "небезпечний" код розміщено у блоці **try**, після чого в блоці catch помилка буде "піймана". Оскільки відомо про те, яка помилка виникне, було явно вказано її тип (*DivideByZeroException*). Після чого були виведені дані про цю помилку на консоль. От що в нас вийшло в процесі роботи:

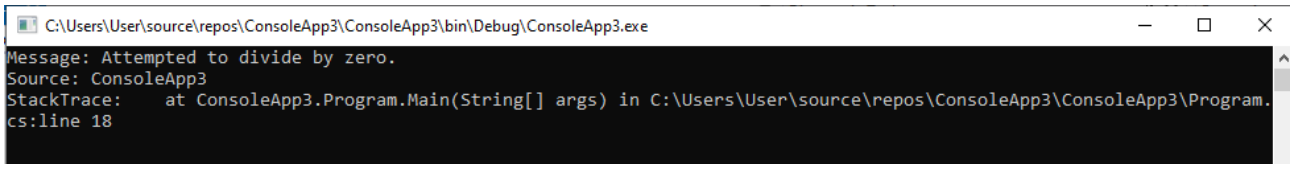


Рисунок 8.2 – Результат роботи програми з прикладу 8.1
після обробки виключення

При цьому програма не "вилітає", а продовжує працювати. Інформація, отримана із властивостей **Message**, **Source** і **StackTrace** потрібна тільки при налагодженні програми. Зрозуміло, що цю інформацію знати користувачеві зовсім не обов'язково. Для нього можна вивести щось на зразок наступної фрази: "На нуль ділити не можна!".

У наведеному прикладі оброблено всього лише одну помилку (ділення на нуль). Звичайно ж, можна обробляти й декілька помилок відразу, для цього потрібно декілька разів писати блок **catch** з різними типами помилок:

```
       try
         {
             //Тут міститься код, що може викликати помилку (а може й не
викликати)
         }
         catch ()
         {
             //Тут міститься код, що буде обробляти помилку 1
         }
         catch ()
         {
             //Тут міститься код, що буде обробляти помилку 2
         }
         finally
         {
             //Тут міститься код, що буде виконаний незалежно
```

```
        //від того, відбулася чи помилка ні
    }
```

Цілком може виникнути ситуація, коли невідомо яка помилка виникне та якого типу вона буде. Тоді можна явно не писати тип помилки. Але при цьому не можна буде довідатися значення властивостей **Message**, **Source** і **StackTrace**:

```
try
            {
                Console.WriteLine(x / y);
}
            catch
            {
                Console.WriteLine("Здається щось трапилося!");
            }
```

Розглянемо як самостійно створювати свої типи помилок.

Приклад 8.2. Для даного прикладу створимо клас **Teacher** та метод **GiveMoney()** (видати зарплату). Даний метод одержує єдиний параметр - розмір зарплати в гривнях. Але що якщо значення цього параметра буде дорівнювати негативному числу? Це і є виняткова ситуація. (Насправді для даного типу помилки передбачене виключення **IndexOutOfRangeException**, але припустимо, що ми цього не знали). Нам необхідно перевіряти значення вхідного параметра і якщо воно менше нуля викликати виключення. Цей підхід є найбільш вірним. Є й інші (невірні) рішення: ви можете взагалі не нараховувати зарплату якщо параметр негативний і не сповіщати про це користувача (недоліки такого методу, думаю, очевидні), ви можете виводити повідомлення про помилку прямо в методі **GiveMoney()** (Наприклад, за допомогою **Console.WriteLine()**), але від цього ваш клас втрачає гнучкість, тому що прив’язується до роботи з консоллю. Розглянемо написання "правильного" методу **GiveMoney()**:

```
public class Teacher : Worker
    {
        public override void GiveMoney(int money)
        {
            if (money < 0)
            {
                throw new NoMoneyException();
}
              ...
        }
    }
```

Перевіряємо значення зарплати і якщо воно менше нуля викликаємо за допомогою команди **throw** помилку типу **NoMoneyException**. Зрозуміло, що такого типу помилки не існує, ми її придумали самі, тому потрібно такий тип

помилки зробити. Робиться це за допомогою класу, що успадковується від **System.Exception**. Для прикладу достатньо перевизначити метод **Message**, щоб він виводив потрібний нам тип помилки:

```
public class NoMoneyException  : System.Exception
    {
        public  NoMoneyException(){}
        public override string  Message
{
            get
            {
                return  "Невірне значення заробітної плати";
            }
        }
    }
```

Помітьте, що у властивості **Message** є тільки **get** метод, оскільки ця властивість тільки для читання. Тепер у нас є свій тип помилки **NoMoneyException**. Настав час перевірити, як він працює:

```
static void Main(string[] args)
        {
           Teacher x = new Teacher();
           try
           {
               x.GiveMoney(-100);
           }
           catch (NoMoneyException e)
           {
               Console.WriteLine(e.Message);
           }
           Console.ReadLine();
}
```

Запустивши додаток, ви побачите на консолі очікуваний напис "Невірне значення заробітної плати". Наше виключення працює.

Завдання до лабораторної роботи:

1. Опишіть функцію множення двох цілих, обробіть помилку переповнення зверху (overflow).
2. Опишіть функцію ділення двох цілих, обробіть помилку переповнення знизу (underflow).
3. Опишіть функцію аналізу введеного користувачем номера телефону, обробіть помилку введення номера в невірному форматі (припустимий формат - (095) 555-44-33).

Контрольні питання:

1. Що таке виключна ситуація?
2. Які засоби обробки виключних ситуацій є в мові програмування C#?

3. Які оператори застосовуються для обробки виключних ситуацій?

4. За допомогою яких властивостей можна отримати інформацію про виключну ситуацію?

5. Як можна самостійно створювати типи помилок?

Лабораторна робота № 9

Тема: Створення власних просторів імен

Мета: Дослідити використання просторів імен в C# та основні методів інкапсуляції.

Теоретичні відомості

У програмах на мові C# простори імен активно використовуються двома способами. По-перше, класами платформи. NET Framework простори імен використовуються для організації великого числа класів. По-друге, оголошення власного простору імен допоможе в керуванні областю дії імен класів і методів у великих програмних проектах.

Доступ до просторів імен

Більшість програм мовою C# починаються з розділу директив using. У цьому розділі перераховуються простори імен, які будуть часто використовуватися додатком, і це рятує програміста від необхідності вказувати повне ім'я кожного разу, коли використовується метод, що міститься в них.

Наприклад, ввівши рядок:

```
using System;
```

на початку програми, програміст може використовувати код:

```
Console.WriteLine("Hello, World!");
```

замість коду:

```
System.Console.WriteLine("Hello, World!");
```

Псевдоніми просторів імен

Директива using також може використовуватися для створення псевдоніма простору імен. Наприклад, у разі використання написаного раніше простору імен, що містить вкладені простори імен, можна оголосити псевдонім для забезпечення швидкого способу звернення до одного з них, як показано в наступному прикладі:

```
using Co = Company.Proj.Nested;  // визначення псевдоніму
```

Використання просторів імен для керування областю дії

Ключове слово namespace використовується для оголошення області дії. Можливість створювати області дії в рамках проекту допомагає організовувати код і дозволяє створювати глобально унікальні типи. У наступному прикладі в двох просторах імен, одне з яких вкладено в інше, визначається клас, що називається SampleClass. Для того, щоб розрізняти, який метод викликається, використовується:

```
namespace SampleNamespace
{
class SampleClass
    {
public void SampleMethod()
```

```
        {
System.Console.WriteLine("SampleMethod всередині SampleNamespace");
        }
    }

// Створення вкладеного простору імен, і визначення іншого класу.
namespace NestedNamespace
    {
class SampleClass
        {
public void SampleMethod()
            {
System.Console.WriteLine("SampleMethod всередині NestedNamespace");
            }
        }
    }

class Program
    {
static void Main(string[] args)
        {
// На екран виводиться "SampleMethod всередині SampleNamespace."
SampleClass outer = new SampleClass();
outer.SampleMethod();

// На екран виводиться "Sample Method всередині SampleNamespace."
SampleNamespace.SampleClass outer2 = new SampleNamespace.SampleClass();
   outer2.SampleMethod();

// На екран виводиться "Sample Method всередині NestedNamespace."
NestedNamespace.SampleClass inner = new NestedNamespace.SampleClass();
inner.SampleMethod();
        }
    }
}
```

Повні імена

У наступному прикладі показані вкладені класи і простори імен. Повне ім'я зазначено в якості примітки після кожної сутності.

```
namespace N1     // N1
{
class C1      // N1.C1
    {
class C2   // N1.C1.C2
        {
        }
    }
namespace N2  // N1.N2
    {
class C2   // N1.N2.C2
        {
        }
    }
}
```

У попередньому фрагменті коду:

- простір імен N1 є членом глобального простору імен. Його повним ім'ям є N1;

- простір імен N2 є членом простору імен N1. Його повним ім'ям є N1.N2;

- клас C1 є членом простору імен N1. Його повним ім'ям є N1.C1;

- ім'я класу C2 використовується в цьому коді два рази. Однак повні імена є унікальними. Перший екземпляр класу C2 оголошений всередині класу C1; отже, його повним ім'ям є N1.C1.C2. Другий екземпляр класу C2 оголошений всередині простору імен N2; отже, його повним ім'ям є N1.N2.C2.

Використовуючи попередній фрагмента коду, можна додати новий член: клас C3, в простір імен N1.N2, як показано нижче:

```
namespace N1.N2
{
class C3   // N1.N2.C3
    {
    }
}
```

Загалом, використовуйте ключове слово :: для звернення до псевдоніму простору імен або ключове слово global :: для звернення до глобального простору імен і ключове слово . для уточнення типів або членів.

Помилкою є використання ключового слова :: з псевдонімом, що посилається на тип, а не на простір імен, наприклад:

```
using Alias = System.Console;

class TestClass
{
static void Main()
    {
// Помилка
//Alias::WriteLine("Hi");

// Вірно:
Alias.WriteLine("Hi");
    }
}
```

Зверніть увагу, що ключове слово global не є зумовленим псевдонімом. Отже, ім'я global.X не має будь-якого спеціального значення. Воно набуває спеціальне значення тільки при використанні з ключовим словом ::.

У разі визначення псевдоніму global створюється попередження компілятора CS0440, оскільки ключове слово global :: завжди посилається на глобальний простір імен, а не на псевдонім. Наприклад, наступний рядок призведе до генерування попередження:

```
using global = System.Collections;   // Warning
```

Використання ключового слова :: з псевдонімами є правильною методикою, що захищає від несподіваного введення додаткових типів. Розглянемо, наприклад, наступний фрагмент коду:

```
using Alias = System;

namespace Library
{
public class C : Alias.Exception { }
}
```

Цей код працює, але якщо в подальшому буде введений тип Alias, то конструкція Alias стане пов'язана з цим типом. Використання конструкції Alias :: Exception гарантує, що ім'я Alias буде оброблятися як псевдонім простору імен і не буде помилково прийнято за тип.

Завдання до лабораторної роботи:

Створити власний простір імен, у якому створити два класи: MyInterface та MySort. У класі MyInterface реалізувати введення даних із стандартного потоку (консоль) та виведення даних на консоль та до файлу. У класі MySort реалізувати методи сортування масиву методом бульбашки (на вхід подається довільний масив, на виході повертається відсортований масив).

Робота програми:

- користувач вводить з клавіатури масив цілих чисел;
- програма виконує сортування методом бульбашки;
- програма виводить результати сортування на консоль та записує їх до файлу.

Контрольні питання:

1) Що таке простори імен і для чого вони використовуються?
2) Як використати простори імен для керування областю дії?
3) Як використати вкладені класи та простори імен?

Лабораторна робота № 10

Тема: Знайомство з ігровим рушієм Unity

Мета: Навчитись створювати проекти у середовищі Unity

Теоретичні відомості

Завантажте безкоштовну версію Unity з офіційного сайту https://unity.com/.

Запустіть інсталятор і дочекайтеся завершення установки, запустіть Unity Hub. Вам буде запропоновано створити обліковий запис.

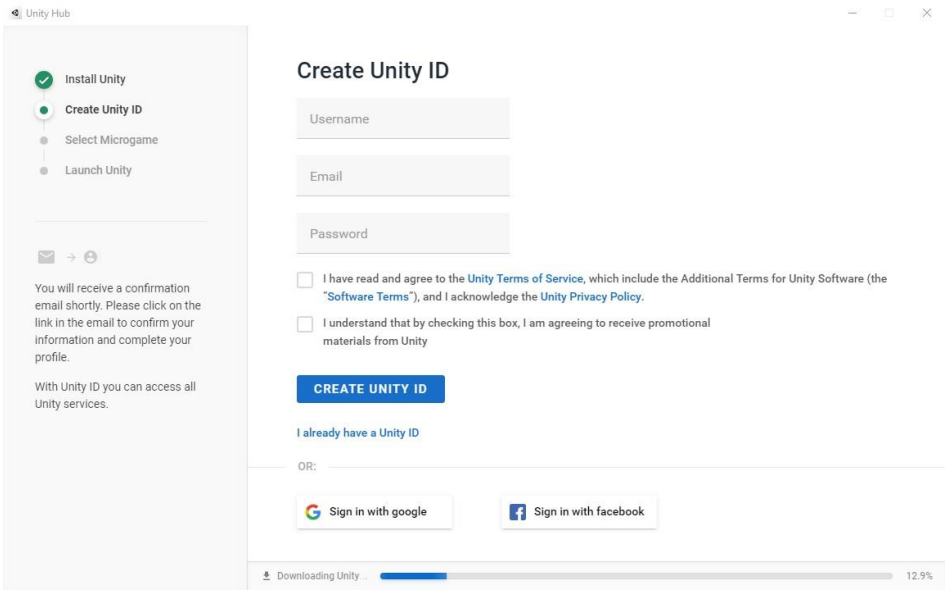


Рисунок 10.1 – Unity Hub

Створивши обліковий запис, запустіть Unity.

Натисніть File-New Project щоб створити новий проект.

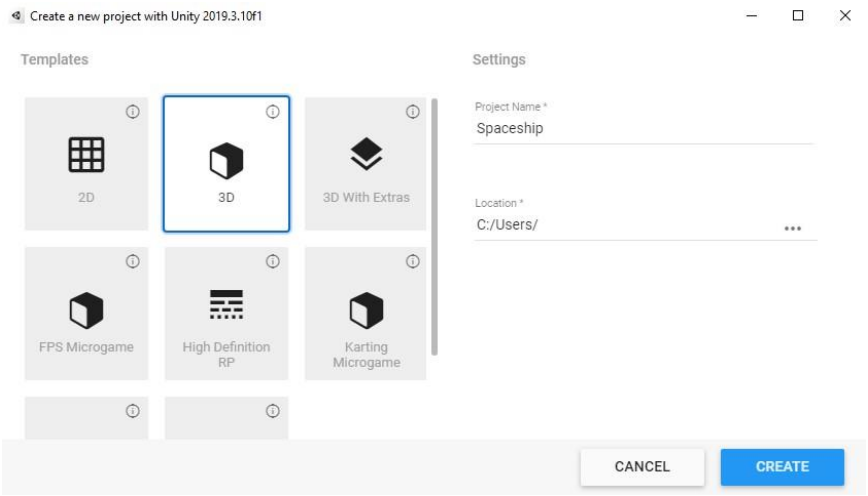


Рисунок 10.2 – Створення проекту

1. Назвіть свій проект Spaceship і виберіть місце на жорсткому диску, щоб зберегти його.

2. Ви помітите, що у вас є декілька варіантів вибору шаблонів. Кожен шаблон попередньо налаштовує Unity, щоб заощадити ваш час в залежності від того, що ви хочете зробити. Зараз виберіть 3D.

3. Натисніть Create, і Unity відкриє ваш перший проект.

Для початку необхідно розібрати простір імен Scene Management.

Давайте почнемо. Для початку створимо в Ассет дві нові сцени і назвемо їх «Scene1» і «Scene2».

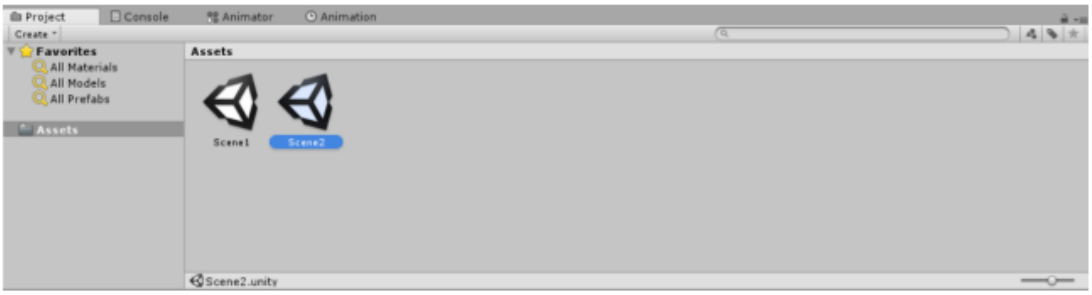


Рисунок 10.3 – Створення сцен «Scene1» і «Scene2»

Щоб створити нову сцену, клацніть правою кнопкою миші по панелі Ассет і виберіть Create> Scene.

У цих сценах нам потрібно створити елементи, за допомогою яких ми будемо переходити між сценами. Зробити це можна різними способами, але для полегшення розуміння будемо використовувати найпростіший спосіб. Ми будемо створювати об’єкт SceneChanger, який буде обробляти весь код зміни сцени.

Зазвичай, коли сцена змінюється від однієї до іншої, всі екземпляри ігрових об’єктів, сценаріїв і т.д., Що належать цій сцені, знищуються, а завантажуються екземпляри з новою. Інший метод полягає у використанні функції DontDestroyOnLoad(), яка буде підтримувати посилання на об’єкт при зміні сцени, але про це слід поговорити в іншому уроці.

Створення об’єкта і префаб SceneChanger

Отже, повернемося в Unity, на вкладці ієрархії в «Scene1» створимо порожній об’єкт і назвемо його «SceneChanger».

Рисунок 10.4 – Створення об’єкту «SceneChanger»

Отже, клацніть правою кнопкою миші на вкладці Hierarchy і виберіть UI > Panel.

Тепер додамо скрипт до об’єкту «SceneChanger» і назвемо його «SceneChanger».

Відкриємо скрипт в Visual Studio і напишемо наступний код:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneChanger : MonoBehaviour
{
 public void ChangeScene(string sceneName)
 {
       SceneManager.LoadScene(sceneName);
 }
}
```

Далі необхідно створити префаб, для цього просто виберіть об’єкт в ієрархії і перетягніть його в папку «Assets» на вкладці «Project». Після цього ви побачите синій куб з тим же ім’ям об’єкта.

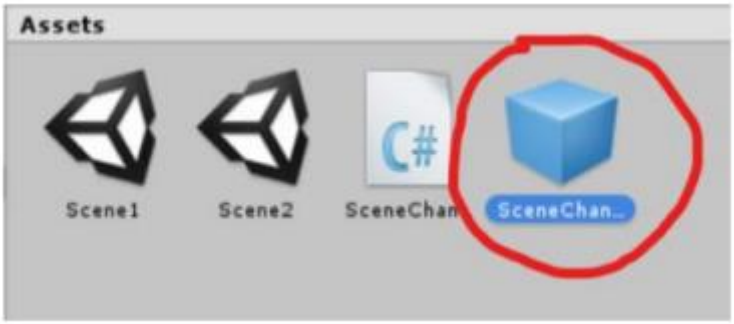


Рисунок 10.5 – Створення префабу «SceneChanger»

Тепер збережіть сцену і двічі клацніть «Scene2». У цій сцені необхідно скопіювати префаб SceneChanger. Для цього просто перенесемо його у вкладку ієрархії.

Створення призначеного для користувача інтерфейсу

В інтерфейсі у нас буде текстовий об’єкт із зазначенням назви сцени і кнопка, яка буде змінювати сцену.

Отже, клацніть правою кнопкою миші на вкладці Hierarchy і виберіть UI > Panel.

Знову клацніть правою кнопкою миші на вкладці Hierarchy і виберіть UI > Text.

Ще раз клацніть правою кнопкою миші на вкладці Hierarchy і виберіть UI > Button.

Змінимо текст на «THIS IS SCENE 1» і текст в кнопці на «CHANGE SCENE». Результат виглядає наступним чином:

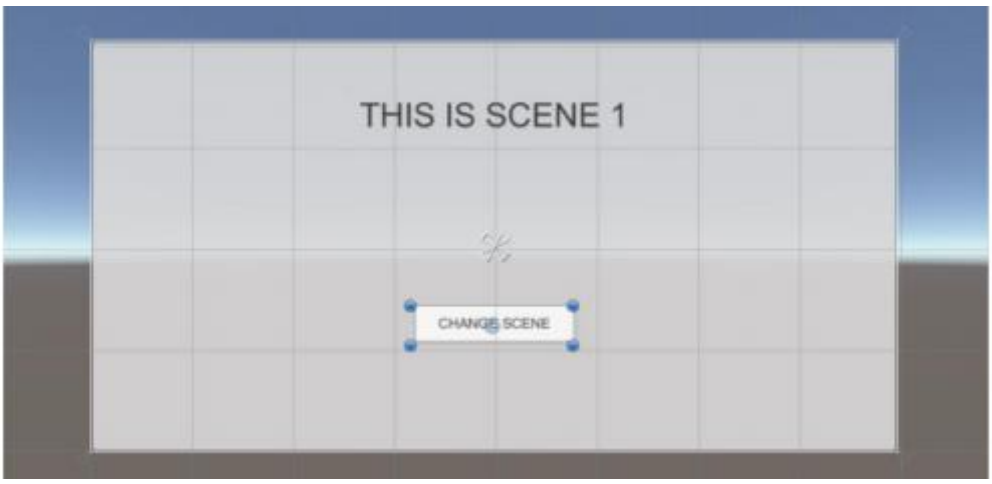


Рисунок 10.6 – Створення інтерфейсу користувача для переходу між сценами «Scene1» і «Scene2»

Далі необхідно вибрати об’єкт Button в ієрархії. На вкладці інспектора натиснемо значок «плюс» в компоненті «Button».

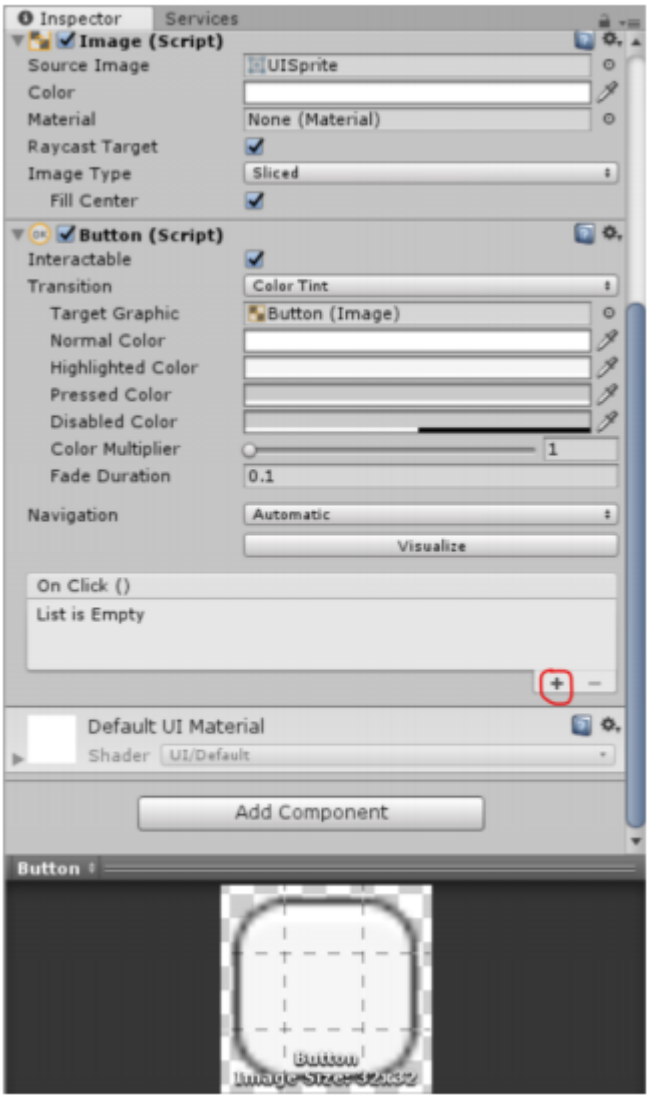


Рисунок 10.7 – Вкладка інспектора для компоненту «Button»

З’явиться наступне:

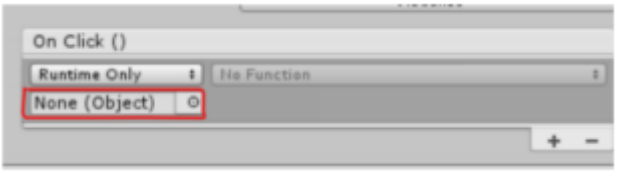


Рисунок 10.8 – Обираємо спосіб реагування на натиснення кнопки

Перетягнемо об’єкт SceneChanger з ієрархії на заслання на об’єкт (виділено червоним кольором)

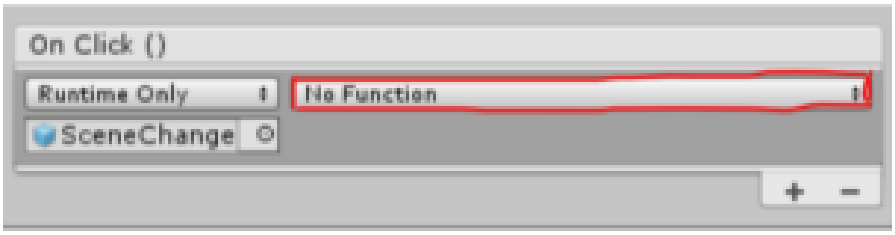


Рисунок 10.9 – Вказали спосіб реагування на натиснення кнопки

Тепер потрібно вибрати функцію (виділено червоним). У списку виберіть SceneChanger > ChangeScene (string).

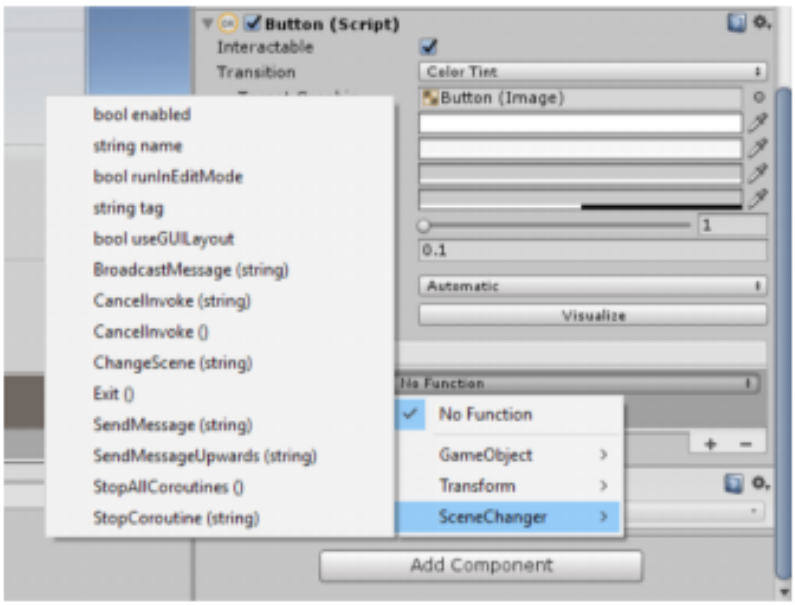


Рисунок 10.10 – Обираємо вибрати функцію
для реагування на натиснення кнопки

Тепер ви побачите порожнє місце під списком функцій. Введіть в ньому «Scene2». Тут ми вибрали об’єкт SceneChanger і отримали доступ до функції ChangeScene(). «Scene2» є входом. Тепер при кожному натисканні кнопки викликається функція ChangeScene(), і сцена змінюється на «Scene2».

Зробіть те ж саме для «Scene2», але введіть «Scene1» в якості входу і змініть текст в Scene2 відповідним чином.

Зараз потрібно додати сцени в Build Settings в File menu. Це необхідно, для того, щоб Unity розпізнала сцени. Без цього між ними не станеться ніякого переходу.

Натисніть на кнопку відтворення. Натискання на кнопку в сцені призведе до переходу до іншої сцени.

Для реалізації функції, яка відповідає за можливість приховати об'єкт та зміну кольору, розширимо наш скрипт SceneChanger.

Для роботи з кнопками та іншими елементами UI підключаємо бібліотеку UnityEngine.UI:

```
using UnityEngine.UI;
```

Оголошуемо в нашому скрипті кнопку, яку ми будемо приховувати та кнопку, яка буде відповідати за зміну кольору:

```
public GameObject hideButton;
public GameObject changeColorButton;
```

Тепер нам потрібно реалізувати самі функції:

```
public void HideButton()
{
    if (hideButton.activeInHierarchy)
          hideButton.SetActive(false);
    else
          hideButton.SetActive(true);
}
public void ChangeColor()
{
    Color randomColor = Random.ColorHSV();
    changeColorButton.GetComponent<Image>().color = randomColor;
}
```

Останнім кроком для виконання цього завдання буде підставити у відповідний SceneChanger на обох сценах відповідні дані для кнопок та додати до них відповідні функції.

Завдання:

1) Створити дві сцени з різними за кольором кнопками та текстами. Реалізувати перехід між сценами.

2) На першій сцені реалізувати кнопку, яка буде приховувати та показувати кнопку для перехода на іншу сцену.

3) На другій сцені створити додаткову кнопку, яка буде при натисканні змінювати свій колір.

Контрольні питання:

1) Що собою представляє середовище розробки Unity і які проекти у ньому можна створювати?

2) Як створити сцену у Unity?

3) Як створити кнопку у Unity?

Лабораторна робота № 11

Тема: Створення та рух об’єктів у Unity

Мета: Сворення об’єктів з фізичними властивостями у Unity

Теоретичні відомості

Об’єкти у Unity можна створювати за допомогою графічного інтерфейсу або у програмному коді. Для створення та видалення об’єктів у програмному коді ігровий рушій Unity має методи Instantiate та Destroy відповідно. Метод Instantiate для створення об’єкту першим параметром приймає Prefab об’єкта, який необхідно створити, другим параметром приймає Vector3, який вказує на позицію об’єкта на сцені та останнім параметром приймає Quaternion.identity, який вказує на те, що створюваний об’єкт не повинен мати нахилів по всім осям координат. Метод Destroy для видалення об’єкту приймає параметр типу GameObject, який є об’єктом на сцені.

Деякі ігри мають постійну кількість об’єктів на сцені, проте зазвичай персонажі, скарби та інші об’єкти створюються і видаляються під час гри. У Unity, ігровий об’єкт (GameObject) може бути створений за допомогою функції Instantiate, яка робить копію існуючого об’єкта:

```
public GameObject enemy;

void Start () {
    for (int i = 0; i <5; i ++) {
        Instantiate (enemy);
    }
}
```

Зауважте, що об’єкт, з якого береться копія не зобов’язаний бути присутнім на сцені. Набагато частіше використовується префаб, який був перетягнутий на відкриту змінну (public variable) з файлів проекту в панелі Project. Також, копіюючи ігровий об’єкт (GameObject), ви копіюєте всі компоненти оригінального об’єкта.

Також є функція Destroy, яка знищить об’єкт після того, як завантаження кадру буде завершене або опціонально після короткої паузи:

```
void OnCollisionEnter (Collision otherObj) {
    if (otherObj.gameObject.tag == "Missile") {
        Destroy (gameObject, .5f);
    }
}
```

Зауважте, що функція Destroy може знищувати окремі компоненти без впливу на сам об’єкт. Часта помилка – писати щось на зразок цього

```
 Destroy (this);
```

що насправді знищить тільки викликаний скриптовий компонент, замість того, щоб знищити ігровий об’єкт, до якого приєднаний цей скрипт.

Для створення рухомих об’єктів, що підчиняються законам фізики,

необхідно використовувати модулі Unity для роботи с колайдерами, гравітацією, переміщенням об’єктів, створенням та видаленням об’єктів.

Деякі з цих модулів підключаються через додавання до властивостей об’єкту відповідних компонентів. Після чого у об’єкта заявляється ряд додаткових властивостей, які інкапсульовані у компоненті. Розробник має можливість реалізувати ряд методів, які будуть викликані системою івентів для відповідних компонентів.

Щоб додати модуль колайдеру необхідно знайти об’єкт в ієрархії об’єктів та на вікні властивостей натиснути на кнопку «Add Component» й у випадаючему списку вибрати колайдер відповідно до вибраного об’єкту. Приклад вибору колайдеру для об’єкту кулі зображено на рис. 11.1.

Рисунок 11.1 – Додавання колайдеру до об’єкта кулі

Після того як у об’єкту з’являється властивість колайдеру цей об’єкт може зіштовхуватись з іншими об’єктами, у яких є їх власний колайдер. У ігрового рушія Unity є ряд методів які можуть бути реалізовані у методах скрипта об’єкту. У випадку з колайдерами у 3D вімірі існують методи

OnParticleCollision, OnCollisionStay, OnCollisionExit, OnCollisionEnter, у випадку коли необхідно обробити колайдери у 2D сцені використовуються методи, які закінчуються на «2D». Ці методи описують поведінку об’єкта під час часткової колізії, перебування у колізії більше, ніж один кадр, виходу з колізії та входження у колізію. До кожного з цих методів передається параметр типу Collision, у якому знаходиться об’єкт, з яким відбулася колізія. Опис поведінки об’єкта під час будь-яких вищеописаних колізій залишається за розробником. Приклад реалізації методу входження у колізію з іншим об’єктом наведенно нижче:

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "Enemy")
    {
        Destroy(collision.gameObject);
        score++;
        UIScore.GetComponent<Text>().text = score.ToString();
    }
}
```

Для того, щоб можна було додати на об’єкт гравітацію, необхідно до об’єкту додати «Rigidbody» компонент. Цей компонент має властивості, які можна налаштувати відповідно до вимог користувача. Для того, щоб на об’єкт почала впливати гравітація, необхідно виставити властивість «Use Gravity» true. Приклад цього компоненту наведений на рис. 11.2.

Рисунок 11.2 – Додавання колайдеру до об’єкта кулі
приклад компоненту «Rigidbody»

Кожен кадр сцени рушій Unity виконує метод «Update», який є методом скрипта підключеного до об’єкта сцени. У цьому методі можна реалізувати переміщення об’єкта. Для цього необхідно, щоб клас скрипта наслідувався від класу MonoBehaviour, типовий алгоритм переміщення об’єкту може виглядати наступним чином:

```
public class PlayerController : MonoBehaviour
{
    public float Speed = 2;
```

```
    private Rigidbody componentRigidbody;

    void Start()
    {
        componentRigidbody = GetComponent<Rigidbody>();
    }

    void Update()
    {
        componentRigidbody.velocity = Vector2.zero;
        if (Input.GetKey(KeyCode.A))
        {
            componentRigidbody.AddForce(Vector3.left * Speed);
        }
        if (Input.GetKey(KeyCode.D))
        {
            componentRigidbody.AddForce(Vector3.right * Speed);
        }
        if (Input.GetKey(KeyCode.W))
        {
            componentRigidbody.AddForce(Vector3.forward * Speed);
        }
        if (Input.GetKey(KeyCode.S))
        {
            componentRigidbody.AddForce(Vector3.back * Speed);
        }
    }
}
```

Клас PlayerController містить публічну зміну Speed, яка відповідає за дельту переміщення об’єкта за один кадр та приватну змінну componentRigidbody, яка зберігає стан Rigidbody об’єкту. Під час створення об’єкта на сцені виконується метод Start, у якому значення змінної встановлюється за допомогою виклику Unity GetComponent<Rigidbody>(). Цей метод використовує узагальнений тип <Rigidbody> для того, щоб повернути Rigidbody поточного об’єкту. Далі під час формування нового кадру Unity викликає метод Update. Цей метод за допомогою методу Input.GetKey перевіряє, яка клавіша буде натиснута і відповідно до натиснутої клавіши викликає метод AddForce об’єкта componentRigidbody, який здійснює переміщення об’єкту.

Завдання: Створіть два різні об’єкти, нехай у них будуть різний колір та форма. Додайте до них колайдери та Rigidbody, виставіть гравітацію об’єктів у Rigidbody рівною нулю. Напишіть скрипти для руху об’єктів по сцені. Нехай вони рухаються один одному назустріч. При зіткненні об’єктів реалізуйте знищення другого об’єкту.

Контрольні питання:

1. Що таке колайдер об’єкту? Для чого він потрібен?
2. Що таке Rigidbody об’єкту? Для чого він потрібен?
3. Як можна реалізувати переміщення об’єкту на сцені?
4. Як створити об’єкт за допомогою програмного коду?
5. Як видалити об’єкт за допомогою програмного коду?

СПИСОК ЛІТЕРАТУРИ

1. Albahari J. C# 10 in a Nutshell: The Definitive Reference 1st Edition. – O'Reilly Media, 2022. – 1058 p.
2. Baidachnyi S. Developing Windows 10 Applications with C# Kindle Edition, 2016, 521 p.
3. Buttfield-Addison P., Manning J., Nugent T. Unity Game Development Cookbook: Essentials for Every Game 1st Edition. – O'Reilly Media, 2019. – 408 p.
4. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 sp.
5. Ferrone H. Learning C# by Developing Games with Unity 2021: Kickstart your C# programming and Unity journey by building 3D games from scratch, 6th Edition 6th ed. Edition. – Packt Publishing, 2021. – 428 p.
6. Griffiths I. Programming C# 10: Build Cloud, Web, and Desktop Applications 1st Edition. – O'Reilly Media, 2022. – 833 p.
7. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 c.
8. Jain H. Data Structures & Algorithms In Go. – Hemant Jain, 2022. – 584 c.
9. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.
10. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.
11. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.
12. Rocca La M. Advanced Algorithms and Data Structures. – Manning, 2021. – 768 p.
13. Skeet J. C# in Depth: Fourth Edition 4th Edition. – Manning, 2019. – 528 p.
14. Stellman A., Greene J. Head First C#: A Learner's Guide to Real-World Programming with C# and .NET Core 4th Edition. – O'Reilly Media, 2021. – 800 p.
15. Ullman J.D., Aho A.V., Hopcroft J.E. The Design and Analysis of Computer Algorithms - International Economy Edition Paperback. – Pearson education, 1905. – 470 p.
16. Смірнов О.А., Коваленко О.В., Мелешко Є.В., Константинова Л.В., Кожанова А.С. Інженерія програмного забезпечення // Навчальний посібник. – Кіровоград: Вид. КНТУ, 2012. – 409 с.

Додаток 1
Приклад оформлення звіту з лабораторної роботи

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра програмування та захисту інформації

Лабораторна робота №_
з дисципліни «Візуальне програмування»
на тему:
_____

Виконав:
студент групи _____
_____
(П.І.Б.)

Перевірив:
викладач
_____
(П.І.Б.)

Кропивницький 20__

Тема: _____
Мета: _____

Варіант №_

Завдання: _____
_____
_____
_____
_____

Хід роботи:

<Повинен містити теоретичні дані, скриншоти розробленого програмного забезпечення та лістинг>

Відповіді на контрольні питання:
1.
2.
3.
…

Додаток 2
Приклад оформлення лістингу програми

Лістинг програми

Файл Lab_2_1.cs

```
// (c) Іваненеко І.І., ЦНТУ, 2024
// дисципліна «Візуальне програмування»
// лабораторна робота №2, завдання 2.1

using System;
using System.Text;

namespace ConsoleApplication9
{
class Program
    {
        //метод для введення цілих чисел із клавіатури
staticint ReadInt(string prompt)
        {
                Console.Write(prompt);
int x = int.Parse(Console.ReadLine());
return x;
        }
        //метод для виведення масиву на екран
staticvoid PrintArray(int[] array)
        {
for (int i = 0; i < array.Length; i++)
            {
                Console.Write("{0,5} ", array[i]);
            }
            Console.WriteLine();
        }

staticvoid Main(string[] args)
        {
int N = ReadInt("Введіть розмірність масиву: ");

int[] a = newint[N];

            //введемо елементи масиву з клавіатури
for (int i = 0; i < N; i++)
            {
                try
                {
                    a[i] = ReadInt("Введіть " + (i+1).ToString() +
                         "-й елемент масиву: ");
                }
catch(FormatException)
                {
                    Console.WriteLine("Невірний формат числа!");
                }
            }
```

…

Додаток 3
Шкала оцінювання: національна та ECTS

№	Сума балів за всі види навчальної діяльності	Оцінка ЄКТС	Оцінка за національною шкалою для екзамену
1	**90-100**	**A**	«відмінно»
2	**82-89**	**B**	«добре»
3	**74-81**	**C**	
4	**64-73**	**D**	«задовільно»
5	**60-63**	**E**	
6	**35-59**	**FX**	«незадовільно» з можливістю повторного складання
7	**1-34**	**F**	«незадовільно» з обов'язковим повторним вивченням дисципліни

Критерії оцінювання. Еквівалент оцінки в балах для кожної окремої теми може бути різний, загальну суму балів за тему визначено в навчально-методичній карті. Розподіл балів між видами занять (лекції, практичні заняття, самостійна робота) можливий шляхом спільного прийняття рішення викладача і студентів на першому занятті:

1) оцінку **«відмінно» (90-100 балів, A)** заслуговує студент, який:

- всебічно, систематично і глибоко володіє навчально-програмовим матеріалом;
- вміє самостійно виконувати завдання, передбачені програмою, використовує набуті знання і вміння у нестандартних ситуаціях;
- засвоїв основну і ознайомлений з додатковою літературою, яка рекомендована програмою;
- засвоїв взаємозв'язок основних понять дисципліни та усвідомлює їх значення для професії, яку він набуває;
- вільно висловлює власні думки, самостійно оцінює різноманітні життєві явища і факти, виявляючи особистісну позицію;
- самостійно визначає окремі цілі власної навчальної діяльності, виявив творчі здібності і використовує їх при вивченні навчальнопрограмового матеріалу, проявив нахил до наукової роботи.

2) оцінку **«добре» (82-89 балів, B)** – заслуговує студент, який:

- повністю опанував і вільно (самостійно) володіє навчально-програмовим матеріалом, в тому числі застосовує його на практиці, має системні знання достатньому обсязі відповідно до навчально-програмового матеріалу, аргументовано використовує їх у різних ситуаціях;
- має здатність до самостійного пошуку інформації, а також до аналізу, постановки і розв'язування проблем професійного спрямування;
- під час відповіді допустив деякі неточності, які самостійно виправляє, добирає переконливі аргументи на підтвердження вивченого матеріалу.

3) оцінку **«добре» (74-81 бал, C)** - заслуговує студент, який:

- в загальному роботу виконав, але відповідає на екзамені з певною кількістю помилок;
- вміє порівнювати, узагальнювати, систематизувати інформацію під керівництвом викладача, в цілому самостійно застосовувати на практиці, контролювати власну діяльність;
- опанував навчально-програмовий матеріал, успішно виконав завдання, передбачені програмою, засвоїв основну літературу, яка рекомендована програмою.

4) оцінку **«задовільно» (64-73 бали, D)** – заслуговує студент, який:

- знає основний навчально-програмовий матеріал в обсязі, необхідному для подальшого навчання і використання його у майбутній професії; - виконує завдання, але при рішенні допускає значну кількість помилок;
- ознайомлений з основною літературою, яка рекомендована програмою;
- допускає на заняттях чи екзамені помилки при виконанні завдань, але під керівництвом викладача знаходить шляхи їх усунення.

5) оцінку **«задовільно» (60-63 бали, E)** – заслуговує студент, який:

- володіє основним навчально-програмовим матеріалом в обсязі, необхідному для подальшого навчання і використання його у майбутній професії, а виконання завдань задовольняє мінімальні критерії. Знання мають репродуктивний характер.

6) оцінка **«незадовільно» (35-59 балів, FX)** – виставляється студенту, який:

- виявив суттєві прогалини в знаннях основного програмового матеріалу, допустив принципові помилки у виконанні передбачених програмою завдань.

7) оцінку **«незадовільно» (35 балів, F)** – виставляється студенту, який:

- володіє навчальним матеріалом тільки на рівні елементарного розпізнавання і відтворення окремих фактів або не володіє зовсім;
- допускає грубі помилки при виконанні завдань, передбачених програмою;
- не може продовжувати навчання і не готовий до професійної діяльності після закінчення університету без повторного вивчення даної дисципліни.