Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп’ютерних систем та мереж

Методичні вказівки

до виконання лабораторних робіт
з дисципліни «Об’єктно-орієнтоване програмування»
для студентів спеціальності
123 «Комп’ютерна інженерія»
усіх форм навчання
Частина 1

Кривий Ріг
2021

Укладачі: Музика І. О., канд. техн. наук, доцент
Кузнєцов Д. І., канд. техн. наук, доцент
Рецензент: Тиханський М. П., канд. техн. наук, доцент

Дані методичні вказівки містять завдання та теоретичні відомості для виконання лабораторних робіт з дисципліни «Об’єктно-орієнтоване програмування» за спеціальністю 123 «Комп’ютерна інженерія». Висвітлені основи програмування мовою C# із застосуванням класів, принципів абстракції, інкапсуляції, успадкування та поліморфізму.

Розглянуто
на засіданні кафедри
комп’ютерних систем та мереж

Протокол № 1
від 27.08.2021 р.

Схвалено
на вченій раді факультету
інформаційних технологій

Протокол № 1
від 30.08.2021 р.

ВСТУП

Об’єктно-орієнтоване програмування (ООП) – одна з парадигм програмування, яка розглядає програму як множину «об’єктів», що взаємодіють між собою. У ній використано декілька технологій, зокрема успадкування, модульність, поліморфізм та інкапсуляцію. Незважаючи на те, що ця парадигма з’явилась ще в 1960-х роках, вона не мала широкого застосування до 1990-х. Сьогодні багато мов програмування (зокрема, C#, C++, Java, D, Python, PHP, Ruby, Objective-C, Object Pascal та ін.) підтримують ООП.

Дисципліна «Об’єктно-орієнтоване програмування» – одна із фундаментальних дисциплін професійної підготовки студентів спеціальності 123 «Комп’ютерна інженерія». Передумовою успішного вивчення даної дисципліни є володіння навичками програмування та алгоритмізації, які отримані під час вивчення дисциплін першого року навчання: «Основи інформаційних технологій», «Вища математика» та «Програмування».

Метою даної дисципліни є оволодіння мовою програмування C#, а також методами об’єктно-орієнтованого проектування та розробки складних програм. Як результат навчання за курсом «Об’єктно-орієнтоване програмування» студенти повинні: знати основи побудови класів, способи обробки виключних ситуацій, організовувати перевантаження операцій, працювати з інтерфейсами, виконувати читання та запис текстових і бінарних файлів, вміти будувати додатки типу Console, Windows Forms, Windows Presentation Foundation на базі .NET Framework з використанням середовища Microsoft Visual Studio. Згідно навчального плану вивчення даної дисципліни передбачено протягом двох семестрів.

Запропонований матеріал буде корисним як студентам, що вивчають програмування за спеціальностями галузі знань 12 «Інформаційні технології», так і професійним програмістам, а також усім тим, хто має інтерес до даної області знань.

ЛАБОРАТОРНА РОБОТА № 1

Тема: основи програмування на мові C# та введення до платформи .NET Framework.

Мета: розробити елементарні програми на мові C# з використанням базових конструкцій мови, типів, змінних, засобів роботи з консоллю.

Теоретичні відомості

Технологія .NET – порівняно нова технологія для розробки Web-застосувань і програмного забезпечення, орієнтованого на операційну систему Windows. За допомогою .NET можна також розробляти програмне забезпечення для портативних комп'ютерів і мобільних телефонів.

Microsoft – безумовний лідер у розвитку операційних систем і технологій програмування – відзначає .NET як головну платформу розробки програмного забезпечення на найближчі роки. Цей факт зумовлює для сучасного програміста необхідність володіти архітектурою та алгоритмічними мовами .NET.

Мова C# – нова алгоритмічна мова, розроблена для написання програм у середовищі .NET. Основою C# є мови Java та C++. За детальнішого ознайомлення з C# бачимо, що вона успішно акумулювала кращі особливості Java, C++ та інших сучасних мов. Водночас C# не є надлишковою мовою. Вона містить лише необхідні конструкції.

C# поза технологією .NET не існує. Мова ґрунтується на типах даних базової бібліотеки .NET. Для виконання C#-програм необхідне загальномовне середовище виконання (CLR).

Дані вказівки містять, окрім опису конструкцій мови, вибіркову інформацію про архітектуру .NET та базову бібліотеку класів. Детальний опис цих елементів можна зробити лише в багатотомному виданні. Значний обсяг інформації та обмежений розмір вказівок зумовили деяку конспективність викладок. Базова бібліотека .NET

містить тисячі класів та десятки тисяч методів, більшість з яких мають декілька реалізацій. Цю інформацію неможливо запам'ятати. Отож при розробці проектів під .NET необхідний доступ до довідкової системи MSDN Library.

Ключові терміни платформи .NET

CLR (Common Language Runtime) – середовище виконання .NET. Можна розглядати як код, який завантажує програму, виконує та надає їй усі необхідні служби [8].

Керований код (Managed Code) – довільний код, розроблений для виконання в середовищі CLR. Код, який виконується безпосередньо під операційною системою і не потребує .NET платформи, називають некерованим.

IL (Intermediate Language, MSIL) – проміжна мова, на якій написано код, якщо він призначений для завантаження та виконання середовищем .NET. При обробці керованого коду компілятор генерує код на IL, а CLR виконує завершальну стадію компіляції в машинні коди безпосередньо перед виконанням.

CTS (Common Type System) – спільна система типів даних .NET. Розроблена для забезпечення сумісності адаптованих до .NET алгоритмічних мов. CTS надає також правила для означення нових типів даних.

CLS (Common Language Specification) – загальна специфікація алгоритмічних мов. CLS є підмножиною CTS і мінімальним набором стандартів, який повинні підтримувати усі компілятори для .NET.

Простір імен (Namespace) – логічна схема групування типів відповідної функціональності. Простори імен мають ієрархічну структуру.

Складений модуль (Assembly) – модуль, який містить компільований керований код. На відміну від виконуваних EXE чи DLL файлів, складені модулі містять також *метадані:* інформацію про модуль та всі визначені в ньому типи, методи та інше. Складений модуль може бути приватним (доступним для використання однією

аплікацією) або розподіленим (доступним для використання багатьма аплікаціями).

Глобальний кеш складених модулів (Global Assembly Cashe, GAC) – місце на диску, де зберігаються розподілені складені модулі.

Відображення (Reflection) – технологія, яка передбачає програмний доступ до метаданих складеного модуля.

Компіляція Just-In-Time (JIT) – процес виконання завершальної стадії компіляції з IL у машинний код. Передбачає компіляцію не коду загалом, а лише його частин за необхідністю.

Маніфест (Manifest) – область складеного модуля, що містить метадані.

Область застосування (AppDomain) – спосіб, за якого CLR дає змогу різним програмам виконуватися в одному і тому ж просторі процесів.

Прибирання сміття (Garbage collection) – механізм чищення пам’яті, реалізований у .NET.

Процес компіляції та запуску програми

Скомпільований код програми не містить інструкцій асемблера, а лише інструкції на IL. Ці інструкції розташовані у складеному модулі [8]. Сюди ж долучають метадані:

- опис типів даних;
- методи всередині складеного модуля;
- простий кеш (будується на основі вмісту складеного модуля та може бути використаний для перевірки його цілісності);
- інформація про версії складеного модуля;
- інформація про необхідні зовнішні складені модулі;
- інформація про привілеї, необхідні для виконання коду складеного модуля.

Пакет програм збирається зі складених модулів. Один з цих модулів є виконуваним і містить точку входу основної програми, а інші є бібліотеками. .NET завантажує виконуваний модуль, перевіряє його

цілісність та метадані, а також рівень привілеїв користувача на відповідність потребам складеного модуля.

На цьому ж етапі CLR також робить перевірку *безпеки коду за типом пам'яті* (memory type safety). Код вважають безпечним за типом пам'яті лише тоді, коли він звертається до пам'яті способами, які може контролювати середовище CLR. Якщо CLR не впевнене у безпеці коду за типом пам'яті, то (залежно від локальної політики безпеки) може відмовити у виконанні коду.

Для виконання коду CLR утворює *процес* операційної системи Windows і зазначає область застосування, в якій розташовано головний *потік* програми. CLR вибирає першу частину коду, який необхідно виконати, компілює її з мови IL на мову асемблера та виконує з відповідного потоку програми. Коли під час виконання трапляється новий метод, він компілюється у виконуваний код. Процес компіляції цього методу відбувається лише один раз. У процесі виконання коду CLR відстежує стан пам'яті та періодично запускає процедуру прибирання «сміття», тобто звільнення пам'яті від об'єктів, на які відсутні вказівники у коді.

Програмування в середовищі Microsoft Visual Studio

Коротко розглянемо елементи середовища програмування Microsoft Visual Studio 2012 (далі VS).

Типи проектів

Новий проект можна створити за допомогою меню *File | New | Project* (або натисненням кнопки *New Project*). У діалоговому вікні потрібно вибрати тип проекту та мову програмування. Для C# означені такі типи проектів (конкретний набір проектів залежить від вибору, здійсненого в процесі налаштування середовища).

Таблиця 1.1 – Типи проектів Microsoft Visual Studio

Тип проекту	Стартовий код
Windows	
Windows Application	Порожня форма.
Class Library	Бібліотека класів, яку можна використовувати в довільному коді, призначеному для платформи .NET.
Windows Control Library	Клас .NET, який можна активізувати іншим кодом .NET. Має інтерфейс користувача (подібно компонентам ActiveX).
Web Control Library	Елемент керування. Активізується сторінками ASP.NET для генерування HTML-коду представлення елемента керування при перегляді у вікні браузера.
Console Application	Аплікація, яка виконується з командної стрічки або у вікні консолі.
Windows Service	Служба, яка працює у фоновому режимі Windows.
Empty Project	Проект Windows Application без стартового коду.
Crystal Reports Windows Application	Порожня форма проекту для генератора звітів Crystal Reports.
Office	
Проекти для Microsoft Excel та Word	
Smart Device	
Проекти для Pocket PC, Smartphone та Windows CE	
Database	
Проект для SQL Server	
Web Site	
ASP.NET Web Site	Web-сайт на основі ASP.NET
ASP.NET Web Service	Клас Web-служби
Empty Web Project	Проект ASP.NET Web Site без стартового коду.
Project From Existing Code	
Project From Existing Code	Нові файли для порожнього проекту. Використовують для конвертування існуючого коду C# у проект VS.NET.

Для початку роботи необхідно завантажити Microsoft Visual Studio останньої версії та запустити його. У даній лабораторній роботі буде створюватися консольний додаток, як показано на рисунку 1.1.

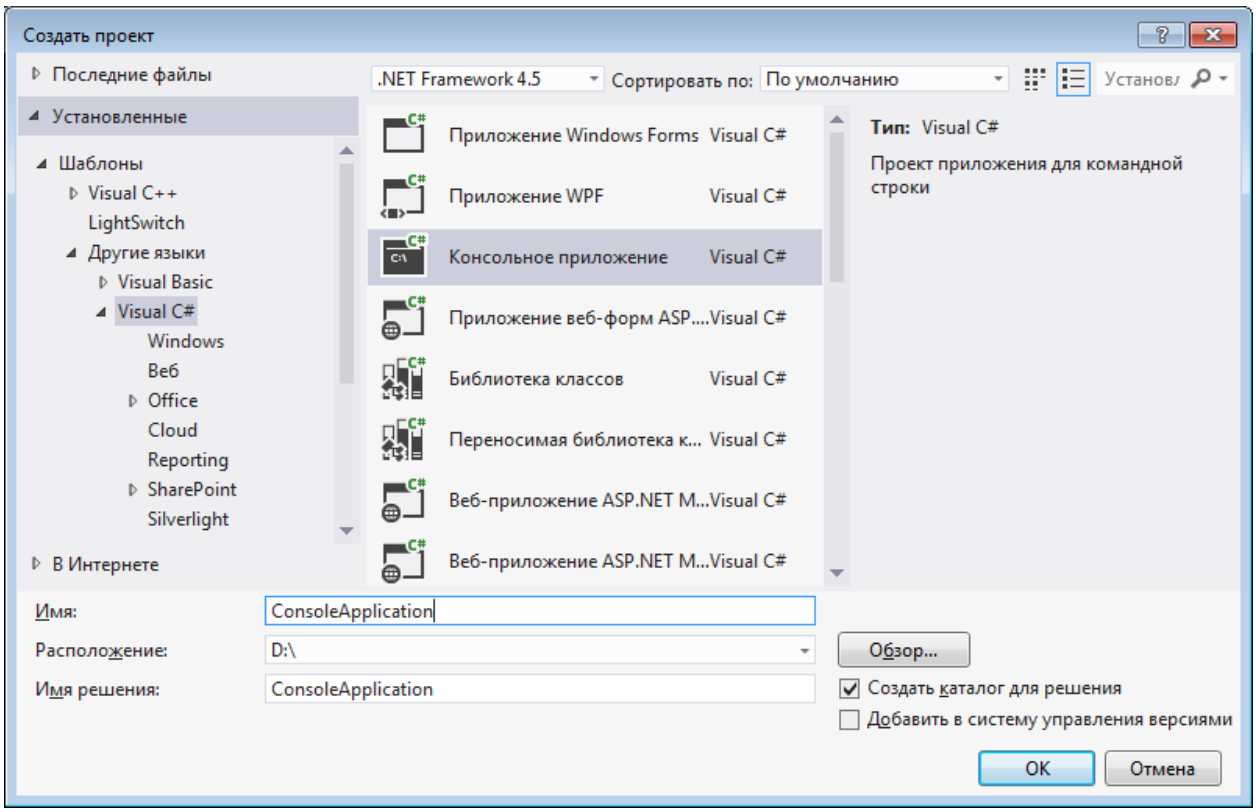

Рисунок 1.1 – Діалог створення нового проекту

Файли проекту

При створенні нового проекту VS утворює папку проекту наступної структури (рисунок 1.2).

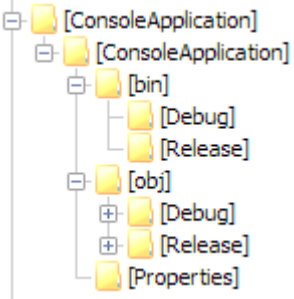

Рисунок 1.2 – Структура консольного проекту

Каталоги *bin* та *obj* призначені для розташування компільованих і тимчасових файлів.

Основний каталог *ConsoleApplication* (назва каталогу відповідатиме назві проекту) призначений винятково для VS.NET і містить інформацію щодо проекту. У процесі розробки в проект можна додавати нові каталоги.

Рішення та проекти

Рішення (*Solution*) – це набір усіх проектів, який утворює програмне забезпечення для поставленої задачі.

Проект (*Project*) – це набір усіх файлів вихідного коду та ресурсів, які компілюються в один складений модуль (assembly). У простіших проектах складений модуль – це один модуль.

Структура рішення відображається у вікні провідника рішень *(Solution Explorer*). Якщо створити консольний проект, то *Solution Explorer* відображатиме приблизно структуру, як показано на рисунку 1.3.

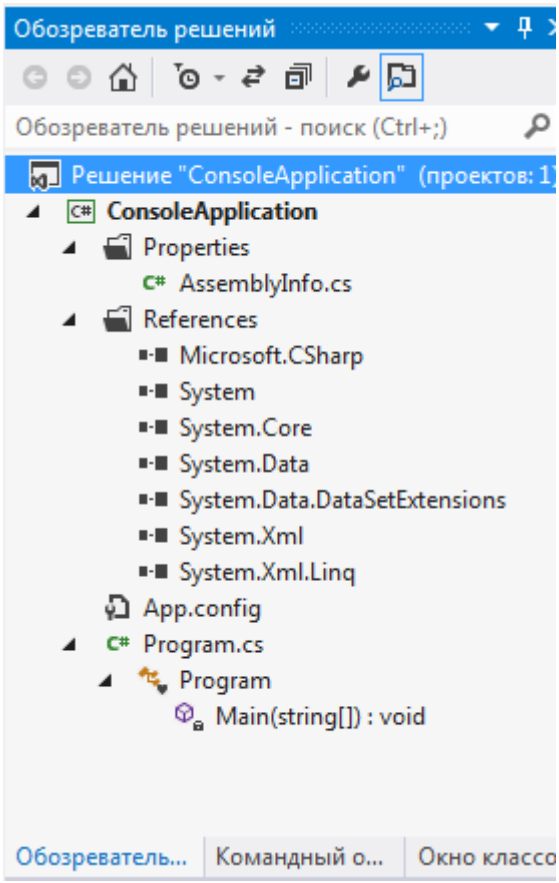

Рисунок 1.3 – Структура рішення *ConsoleApplication*

Рішення може містити проекти на різних алгоритмічних мовах, адаптованих до .NET. Натиснення правої клавіші миші на довільному вузлі активізує контекстне меню, орієнтоване на особливості обраного вузла.

Перегляд та написання коду

VS містить великий набір інструментів для розробки проектів. Ці інструменти – вікна мають декілька режимів розташування та встановлення розмірів (таблиця 1.2).

Таблиця 1.2 – Режими вікон у середовищі Visual Studio

Режим вікна	Розташування та розміри
Плаваючий *(Floating)*	Окреме вікно із заданими користувачем розмірами.
Прикріплений *(Dockable)*	Вікно розташовується усередині деякого іншого вікна-контейнера. Контейнери можна міняти, перетягуючи вікно за його заголовок. Для прикріпленого вікна контейнер утворює сторінку табуляції. Вікно займає усю вільну область контейнера.
Табульований документ *(Tabbed Document)*	Вікно розташоване усередині вікна Редактора, утворює відповідну закладку. Займає всю вільну область контейнера або ділить область Редактора навпіл (по горизонталі або по вертикалі).

Вікно можна зробити невидимим *(Hide)***,** що є аналогом закритого. Для активізації невидимого вікна його потрібно знайти у списку меню *View*. Контейнер можна зробити прихованим *(Auto hide***)**. У цьому випадку вікно ховається за одним із країв екрана і з'являється при наближенні курсора миші.

Редактор VS містить усі стандартні можливості редакторів тексту, форм, ресурсів та інших елементів проекту (рисунок 3.4). Блоки коду (класи, члени класу, цикли, набори однотипних стрічок та ін.) розглядаються як елементи дерева. їх можна згорнути або розгорнути, акцентуючи увагу лише на необхідному коді. З допомогою директив `#region` та `#endregion` можна формувати свої гілки дерева перегляду коду. Редактор коду використовує технологію *IntelliSense*. Зокрема, виводиться контекстний список можливих елементів після крапки наприкінці назви класу. Цей список можна також активізувати комбінацією клавіш Ctrl+Space. Ctrl+Shift+Space активізує список та опис параметрів методів. Редактор проводить часткову синтаксичну перевірку коду. Синтаксичні помилки підкреслюються хвилястою

лінією. При наведенні курсора миші на підкреслене слово VS виводить віконце з описом помилки.

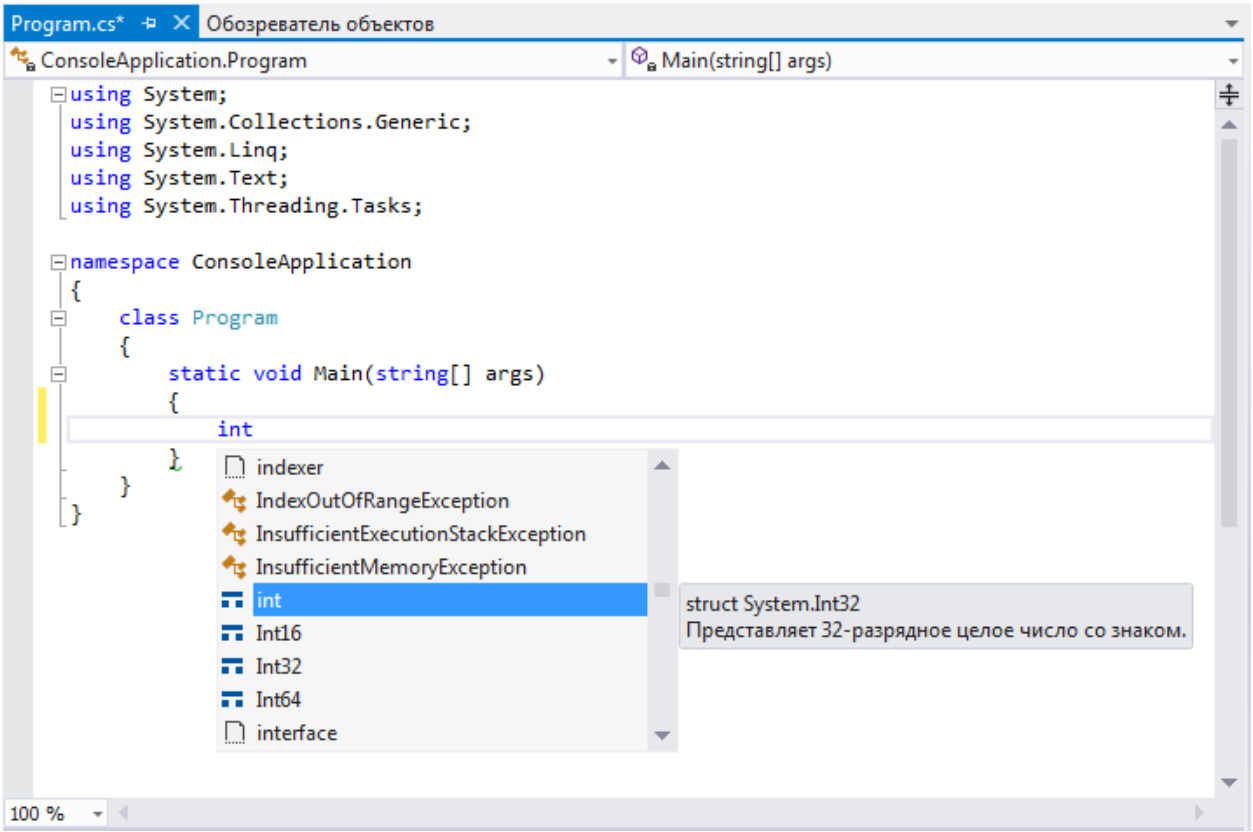

Рисунок 1.4 – Редактор коду з підказкою синтаксису IntelliSense

Вікно інструментів (*ToolBox*) містить згруповані за категоріями компоненти .NET, які використовують при розробці застосувань. Компоненти перетягують у програму з допомогою миші. Можна додавати власні категорії елементів (контекстне меню *Add Tab*). Елементи ActiveX та компоненти COM долучають опцією меню *Customize ToolBox*. Вікно властивостей (*Properties*) відображає та дає змогу редагувати значення властивостей і подій (*events*) активного (виокремленого) керуючого елемента (компонента). Властивості та події можна впорядкувати за категоріями чи алфавітом. Виокремлений елемент супроводжується коротким описом.

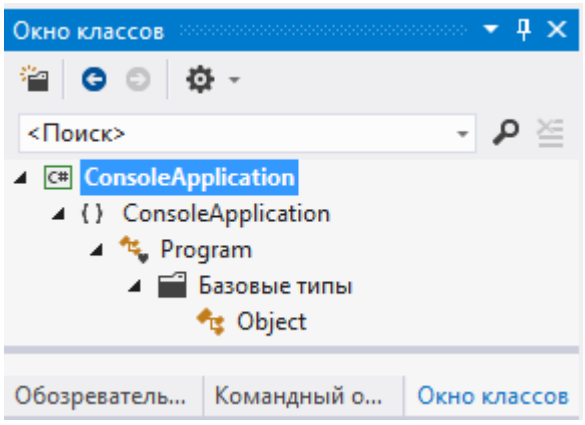

Рисунок 1.5 – Вікно класів проекту *ConsoleApplication*

Вікно класів *(Class View)* дає ієрархічний список просторів імен, класів та об’єктів проекту. Список можна сортувати та групувати за категоріями. Контекстне меню для активного елемента списку містить опцію Go To Definition (F12) – перехід до означення активного елемента в коді програми (рисунок 1.5).

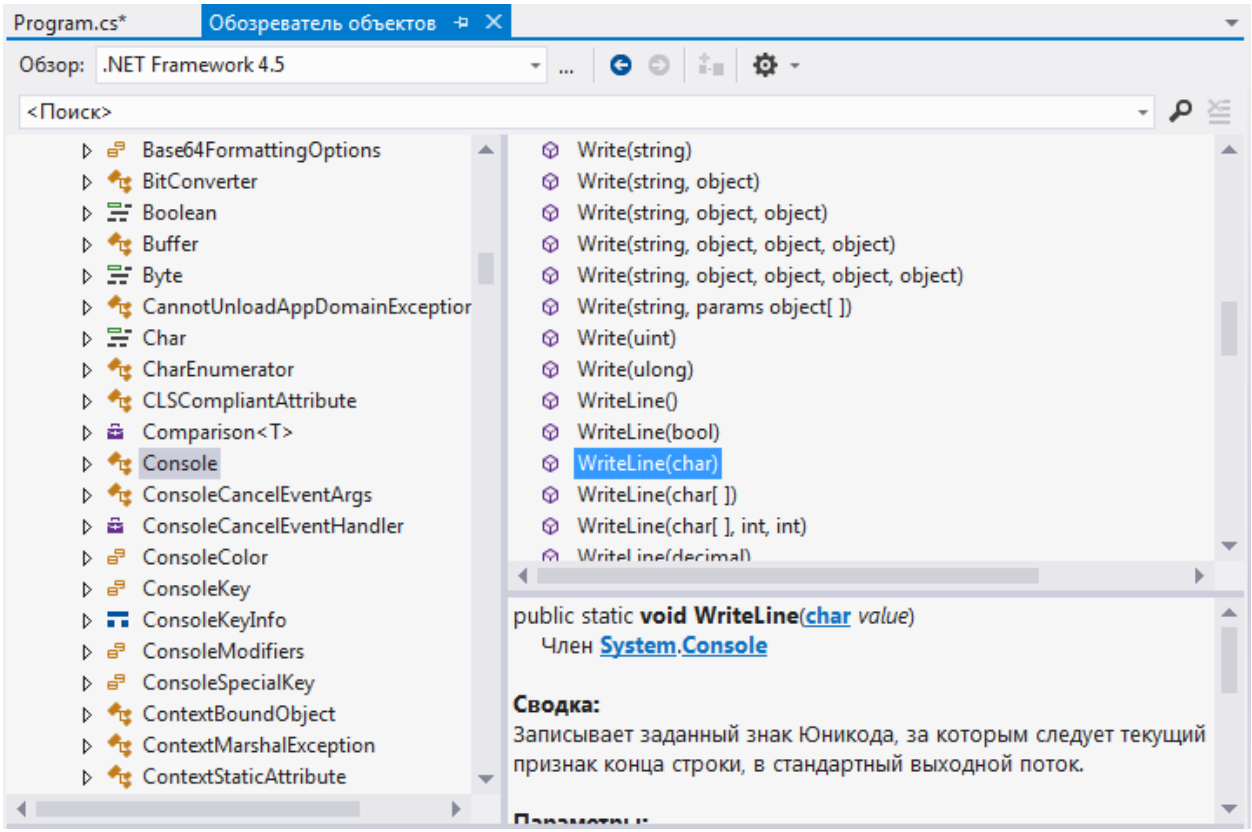

Рисунок 1.6 – Вікно браузера об’єктів

Браузер об’єктів *(Object Browser)* містить ієрархічний список класів проекту. Причому, на відміну від вікна Class View, передбачає перегляд просторів імен та класів у всіх складених модулях, які використовує проект. Вікно реалізоване подібно до файлового провідника: ліва панель показує дерево класів, а права – члени класу та опис синтаксису (рисунок 1.6).

Для перегляду COM-об’єктів доцільно використовувати програму OLEVIEW або інтерфейси .NET обгортки над COM- об’ єктом, автоматично згенерованої середовищем розробки.

Серверний провідник *(Server Explorer)* надає інформацію про комп’ютер: з’єднання з базами даних, служби, Web-служби, запущені процеси, журнали подій та інше. Server Explorer з’єднаний з вікном властивостей *Properties*: вибір елемента ініціює налаштування вікна *Properties* на показ властивостей цього елемента.

Розробка проекту

При створенні проекту VS.NET автоматично генерує дві конфігурації: *Debug* та *Release.* Головною відмінністю конфігурації *Debug* від *Release* є те, що оптимізація коду не проводиться, а у виконавчі файли додається інформація відлагодження. Очевидно, що відлагоджений продукт повинен розповсюджуватися у конфігурації *Release*. Visual Studio дає змогу також утворювати власні конфігурації. Для вибору конфігурації використовують опцію меню *Debug | Set Active Configuration*, а для редагування – *Project | Properties*. Процес розробки проекту загалом містить етапи проектування, кодування (написання програм) та відлагодження (пошук і виправлення помилок).

Одним з головних інструментів відлагодження є точки переривання (*Break Points*) процесу виконання програми. У VS для точок переривання можна задавати умови (опція меню *Debug | Break Points*), зокрема:

- задати лічильник, за умови досягнення яким заданого значення здійснити переривання;

– здійснити переривання через заданих *n* проходів;

– додати точку переривання для змінної (спрацьовує за умови зміни значення).

Важливим є також дослідження стану об’єктів у довільній точці виконання програми. З цією метою використовують такі вікна:

– *Autos* – відстежує декілька останніх змінних, доступ до яких здійснювався в процесі виконання програми;

– *Locals* – локальні змінні методу, який виконується;

– *Watch* 1, *Watch* 2, ... – явно задані змінні.

У процесі відлагодження корисним може бути вікно *Call Stack* (стек викликів), яке містить упорядкований перелік методів, які виконуються в поточний момент часу, а також значення аргументів цих методів. Вікно Exceptions (винятки) дає змогу зазначити, які дії виконати при генеруванні обраного винятку. Можна обрати і варіант продовження виконання, і варіант переходу до відлагодження. В останньому випадку виконання програми призупиняється, а відлагоджувач переходить до відповідного оператора `throw`. Меню *Debug* містить значну кількість опцій активізації інших корисних інструментів відлагодження програми.

Проста консольна програма

Розглянемо код:

```
using System; // використання простору імен System
namespace ConsoleApplication    // простір імен
{
    class Program   // головний клас програми
    {
        static void Main(string[] args) // початок програми
        {
            // виведення строки тексту в консоль
            Console.WriteLine("Hello World!");
            // затримка зчитування натиснення клавіші
            Console.ReadKey();
        }
    }
}
```

У C# кожен оператор закінчується символом «;». Для об'єднання операторів у блоки використовують фігурні дужки: {...}. Текст, розташований між наборами символів /* та */, є коментарем і компілятором ігнорується. Текст, розташований після символів // і до кінця стрічки, також є коментарем. Весь код програми повинен міститися в класі або в іншому означенні типу. Класи (та інші типи) на платформі .NET організовуються у *простори імен (namespaces).* Якщо простір імен не зазначено, то клас належить неіменованому загальному простору імен. Директива using дає вказівку компілятору, що неописані в поточному просторі імен класи потрібно шукати в поданому списку просторів імен.

Кожен виконуваний файл C# повинен мати точку входу – метод Main. Цей метод може не повертати результат (тип void), або повертати ціле число (тип int). Означення методів у C# таке:

```
[модифікатори] тип_результату НазваМетоду
( [параметри])
{
// тіло методу
}
```

Тут квадратні дужки позначають необов'язкові елементи. Модифікатори використовують для встановлення рівня доступу до методу.

Типи даних

C# підтримує загальну систему типів CTS. Кожен тип CTS є класом і володіє методами, корисними для форматування, серіальності та перетворення типів.

Дані програми зберігаються у *стеку.* Стек зберігає дані фіксованої довжини, наприклад, цілі значення. Якщо деяка функція *A* активізує функцію *B*, то всі змінні, які є локальними щодо функції *A*, зберігаються у стеку. Після виконання коду функції *B* керування повертається функції *A*. У цьому випадку зі стека зчитуються збережені значення локальних змінних.

Дані змінної довжини зберігаються в динамічно розподіленій пам'яті *(heap allocation*). Динамічну пам'ять використовують також для збереження даних, тривалість життя яких повинна бути довшою за тривалість життя методу, у якому їх означено.

C# ділить свої типи даних на дві категорії залежно від місця зберігання. *Змінні типів за значенням* зберігають свої дані в стеку, а *змінні типів за посиланням* – в *д*инамічній пам'яті. Операція присвоєння одній змінній за значенням іншої змінної за значенням утворює дві різні копії одних і тих же даних у стеку. Операція присвоєння одній змінній за посиланням іншої змінної за посиланням спричинює виникнення двох посилань на одну й ту ж ділянку пам'яті, тобто реально дані не дублюються. У C# базові типи даних, такі, як `bool` і `long`, є типами за значенням. Здебільшого складні типи C#, у тім числі й класи, означені користувачем, є типами за посиланням. Зауважимо, що на відміну від C++, де тип `struct` є специфічним класом, у C# тип `struct` є типом за значенням.

Типи C#

У C# ціле число можна оголосити з використанням типу CTS:

```
System.Int32 x;
```

Однак це виглядає неприродно для програміста C. Тому у C# означені псевдоніми *(aliases)* типів CTS. C# має 15 базових типів: 13 типів за значенням та 2 типи (*string* та *object*) за посиланням.

Типи за значенням. C# вимагає, щоб кожна змінна за значенням була явно ініціалізована початковим значенням до того, як її використовуватимуть в операціях.

Змінним цілого типу можна присвоювати значення у десятковій та шістнадцятковій системах числення. Остання вимагає префікс `0x`:

```
long x = 0x1ab;
```

Можна використовувати явно типізовані значення:

```
uint ui = 12U;
long l = 12L;
ulong ul = 12UL;
```

```
double f = 1.2F;
decimal d = 1.20M;
```

Таблиця 1.3 – Перелік типів за значенням

Тип C#	Тип .NET Framework	Діапазон	Опис та розмір
Цілі типи			
`sbyte`	`System.SByte`	-128 – 127	8-розрядне знакове ціле число
`byte`	`System.Byte`	0 – 255	8-розрядне ціле число без знака
`short`	`System.Int16`	-32768 – 32767	16-розрядне знакове ціле число
`ushort`	`System.UInt16`	0 – 65535	16-розрядне ціле число без знака
`int`	`System.Int32`	-2 147 483 648 – 2 147 483 647	32-розрядне знакове ціле число
`uint`	`System.UInt32`	0 – 4 294 967 295	32-розрядне ціле число без знака
`long`	`System.Int64`	-9 223 372 036 854 775 808 – 9 223 372 036 854 775 807	64-розрядне ціле число зі знаком
`ulong`	`System.UInt64`	0 – 18 446 744 073 709 551 615	64-розрядне ціле число без знака
Числа з плаваючою крапкою			
`float`	`System.Single`	$\pm1{,}5*10^{-45}$ – $\pm3{,}4*10^{38}$	32-розрядне дійсне число
`double`	`System.Double`	$\pm5{,}0*10^{-324}$ – $\pm1{,}7*10^{308}$	64-розрядне дійсне число
`decimal`	`System.Decimal`	$(-7{,}9*10^{28}$ – $7{,}9*10^{28})$ / / (10^{0-28})	128-розрядне фінансове число
Логічний тип			
`bool`	`System.Boolean`	`false, true`	8-бітне булеве значення
Символьний тип			
`char`	`System.Char`	U+0000 – U+FFFF	16-розрядний символ Unicode

Десятковий тип `decimal` призначений для забезпечення високої точності фінансових операцій. Змінні логічного типу можуть набувати лише значення `false` або `true`. Значення змінних символьного типу – це символи Unicode. У коді символи розташовують між одинарними лапками. Наприклад: `‘a’, ‘c’, ‘\u0041’, ‘\x0041’`. Останні два значення – це символи, задані своїми номерами. Передбачено також зведення типів: `(char) 65`.

Існують також спеціальні символи:

`\'` – одинарна лапка	`\f` – подання сторінки
`\"` – подвійна лапка	`\n` – нова стрічка
`\\` – зворотний слеш	`\r` – повернення каретки
`\o` – null-значення	`\t` – символ табуляції
`\e` – увага	`\v` – вертикальна табуляція
`\b` – повернення назад на 1 символ	

Типи за посиланням. Нехай існує деякий клас `myClass`. Розглянемо стрічку коду

```
myClass objMyClass;
```

Ця стрічка формує посилання (тобто резервує `sizeof (int)` байт для розташування адреси) на ще не утворений об'єкт `objMyClass`. Посилання `objMyClass` матиме значення `null`. У C# утворення екземпляра об'єкта за посиланням вимагає ключового слова `new`:

```
objMyClass = new myClass();
```

Цей код утворює об'єкт у динамічній пам'яті та його адресу записує в `objMyClass`. C# містить два базових типи за посиланням:

- object (тип CTS `System.Object`) – кореневий тип, який успадковують усі інші типи CTS (у тім числі типи за значенням);
- string (тип CTS `System.String`) – стрічка символів `Unicode`.

У C# тип `object` є стартовим типом-предком, від якого беруть початок усі внутрішні та всі визначені користувачем типи. Цей тип реалізує низку базових універсальних методів: `Equals()`, `GetHashCode()`, `GetType()`, `ToString()` та інші. Класам користувача, можливо, доведеться замінити реалізації деяких із цих методів, використовуючи об'єктно-орієнтований принцип перекриття (*overriding*).

При оголошенні змінної за посиланням типу `object` або похідного від `object` класу цій змінній початково надається значення `null` (за умови, що ця змінна є частиною класу). Якщо ж неініціалізована змінна оголошена в межах методу, у програмі виникне помилка на етапі компіляції.

Об’єкт `string` дає змогу зберігати (в динамічній пам’яті) набір символів Unicode. Зауважимо, що коли одну стрічкову змінну присвоїти іншій, то в результаті одержимо два посилання на одну й ту ж стрічку в пам’яті. Якщо ж подальший код вносить зміни в одну із цих стрічок, то утворюється новий об’єкт `string`, водночас інша стрічка залишається без змін. Стрічкові літерали розташовуються у подвійних лапках `"..."`. Стрічки C# можуть містити ті ж символи, що й тип `char`. Символ «\», якщо він не призначений для означення спеціального символу, подвоюється:

```
string filepath = "C:\\Program Files\\FileName.cs";
```

Стрічковому літералу може передувати символ @, який дає вказівку всі символи трактувати за принципом «як є»:

```
string filepath = @"The file
    C:\Program Files\F.cs is C#-file";
```

Значення цієї стрічки містить також і всі пробіли перед C:\ .

Структури

Структура – це подібний до класу тип даних за значенням. Оскільки структура розташовується в стеку, вона утворюється більш ефективно, ніж клас. До того ж копіюється простим присвоєнням. Структура ініціалізується відразу після свого оголошення, а всі поля встановлюються у значення за замовчуванням (0, `false`, `null`). Однак компілятор не дає змоги копіювати одну структуру в іншу до її ініціалізації за допомогою ключового слова `new`.

Приклад:

```
public struct Student
{
    public string FirstName;
    public string LastName;
    public string Group;
}
Student st1, st2;
st1 = new Student;
```

```
st1.FirstName = "Володимир";
st1.LastName = "Воронін";
st1.Group = "КСМ-14";
st2 = st1;
```

У C# структура може виконувати більшість функцій класу (однак не підтримує наслідування реалізацій). Оскільки екземпляри структур розташовані в стеку, доцільно їх використовувати для представлення невеликих об’єктів. Це одна з причин, за якої їх застосовують для реалізації всіх інших типів даних за значенням у платформі .NET.

Переліки

Перелік – це означуваний користувачем цілий тип даних. Переліки мають тип за значенням. При оголошенні переліку вказується набір допустимих значень, які можуть набувати екземпляри переліку:

```
public enum Light
{
    Green = 0,
    Yellow = 1,
    Red = 2
}
Light l;
l = Light.Red;
```

Масиви

Масив – це колекція змінних однакового типу, звернення до яких відбувається з використанням загального для всіх імені.

Для оголошення одновимірного масиву використовують такий синтаксис:

```
тип [] ім'я_масиву = new тип [розмір];
```

Наприклад,

```
int[] arrInt;
int[] arrInt = new int[10];
```

Масиви у C# є 0-базованими, тобто перший елемент масиву має індекс 0:

```
arrInt[0] = 100;
```

Для встановлення розміру масиву використовують число, константу або змінну. Установити розмір можна також наданням значень елементам масиву при утворенні масиву:

```
string[] str = new string{"1-й", "2-й"};
```

Масиви є класом у C# і володіють певними властивостями та методами. Наприклад,

```
int L = arrInt.Length; //повертає розмірність масиву
int L = arrInt.GetLength(0); //повертає розмірність
// заданого виміру для багатовимірних масивів
```

До масивів застосовують статичні методи класу Array. Наприклад, елементи масиву можна сортувати:

```
Array.Sort (arrInt) ; //сортувати у порядку зростання
Array.Reverse(arrInt) ; //змінити напрям сортування
```

Багатовимірні масиви

У C# підтримуються багатовимірні масиви двох типів: прямокутні та ортогональні.

Роботу з прямокутними масивами демонструють такі приклади:

```
int[,] arr2lnt = new int[5,10];
string[,] studList = {{"Андрій","Банах"},
                      {"Ірина","Бужська"},
                       {"Семен","Вовк"}
};
int[,,] arr3lnt = new int[5,10,5]; arr3lnt[0,0,0] = 100;
```

В ортогональних масивах кожен вимір може мати свою довжину:

```
int[] [] a = new int[3][];
a[0] = new int [5];
a[1] = new int [3];
a[2] = new int [10];
```

Типи внутрішніх масивів не обов’ язково збігаються з оголошеним:

```
int [][,] b = new int[3] [,];
b[0] = new int[5,2];
```

На відміну від прямокутних масивів кожен індекс в ортогональних масивах виокремлено набором квадратних дужок:

```
a[0][0] = 100; b[0][0,0] = 100;
```

Перетворення типів

C# дає змогу присвоїти значення змінних одного типу змінним деяких інших типів. Це присвоєння може бути явним або неявним.

Виконати неявні перетворення для цілих типів можна лише тоді, коли перетворюються цілі у більш крупніші цілі або цілі без знака в цілі зі знаком такого ж розміру. В інших випадках компілятор генерує помилку. Наприклад,

```
byte b1 = 1;
byte b2 = 2;
byte b = b1 + b2; //помилка компіляції
int i = b1 + b2;  //коректне неявне перетворення
```

Довільне ціле можна перетворювати в типи `float, double та decimal`. Однак можлива втрата точності для перетворень із `int, uint, long або ulong до float та з long або ulong до double.`

Дозволеними є також неявні перетворення з типу `float` у тип `double` та з типу `char` до `ushort, int, uint, long, ulong, float,double` і `decimal.`

Явне перетворення типів використовують з метою уникнення помилки компіляції при неявному перетворенні. Типовий синтаксис:

```
long l = 1000;
int i = (int)l;
```

Якщо значення, яке явно перетворюється, лежить за межами діапазону значень типу призначення, то помилка не виникає, а результат перетворення залежатиме від конкретного типу. Для

перевірки коректності перетворення використовують оператор checked:

```
int i = checked((int)l);
```

Якщо значення l виходить за межі значень типу int, то оператор checked згенерує виняткову ситуацію (переповнення). Типи за значенням допускають перетворення лише у числові типи, типи enum і тип char. Не можна безпосередньо перетворити тип bool до довільного іншого, і навпаки.

Тип стрічки перетворюється в інші типи (і навпаки) за допомогою відповідних методів .NET. Оскільки кожен клас C# є нащадком класу object, то він успадковує (або має власну реалізацію) метод ToString():

```
int i = 1; string s = i.ToString();
```

Для переводу стрічкового значення у числове або значення типу bool використовують метод Parse:

```
string s = "1"; int i = Int32.Parse(s);
```

Широкий набір методів перетворення типів надає також клас System.Convert.

Упакування та розпакування

Упакування (boxing) та розпакування даєзмогу перетворювати типи за значенням у типи за посиланням, і навпаки.

Приклад упакування: `int i = 10; object obj = i;`

Приклад розпакування: `int j = (int)obj;`

Розпаковувати можна лише змінну, яка попередньо була упакована. Змінна, у яку розпаковують об’єкт, повинна мати достатній розмір для розташування усіх байтів змінної, яку розпаковують.

Змінні

Змінні оголошуються у C# із використанням такого синтаксису:

```
[модифікатори] тип_даних ідентифікатор;
```

Якщо в одному виразі оголошуються та ініціалізуються кілька змінних, то всі вони матимуть однаковий тип і модифікатори. Змінні з однаковими іменами не можуть оголошуватися двічі в одній області видимості. Ідентифікатори чутливі до регістру символів. Ідентифікатори повинні починатися літерою або символом «_». Ідентифікаторами не можуть бути ключові слова C#. Якщо необхідно використати ключове слово як ідентифікатор, то перед ним ставлять символ @. Цей символ дає вказівку компілятору сприймати такі символи як ідентифікатор, а не ключове слово. Ідентифікатори можна записувати і з допомогою символів Unicode \uxxxx. Наприклад, `\u050fidentifier`.

Змінні з однаковими ідентифікаторами можна використовувати за умови їхнього розташування у різних областях видимості:

```
for (int i =0; i < 10; i++) {...};
for (int i = 0; i < 50; i++) {...};
```

Дві змінні з однаковимиідентифікаторамиможна використовувати у спільній області видимості лише у випадку, коли одна змінна – поле класу, а друга – локальна змінна, оголошена в методі класу. Наприклад:

```
public class A
{
    private int j = 10;
    public int Method()
    {
        int j = 20;
        return j + this.j;
    }
}
```

Ключове слово `this` вказує, що елемент (змінна `j`) належить поточному екземпляру класу.

Константи

Константи – це змінні, описані з використанням модифікатора const. Наприклад, `const int a = 10;` Константи є подібними до статичних полів лише для читання із такими розбіжностями:

- локальні змінні та поля можуть бути оголошені константами;
- константи ініціалізуються в оголошенні;
- значення константи повинно бути обчислюваним під час компіляції, отож константу не можна ініціалізувати виразом із використанням змінної;
- константа завжди є статичною.

Таблиця 1.4 – Перелік операцій C#

Категорія	Операції
Арифметичні	`+ - * / %`
Логічні	`& | ^ ~ && || !`
Конкатенація стрічок	`+`
Інкремент та декремент	`++ --`
Побітовий зсув	`<< >>`
Порівняння	`== != < > <= >=`
Присвоєння	`= += -= *= /= %= |= ^= <<= =>>`
Доступ до членів (для об'єктів)	`.`
Індексування масивів та індексаторів	`[]`
Перетворення типу	`()`
Умовна (тернарна операція)	`?:`
Створення об'єкта	`new`
Інформація про тип	`sizeof is typeof`
Керування винятками переповнення	`checked unchecked`
Розіменування та адресація	`* -> & []`

Зазначимо, що чотири із цих операцій (`sizeof, *, ->, &`) доступні лише в «небезпечному коді» *(unsafe* – код, для якого C# не виконує перевірку безпечності за типом).

Скорочений запис операцій

Операції інкремента та декремента (`++` та `--`) можуть стояти як до, так і після змінної. Вирази `x++` та `++x` еквівалентні виразу `x = x + 1`.

Однак префіксний оператор (++x) збільшує значення x до його використання у виразі. Навпаки, постфіксний оператор (x++) збільшує значення x після обчислення виразу. Наприклад:

```
bool b1, b2;
int x = 0;
b1 = ++x == 1; // b1=true
b2 = x++ == 2; // b2=false
```

Префіксний і постфіксний оператори декремента поводять себе аналогічно, однак зменшують операнд. Інші скорочені оператори (+= -= *= /= %= |= ^= <<= =>>)вимагають двох операндів. їх використовують для зміни значення першого операнда шляхом виконання над ним арифметичної, логічної чи бітової операції. Наприклад, оператори `x += y;` та `x = x + y;` є еквівалентними.

Тернарний оператор

Тернарний оператор `?:` є скороченою формою конструкції `if...else`. Його назва вказує на використання трьох операндів. Оператор обчислює умову та повертає одне значення у випадку, якщо умова істинна, та інше – в протилежному випадку. Синтаксис оператора:

```
умова ? значення_істина : значення_хибність
```

Наприклад:

```
string s = (x >= 0 ? "Додатне" : "Від'ємне");
```

Оператори `checked` та `unchecked`

Використання оператора `checked` ми вже демонстрували, розглядаючи перетворення типів. Зауважимо, що перевірку на переповнення можна увімкнути відразу для всього коду, виконавши компіляцію програми з опцією `/checked`. Власне щодо цього випадку може виникнути потреба відміни перевірки на переповнення деякого коду, для чого й використовують оператор `unchecked`.

Оператор `sizeof`

Розмір (у байтах), необхідний для збереження у стеку змінної за значенням, можна визначити з допомогою оператора `sizeof`:

```
int L;
unsafe
{
    L = sizeof(double);
}
```

У результаті змінна L набуде значення 8. Зазначимо, що оператор `sizeof` можна використовувати лише в блоках коду, який не є безпечним. За замовчуванням компілятор C# не сприймає такий код. Отож його підтримку необхідно активізувати або використанням опції командної стрічки компілятора `/unsafe`, або встановленням у значення true пункту *Allow unsafe code blocks* на сторінці властивостей проекту.

Пріоритет операцій

Таблиця 1.5 – Пріоритет операцій C#

Група	Операції
Унарні	`() . [] x++ x-- new typeof sizeof checked unchecked + - ! ++x --x` та операції приведення типів
Множення / ділення	`* / %`
Додавання / віднімання	`+ -`
Операції побітового зсуву	`>> <<`
Відношення	`< > <= >= is`
Порівняння	`== !=`
Побітовий AND	`&`
Побітовий XOR	`|`
Побітовий OR	`^`
Логічний AND	`&&`
Логічний OR	`||`
Тернарний оператор	`?:`
Присвоєння	`= += -= *= /= %= |= ^= <<= =>>`

Оператори керування

Умовні оператори

C# має два умовні оператори: `if` та `switch`.

Синтаксис оператора `if` такий:

```
if (умова)
    оператор(и)
[else
    оператор(и)]
```

Квадратні дужки позначають необов'язкову частину оператора `if`. *Умова* повинна повертати результат логічного типу: `true` або `false`. Якщо потрібно виконати декілька операторів, то їх об'єднують у блок (розташовують у фігурних дужках `{...}`).

Оператор `switch...case` призначений для вибору одного з варіантів подальшого виконання програми з декількох. Синтаксис оператора такий:

```
switch (вираз)
{
    case константа:
        оператор(и)
        оператор переходу
    [default:
        оператор(и)
        оператор переходу]
}
```

Якщо *вираз* дорівнюватиме значенню однієї з позначок `case`, то виконуватиметься наступний за цією позначкою код. Зауважимо, що цей код не обов'язково оформляти як блок. Однак він повинен завершуватися оператором переходу (зазвичай, це оператор `break`). Єдиний виняток – кілька позначок `case` підряд:

```
switch (s)
{
    case "A":
    case "D":
        i = 1;
```

```
            break;
        case "C":
            i = 2;
            break;
        default
            i = 0;
            break;
    }
```

Константа може бути константним виразом. Дві позначки `case` не можуть набувати однакового значення.

Цикли

C# реалізує чотири типи циклів. Цикл `for` має такий синтаксис:

```
for ([ініціалізатор]; [умова]; [ітератор])
    оператор(и)
```

Тут *ініціалізатор* – вираз, який обчислюється до початку виконання циклу (зазвичай, тут ініціалізується локальна змінна, яку використовують як лічильник циклу), *умова* – вираз, який обчислюється перед виконанням кожної ітерації циклу (наприклад, перевірка того, що лічильник циклу менший за деяке значення), *ітератор* – вираз, який виконується після кожної ітерації циклу (наприклад, зміна лічильника циклу).

Кожен (або всі) із цих елементів може бути пропущений.

Цикл for є циклом із передумовою. Таким є й цикл `while`:

```
while (умова)
    оператор(и)
```

Цикл do...while є прикладом циклу з постумовою:

```
do
    оператор(и)
while (умова);
```

Оскільки *умова* перевіряється після виконання тіла циклу, то хоча б одна ітерація виконається обов’язково. Усі розглянуті цикли завершують свої ітерації, якщо *умова* набуде значення `false`.

C# пропонує ще один механізм циклів – `foreach`:

```
foreach (тип ідентифікатор in вираз)
    оператор(и)
```

Тут *тип* – тип ідентифікатора, *ідентифікатор* – змінна циклу, що представляє елемент колекції або масиву, а *вираз* – об'єкт колекції або масив (або вираз над ними).

Цикл `foreach` дає змогу проводити ітерацію по кожному об' єкту в контейнерному класі, який підтримує інтерфейс `IEnumerable`. До контейнерних класів належать масиви C#, класи колекцій у просторі імен `System.Collection` та означені користувачем класи колекцій:

```
int even = 0, odd = 0;
int[] arr = new int [] {0,1,2,5,7,8,11};
foreach (int i in arr)
{
    if (i%2 == 0)
        even++;
    else
        odd++;
}
```

Зауважимо, що значення об'єкта в колекції всередині циклу `foreach` змінювати не можна.

Оператори переходу

Програма C# може містити позначки – ідентифікатор із двокрапкою. Оператор `goto` дає змогу передати керування стрічці програми з позначкою:

```
goto позначка;
```

Не можна передавати керування у блок коду, за межі класу, а також не можна вийти з блоку `finally`, розташованого після блоків `try...catch`. Оператор `goto` може використовуватися для переходів усередині оператора `switch`. У цьому випадку синтаксис такий:

```
goto case константа;
goto default;
```

Оператор `break` використовують для виходу з коду позначки в операторі `switch`, а також для виходу із циклів `for, foreach, while` та `do...while`.

У циклах також можна використовувати оператор continue, який перериває поточну ітерацію та ініціює наступну.

Оператор `return` використовують для припинення роботи поточного методу та передачі керування в метод, який його активізував. Якщо метод повертає значення, то `return` повинен повернути значення відповідного типу:

`return [`*`вираз`*`];`

Аудиторні завдання

Завдання № 1

Розглянемо програму, яка узагальнює наведені вище теоретичні відомості про основні конструкції, керуючі оператори та синтаксис мови C#.

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            //Hello World
            Console.WriteLine("Hello   World!!!");     //Обычный
текст
            Console.WriteLine("Нажмите любую клавишу...");
            Console.ReadKey(); //Ожидание ввода

            //Вывод числовых типов
            int x = 4;
            double y = 20.2;
```

```
            Console.WriteLine("У меня было {0} яблока и {1}
рублей", x, y); //Вывод числовых значений
            Console.WriteLine("Нажмите любую клавишу...");
            Console.ReadKey();

            //Вывод строкового типа
            string str = "У меня было 4 яблока и 20.2 рублей";
            Console.WriteLine(str); //Вывод строчного типа
            Console.ReadKey();

            //Объявление одномерного массива
            int[] intArray = new int[10];
            //Объявление двухмерного массива
            int[,] intDoubleArray = new int[10, 10];

          //Инициализация одномерного массива(пример цикла for)
            for (int i = 0; i < 10; i++)
            {
                intArray[i] = i;
            }

          //Инициализация двухмерного массива(пример цикла for)
            for (int i = 0; i < 10; i++)
            {
                for (int j = 0; j < 10; j++)
                {
                    intDoubleArray[i, j] = i + j;
                }
            }

          //Сумма чисел одномерного массива(пример цикла while)
            int count = 0;
            int sum = 0;
            while (count < intArray.Length)
            {
                sum = sum + intArray[count];
                count++;
            }
```

```
            Console.WriteLine("Сумма элементов массива равна
Sum={0}", sum);
            Console.WriteLine("Нажмите любую клавишу...");
            Console.ReadKey();
        }
    }
}
```

Завдання № 2

Реалізувати програму обчислення суми квадратів двох цілих чисел, які вводить користувач.

```
using System;
namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            string str;

            Console.Write("Введите переменную X: ");
            str = Console.ReadLine();
            int X = Convert.ToInt32(str);

            Console.Write("Введите переменную Y: ");
            str = Console.ReadLine();
            int Y = Convert.ToInt32(str);

            Console.WriteLine("X^2+Y^2 = {0}", X*X + Y*Y);
            Console.ReadKey();
        }
    }
}
```

Як бачимо, для введення даних користувачем була використана проміжна строкова змінна `string str;` яка потім за допомогою класу `Convert` конвертувалася в ціле число. Таке перетворення є найбільш

уразливим місцем у даній програмі, оскільки у випадку введення У випадку букви замість цілого числа виникне помилка та генеруватиметься виняток `System.FormatException`.

Завдання до лабораторної роботи

Відповідно до варіанта та обраного рівня складності виконати наступні завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Завдання № 1	Завдання № 1
2	–	Завдання № 2	Завдання № 2
3	–	–	Завдання № 3

Завдання № 1

№ варіанта	Завдання
1	Вивести на консоль своє ім’я як текст та групу як строкову змінну.
2	Реалізувати програму зведення в куб добутку двох цілих чисел.
3	Вивести на консоль своє прізвище як змінну строкового типу та свій вік як змінну типу `int`.
4	Реалізувати програму зведення в квадрат частки двох цілих чисел.
5	Створити дві змінні типу `int`, знайти їх добуток та суму, вивести результат на консоль.
6	Створити дві змінні типу `float`, знайти їх середнє арифметичне, результат вивести на консоль.
7	Реалізувати програму зведення в квадрат суми двох цілих чисел.

№ варіанта	Завдання
8	Створити дві змінні типу `double`, знайти їх суму та різницю, результат вивести на консоль.
9	Вивести на консоль своє ім’я та прізвище як текст.
10	Створити дві змінні типу `double`, знайти їх суму та вивести на консоль
11	Реалізувати програму обчислення добутку двох дійсних чисел.
12	Реалізувати програму обчислення частки двох дійсних чисел.
13	Реалізувати програму зведення цілого числа в квадрат.
14	Реалізувати програму зведення в куб цілого числа.
15	Реалізувати програму зведення в куб суми двох цілих чисел.
16	Реалізувати програму зведення в квадрат добутку двох цілих чисел.
17	Створити дві змінні типу `string`, склеїти їх та вивести на консоль.
18	Реалізувати програму зведення в квадрат суми двох дійсних чисел.
19	Реалізувати програму зведення в квадрат різниці двох дійсних чисел.
20	Реалізувати програму зведення в квадрат добутку двох дійсних чисел.
21	Реалізувати програму зведення в квадрат частки двох дійсних чисел.
22	Створити дві змінні типу `int`, знайти їх різницю, якщо вона більше за нуль, вивести її квадрат, якщо менша за нуль, то її модуль.
23	Реалізувати програму зведення в квадрат різниці двох цілих чисел.

№ варіанта	Завдання
24	Реалізувати програму зведення в куб різниці двох цілих чисел.
25	Створити дві змінні типу `double`, знайти їх суму та вивести на консоль.
26	Реалізувати програму зведення в куб частки двох цілих чисел.
27	Реалізувати програму зведення в куб суми двох дійсних чисел.
28	Реалізувати програму зведення в куб різниці двох дійсних чисел.
29	Реалізувати програму зведення в куб добутку двох дійсних чисел.
30	Реалізувати програму зведення в куб частки двох дійсних чисел.

Завдання № 2

№ варіанта	Завдання
1	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти їх суму та добуток.
2	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за зростанням та вивести на консоль.
3	Створити, ініціалізувати масив з 20 цілих чисел, знайти їх середнє арифметичне.
4	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), вивести на консоль усі числа, більші за число 10.
5	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за спаданням та вивести на консоль.
6	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), вивести на консоль усі числа, менші за число 15.

№ варіанта	Завдання
7	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти кількість чисел більших за 10 та вивести отримане число на консоль.
8	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), вивести на консоль усі числа, менші за число 20, але більші за 10.
9	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти добуток чисел, які менші за середнє арифметичне чисел масиву.
10	Створити, ініціалізувати масив з 10 чисел типу double у проміжку (0..25), знайти суму чисел більших за 15 і добуток чисел менших за 10.
11	Створити, ініціалізувати масив з 8 цілих чисел у проміжку (0..15), знайти їх суму та частку.
12	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за спаданням та вивести на консоль.
13	Створити, ініціалізувати масив з 15 цілих чисел, знайти їх середнє геометричне.
14	Створити, ініціалізувати масив з 12 цілих чисел у проміжку (0..35), вивести на консоль усі числа, більші за число 15.
15	Створити, ініціалізувати масив з 15 цілих чисел, відсортувати його за спаданням та вивести на консоль.
16	Створити, ініціалізувати масив з 15 цілих чисел у проміжку (0..40), вивести на консоль усі числа, менші за число 25.
17	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..50), знайти кількість чисел більших за 40 та вивести отримане число на консоль.
18	Створити, ініціалізувати масив з 15 цілих чисел у проміжку (0..18), вивести на консоль усі числа, менші за число 15, але більші за 5.
19	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти добуток чисел, які менші за середнє арифметичне чисел масиву.

№ варіанта	Завдання
20	Створити, ініціалізувати масив з 10 чисел типу `double` у проміжку (0..50), знайти суму чисел більших за 15 і добуток чисел менших за 12.
21	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (10..50), знайти їх суму та добуток.
22	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за зростанням та вивести на консоль.
23	Створити, ініціалізувати масив з 20 цілих чисел, знайти їх середнє арифметичне.
24	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (5..25), вивести на консоль усі числа, більші за число 12.
25	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за спаданням та вивести на консоль.
26	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (5..35), знайти добуток чисел, які менші за середнє арифметичне чисел масиву.
27	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти кількість чисел більших за 10 та вивести отримане число на консоль.
28	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (10..60), вивести на консоль усі числа, менші за число 20, але більші за 10.
29	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), вивести на консоль усі числа, менші за число 15.
30	Створити, ініціалізувати масив з 15 чисел типу `double` у проміжку (2..5), знайти суму чисел більших за 3 і добуток чисел менших за 4.

Завдання № 3

Дано матрицю розміру M×M, що заповнена цілими числами у проміжку (-10, 10). Необхідно:

№ варіанта	Завдання
1	Відсортувати за зростанням головну діагональ матриці.
2	Знайти суму непарних рядків матриці.
3	Знайти добуток парних стовпців матриці
4	Відсортувати побічну діагональ матриці за спаданням
5	Знайти усі елементи матриці, що більші за нуль, та знайти їх суму.
6	Відсортувати перший та останній стовпець матриці за спаданням.
7	Знайти максимальний елемент кожного стовпця матриці.
8	Знайти мінімальній елемент кожного рядка матриці.
9	Відсортувати за спаданням парні стовпці матриці
10	Знайти максимальний та мінімальний елемент матриці, та їх суму.
11	Відсортувати за зростанням побічну діагональ матриці.
12	Знайти суму парних рядків матриці.
13	Знайти добуток непарних стовпців матриці
14	Відсортувати побічну діагональ матриці за зростанням
15	Знайти усі від’ємні елементи матриці та знайти добуток їх модулів.
16	Відсортувати перший та останній стовпець матриці за спаданням.
17	Знайти мінімальний елемент кожного стовпця матриці.
18	Знайти максимальний елемент кожного рядка матриці.
19	Відсортувати за зростанням парні стовпці матриці.
20	Знайти максимальний та мінімальний елемент матриці, та їх добуток.
21	Знайти усі елементи матриці, що більші за нуль, та знайти їх суму.
22	Відсортувати перший та останній стовпець матриці за спаданням.
23	Знайти максимальний елемент кожного стовпця матриці.
24	Знайти мінімальній елемент кожного рядка матриці.

25	Відсортувати за спаданням парні стовпці матриці.
26	Знайти максимальний та мінімальний елемент матриці, та їх суму.
27	Відсортувати за зростанням побічну діагональ матриці.
28	Знайти суму парних рядків матриці.
29	Знайти добуток непарних стовпців матриці.
30	Відсортувати побічну діагональ матриці за зростанням.

Контрольні питання

1. Коротко охарактеризуйте платформу Microsoft .NET Framework.
2. Які типи даних існують у мові C#?
3. Чим відрізняються типи за значенням від типів за посиланням?
4. Як перетворити строкову змінну у ціле число?
5. Чим відрізняється мова C# від C++?
6. Перелічіть основні бінарні операції та пріоритет їх виконання.
7. Для чого призначені простори імен?
8. Наведіть синтаксис основних операторів керування.
9. Як відкомпілювати файл програми у командному рядку?
10. Чим відрізняються настроювання проекту у режимі компіляції *Debug* та *Release*?

ЛАБОРАТОРНА РОБОТА № 2

Тема: основи об’єктно-орієнтованого програмування на мові C#. Поняття класу та об’єкта.

Мета: розробити програми з використанням класів C#, реалізувати перевантаження методів, наслідування класів та дослідити принципи інкапсуляції.

Теоретичні відомості

Класи та структури

Алгоритмічна мова C# є цілковито об’єктно-орієнтованою. Програма на C# – це набір класів та маніпулювання ними [8]. Класи та структури дуже подібні між собою. Основні розбіжності між ними:

- об’єкти класів мають тип даних за посиланням, а структури – за значенням;
- структури не підтримують спадковість;
- для структури не можна оголосити конструктор за замовчуванням;
- для структури не можна оголосити деструктор;
- неможливо використати ініціалізатори для надання значень полям.

Оскільки екземпляри структур розташовані у стеку, доцільно їх використовувати для представлення невеликих об’єктів. Зокрема, клас `Point`, який розглядатимемо нижче, з успіхом можна було б реалізувати як структуру. Вибір класу зумовлено лише бажанням продемонструвати на простій логічній побудові більше понять класу та структури.

Розглянемо поняття класу на такому прикладі:

```
public class Point
{
    //статичне поле
    private static uint count = 0;
    //поля
    public double x = 0; public double y = 0;
```

```
    //властивість лише для читання
    public static uint Count { get {return count;}
}
//перекритий метод
public override string ToString()
{
    return "(" + x + "," + y + ")";
}
// метод
public void Set(double x, double y)
{
    this.x = x; this.y = y;
}
//конструктор без параметрів
public Point ()
{
    count++;
}
//конструктор з параметрами
public Point(double x, double y)
{
    count++;
    Set(x, y);
}
//конструктор копій
public Point(Point a): this(a.x, a.y)
{
}
//деструктор
~Point( )
{
    count--;
}
```

За допомогою модифікатора класу `partial` оголошення класу можна розбити на декілька частин, які можна розташувати як в одному, так і в різних файлах.

Члени класу

Дані та функції всередині класу називають членами класу. Дані класу – це *поля, константи* та *події,* тобто ті члени класу, які містять дані для класу [8].

Поле – це довільна змінна, оголошена на рівні класу: `count, x, y`. Якщо поле оголошене як статичне (`static`), то воно єдине для всього класу і жоден екземпляр об'єкта класу не матиме окремого екземпляра цього поля. У протилежному випадку для кожного екземпляра об'єкта цього класу утворюються власні незалежні екземпляри полів.

Клас `Point` описує точку: кожен екземпляр класу – це точка на площині з координатами `x` та `y`. Статичне поле `count` містить кількість утворених і активних в поточний момент часу екземплярів класу.

Подія – це член класу, який дає змогу об'єкту надіслати повідомлення про початок певної події (зміна значення поля, взаємодія з користувачем і т.п.). Клієнт (код, який використовує об'єкт класу) може містити код обробки події.

Функції класу – це ті члени класу, які забезпечують функціональність для роботи з даними класу. Вони містять *методи, властивості, конструктори, деструктори, оператори* та *індексатори.*

Модифікатори змінних

Модифікатори змінних описують ті чи інші особливості оголошуваних змінних. Перелічимо модифікатори змінних: `internal, new, private, protected, public, readonly, static, const`.

Модифікатори застосовують лише до полів (членів класів), однак не до локальних змінних. Змінна може мати декілька модифікаторів.

Модифікатор `new` використовують лише в класах, які є похідними від іншого, поля якого потрібно приховати.

Такі модифікатори утворюють групу модифікаторів доступу:

- `public` – змінна доступна скрізь як поле типу, якому вона належить;
- `internal` – змінна доступна лише поточному складеному модулю;

– `protected` – доступ до змінної відкритий лише з класу, якому вона належить, або з похідних класів;

– `protected internal` – те ж саме, що й `protected`, однак доступ лише з поточного складеного модуля;

– `private` – доступ до змінної лише з класу, у якому її оголошено.

За замовчуванням, поле є членом екземпляра класу. Тобто кожен екземпляр класу має власне поле (виділену ділянку пам'яті) із відповідним ідентифікатором. Якщо до оголошення поля додати модифікатор `static`, то це поле буде єдиним для всіх екземплярів класу.

Модифікатор `readonly` робить змінну доступною лише для читання. її значення можна встановити при першому оголошенні (для статичних змінних) або в конструкторі для типу, до якого вона належить (для нестатичних змінних).

Методи класу

Види функцій класу визначаються синтаксисом оголошення. Синтаксис оголошення методів у C# такий:

```
[модифікатори] тип_результату НазваМетоду([параметри])
{
    //тіло методу
}
```

Модифікатори методу аналогічні модифікаторам змінних. Додатково можна використовувати модифікатори, зазначені в таблиці 2.1.

Клас `Point` містить перекритий (`override`) метод `ToString` класу `object`. Для активізації методу потрібно вказати ім'я об'єкта, для якого активізують метод, а після крапки – ім'я методу та список аргументів у дужках:

```
Point a = new Point(1, 1);
string s = a.ToString();
```

Таблиця 2.1 – Моифікатори методів класу

Модифікатор	Опис

`virtual`	Метод може бути переозначений у дочірньому класі
`abstract`	Віртуальний метод, який визначає сигнатуру методу, однак не містить його реалізації. За наявності абстрактних методів екземпляр класу не може бути утворений
`override`	Метод переозначує успадкований віртуальний або абстрактний метод
`sealed`	Метод переозначує успадкований віртуальний метод і забороняє його переозначення у дочірніх класах. Використовують разом із `override`
`extern`	Метод реалізований поза програмою на іншій мові

Для активізації статичного методу потрібно зазначити назву класу, а не ім’я об’єкта. Статичний метод може працювати лише зі статичними полями класу. Якщо метод активізується всередині класу, то назву класу не використовують.

Аргументи методів можуть передаватися *за посиланням* або *за значенням.* За замовчуванням параметри передаються за значенням, тобто метод одержує копію значення аргументу. Якщо параметр має тип даних за значенням, то довільні зміни цього параметра всередині методу ніяк не вплинуть на значення аргументу-оригіналу. Якщо ж параметр має тип за посиланням (масив, клас), то метод працюватиме безпосередньо з даними аргументу, оскільки передана копія значення – адреса розташування об’єкта. Зауважимо, що стрічки не поводять себе як тип за посиланням, отож зміни у стрічці, що відбулися всередині методу, не зачіпають її оригінал.

Якщо потрібно змінні за значенням передати в метод за посиланням, використовують ключове слово `ref` перед типом параметра. Слово `ref` повинно вказуватися і при виклику методу. Довільна модифікація змінної методом викликає відповідні зміни аргументу-оригіналу.

Перед передачею значень методу всі змінні-аргументи повинні бути ініціалізованими. Інакше компілятор C# видасть повідомлення про помилку. Обійти це обмеження можна шляхом використання ключового слова `out` перед типом параметра. Слово `out` необхідно

зазначити і при виклику методу. Усередині методу параметрові, позначеному як out, необхідно присвоїти значення.

Властивості класу

Властивості використовують з метою зробити виклик методу подібним на поле. їх і оголошують подібно до поля. Однак додатково після оголошення у фігурних дужках розташовують блок коду для контролю даних та реалізації потрібної функціональності. Цей блок може мати два методи доступу: аксесор get та аксесор set.

Клас Point має властивість Count, яка повертає кількість утворених екземплярів класу. Розширимо опис цієї властивості:

```
public static uint Count
{
    get
    {
        return count;
    }
    set
    {
        if (value >= 0)
            count = value;
    }
}
```

Зауважимо, що аксесору set неявно передається параметр з іменем value такого ж типу, як і властивість. Аналогічно, значення типу властивості повинен повертати аксесор get.

Оскільки властивість подібна до поля, то й активізацію її здійснюють подібно:

```
uint cnt = Point.Count;  // аксесор get
Point.Count = cnt;       // аксесор set
```

Якщо властивість містить лише аксесор get, то її використовують тільки для читання значення. Якщо властивість містить лише аксесор set, то її використовують тільки для запису значення.

Конструктори класу

Конструктори – це методи класу, які використовуються разом з оператором `new` для утворення об'єкта. Якщо клас не містить власного конструктора, то компілятор утворить конструктор за замовчуванням (без параметрів). Конструкторів може бути кілька. Вони відрізнятимуться кількістю і типом параметрів, однак матимуть одну й ту ж назву – ім'я класу. Клас `Point` містить три конструктори. Перший із них збільшує лічильник утворених об'єктів, а другий додатково ініціалізує поля x та y. Третій конструктор `public Point (Point a)` належить до так званих *конструкторів копій,* оскільки ініціалізує екземпляр класу на основі значень іншого екземпляра цього ж класу. Для утворення об'єкта можна використати довільний з конструкторів класу, проте лише один.

Конструктор не може повертати значення, отож тип результату не вказують при визначенні конструктора. Якщо конструктор оголошений як `private`, то його можна використовувати лише всередині класу, а якщо як `protected` – то в класах-нащадках. Обмеження доступу до конструктора може бути корисним, наприклад, для методів `Clone(), Copy()` або для аналогічних методів класу, яким потрібно утворювати інші екземпляри цього класу. Очевидно, що клас повинен мати хоча б один конструктор з доступом `public`, якщо передбачається утворення об'єктів цього класу клієнтським кодом.

Клас може також мати статичний конструктор без параметрів. Такий конструктор буде використано лише один раз. Його можна використати для ініціалізації статичних змінних. Наприклад,

```
static Point()
{
    count = 0;
}
```

Статичний конструктор не має модифікатора доступу, оскільки він активізується лише середовищем .NET при завантаженні класу. Зауважимо, що клас може водночас мати конструктор екземплярів без

параметрів і статичний конструктор. Це єдиний випадок, коли може бути два методи класу з однаковими назвами та однаковим списком параметрів. Конструктор може викликати інший конструктор цього ж класу. Для цього випадку існує спеціальний синтаксис:

```
public Point(): this(0,0)
{
    // додатковий код
}
```

Такий синтаксис повідомляє компілятору, що у випадку використання конструктора спочатку потрібно виконати інший конструктор цього класу з переданими йому аргументами після ключового слова `this`, а потім виконати код (якщо є) зазначеного конструктора.

Ключове слово `this` вказує, що використовують елемент поточного класу. Якщо ми використаємо ключове слово `base`, то дамо вказівку шукати відповідний елемент у базовому класі.

Деструктори класу

Деструктор класу використовують для виконання дій, необхідних при знищенні екземпляра класу. У нашому прикладі класу `Point` деструктор зменшує лічильник екземплярів класу.

Як і конструктор, деструктор має те ж ім’я, що й клас, однак з префіксом тильда (~): `~Point`. Деструктор не має параметрів та не повертає результат. Явним чином деструктори у C# не викликають. Виконання коду деструктора ініціюється механізмом прибирання «сміття». Отож не можна передбачити, коли буде виконано код деструктора.

Ініціювати негайне прибирання «сміття» можна з допомогою методу `Collect()` об’єкта .NET `System.GC`, який реалізує «прибиральника». Однак цим доцільно користуватися лише у випадку, коли ви впевнені в необхідності такого кроку. Якщо при знищенні екземпляра класу необхідно негайно звільнити ресурси, зайняті екземпляром класу (наприклад, закрити файл) або надіслати повідомлення іншим об’єктам, доцільно утворити спеціальні методи.

Типові назви таких методів – `Close` та `Dispose`. Клієнтський код повинен явно активізувати ці методи. І це є недоліком такого підходу.

Інший варіант звільнення ресурсів – використання оператора `using`. У цьому випадку клас повинен успадковувати інтерфейс `IDisposable`, означений у просторі імен `System`:

```
class Point: IDisposable
{
    // – – –
    public void Dispose ()
    {
        // код
    }
}
```

Перевантаження операцій

Нехай задано точки $A(x_1, y_1)$ та $B(x_2, y_2)$. Точку $C(x, y)$ назвемо сумою точок *A* та B, якщо $x = x_1 + x_2$, $y = y_1 + y_2$. Для додавання точок у класі `Point` можна утворити метод. Наприклад:

```
public Point Add(Point p)
{
    //код
};
```

Тоді додавання виглядатиме так:

```
C = A.Add(B);
```

Якщо потрібно додати декілька точок, то вираз ускладниться. Значно зручніше використовувати звичний нам синтаксис для простих типів:

```
C = A + B;
```

C# дає змогу перевантажувати операції. Наприклад, до означення класу Point можна додати такий код:

```
public static Point operator + (Point p1, Point p2)
{
    return new Point(p1.x + p2.x, pl.y + p2.y);
```

```
}
```

Операція оголошується аналогічно методу, за винятком того, що замість імені методу пишуть ключове слово `operator` і знак операції. Тепер, якщо `A`, `B` та `C` мають тип `Point`, то ми можемо записати:

```
C = A + B;
```

Перевантажимо тепер операцію множення на число.

```
public static Point operator * (double a, Point p)
{
    return new Point(a * p.x, a * p.y);
}
public static Point operator * (Point p, double a)
{
    return a*p;
}
```

Для операції множення ми утворили два варіанти перевантаження, щоб компілятор коректно сприймав, наприклад, код `10*A` та `A*10`. Зауважимо, що перевантаження операцій `+`, `-`, `*` та `/` використовуються компілятором для реалізації операцій `+=`, `-=`, `*=` та `/=` відповідно. C# дає змогу перевантажувати лише операції, зазначені в таблиці 2.2.

Таблиця 2.2 – Операції, які дозволяють перевантаження

Категорія	Операції
Арифметичні	`+ - * / % ++ --`
Бітові	`<< >> & | : ! ~ true false`
Порівняння	`== != < >= <= >`

Перевантаження методів

Перевантаження методів використовують у тому випадку, коли потрібно, щоб клас виконував деякі дії, але при цьому існувало кілька способів передачі інформації методу, який виконує завдання. Ми вже

демонстрували перевантаження методів на прикладі конструкторів класу `Point` – одна назва за різних наборів параметрів.

Наведемо ще один приклад – перевантаження методу `ToString`:

```
public string ToString(string format)
{
    return String.Format(format, x, y);
}
```

Тепер можна записати код:

```
Point p = new Point(1,1);
string s = p.ToString();               //s = "x=1 y=1"
s = p.ToString("x:{0} y:{1}", x, y);  //s = "x:1 y:1"
```

Кількість перевантажених методів не обмежена. Тобто клас може містити багато методів з одним іменем, однак вони повинні вирізнятися кількістю, порядком або типом параметрів. Очевидно, що не доцільно давати однакове ім’я методам, які виконують зовсім різні задачі.

Приклад використання об’єктів класу

Розглянемо клас, який реалізує функціональність роботи з масивом точок. Наведемо код початкового варіанта такого класу:

```
public class Points
{
    private readonly uint count;
    protected Point[] points;
    public uint Count
    {
        get
        {
            return count;
        }
    }
    public override string ToString()
    {
        string Result = "";
```

```
            for (int i = 0; i < count; i++)
                Result += points[i] + " ";
            return Result;
        }
        public Points(uint count)
        {
            this.count = count;
            points = new Point[count];
            for (int i = 0; i < count; i++)
                points[i] = new Point();
        }
        public Points(Points pts): this(pts.count)
        {
            for (int i = 0; i < count; i++)
                points[i] = pts.points[i];
        }
    }
```

Клас `Points` містить масив точок `points` (елементів типу `Point`). Розмірність масиву задається параметром конструктора і встановлюється при утворенні об’єкта класу:

```
points = new Point[count];
```

Зверніть увагу, що цей код виокремить пам’ять для розташування count вказівників на об’єкти Point, а не власне об’єктів. Тим більше, що самих об’єктів ще не існує. їх утворюють такі дві стрічки:

```
for (int i = 0; i < count; i++)
    points[i] = new Point();
```

Властивість `Count` повертає значення розмірності. Оскільки клас містить конструктори, то конструктор за замовчуванням не утворюється. Отож для утворення об’єкта класу можна використати або конструктор з параметром, який задає розмірність, або конструктор копій.

Індексатори класу

Індексатори дають змогу здійснювати доступ до об’єкта так, ніби він є масивом. Індексатори означуються приблизно так, як

властивості – з використанням функцій get та set. Однак замість імені індексатора використовують ключове слово this.

Якщо ps – об'єкт типу Points, то для доступу до точки з індексом 0 ми повинні використовувати синтаксис: ps.points[0]. Значно елегантніше було б застосувати ps[0], однак для цього потрібно додати індексатор.

Щоб оголосити індексатор для класу Points, додамо до його опису такий код:

```
public Points this[int i]
{
    get
    {
        if (i >= 0 && i < count)
            return points[i];
        else
            throw new IndexOutOfRangeException( "Вихід за
            допустимий діапазон індексів"+ i);
    }
    set
    {
        if (i >= 0 && i < count)
            points[i]=value;
        else
            throw new IndexOutOfRangeException( "Вихід за
            допустимий діапазон індексів"+ i);
    }
}
```

Тепер для змінної ps типу Points ми можемо використати код:

```
string s = "x0=" + ps[0].x + "  y0=" + ps[0].y;
```

Індексатори не є обмежені одномірними масивами та цілочисельними індексами. Для індексаторів можна застосовувати цикли for, do та while, однак не можна написати цикл foreach, оскільки він працює лише з колекціями, а не з масивами.

Успадкування класів

Побудуємо клас, який представлятиме геометричний трикутник. Трикутник визначається трьома точками на площині. Використаємо клас Points, який дає змогу будувати множину точок і проводити деякі дії над ними.

```
    public class Triangle: Points
    {
        public Triangle(double xl, double yl, double x2,
                        double  y2,  double  x3,  double  y3):
base(3)
        {
            points[0].Set(x1, yl);
            points[1].Set(x2, y2);
            points[2].Set(x3, y3);
        }
        public Triangle (Point pl, Point p2, Point p3): base (3)
        {
            points[0] = p1; points[1] = p2; points[2] = p3;
        }
        public override string ToString()
        {
            return "трикутник " + base.ToString();
        }
    }
```

Код `class Triangle:Points` оголошує, що клас `Triangle` успадковує клас `Points`. Це означає, що `Triangle` має всі компоненти класу `Points`: поля `count, points`, метод `ToString` та індексатор. Окрім того, оскільки клас Points успадковує клас `object` (як і всі типи C#), то клас `Triangle` має також усі компоненти класу `object`.

Класи `object` і `Points` є класами-предками для класу `Triangle`. Клас `Points` (клас, який зазначено після двокрапки в оголошенні нового класу) називають *базовим* або *батьківським* класом для класу `Triangle`. Клас `Triangle` називають *похідним* або *дочірнім* для класу `Points`. А для класів `object` і `Points` він буде *нащадком.* Похідний клас може безпосередньо використовувати всі члени базового класу, якщо вони означені з модифікаторами `protected` або `public`.

Однак похідний клас не наслідує конструкторів базового класу (проте може використати, як це зроблено в нашому прикладі). Єдиний виняток – це конструктор за замовчуванням, який викликається конструктором за замовчуванням похідного класу.

Якщо похідний клас наслідує всі члени предків, то об'єкт цього класу містить підмножину, яку можна розглядати як об'єкт деякого класу-предка. Наприклад:

```
Triangle T = new Triangle(1,1,2,2,3,3);
Points P = (Points)T;
object obj = (object)T;
IEnumerable ienum = (IEnumerable)T;
```

Приховування компонентів

C# дає змогу замінити члени базового класу в нащадках. Розглянемо такі два класи:

```
public class A
{
    public int x = 0;
}
public class B: A
{
    public int x = 1;
}
```

Клас B успадковує клас A, отже – і поле x. Однак у класі B оголошене нове поле з ідентичним іменем. У цьому випадку компілятор не є упевненим, що він розуміє логіку програміста, отож видасть повідомлення щодо своїх сумнівів. Проте код буде все ж скомпільовано.

Надання новим членам похідного класу імен, вже використаних у базовому – потенційна небезпека помилок. Однак інколи така потреба виникає. У цьому випадку потрібно приховати компонент x класу A, оголосивши явно компонент x класу B новим за допомогою ключового слова new:

```
public class B: A
```

```
{
    public new int x = 1;
}
```

Для об’єкта класу `B` можемо отримати значення обох компонентів x:

```
B b = new B ();
int bx = b.x;      //bx набуде значення 1
int ax = ((A)b).x; //ax набуде значення 0
```

Приховування методів може стати необхідним у випадку конфлікту версій базового класу. Припустимо, що програміст *A* розробив базовий клас A, а програміст *B* на основі класу A – клас B, у який додав новий метод з назвою M. Через деякий час *A* дописує в класі A новий метод з назвою M та публікує нову версію. Після перекомпілювання програми результат виконання програми може бути не таким, як очікував *B*.

Оскільки компілятор C# відстежує такі ситуації, то він видасть відповідне повідомлення. Програміст *B* має два варіанта дій. Якщо він контролює усі класи, породжені від класу B, то краще перейменувати свій метод M. Якщо ж клас B опублікований для використання іншими користувачами, то до оголошення методу M необхідно додати модифікатор `new`.

Абстрактні методи

Нехай проектується деякий клас A. Вважають, що всі його дочірні класи повинні мати деякий метод M. Однак на рівні класу A недостатньо інформації для змістовного означення цього методу. Якщо існує необхідність присутності методу M у класі A, то цей метод можна оголосити з модифікатором abstract без реалізації. У цьому випадку метод M називатиметься *абстрактним*. Якщо клас містить абстрактні методи, то він повинен також містити модифікатор abstract. Наприклад:

```
abstract public class A
{
    abstract public void M();
}
```

Зауважимо також:

- неможливо утворити об’єкт абстрактного класу;
- неможливо оголосити конструктор абстрактним методом;
- абстрактні класи використовуються для породження інших класів;
- дочірні класи (якщо вони не є також абстрактними) зобов’язані містити реалізацію усіх абстрактних методів, успадкованих від базового класу.

Віртуальні методи

Продовжимо розгляд класу `Points`. Об’єкт цього класу містить індексовану множину точок. Ці точки можна розглядати як вузли ламаної лінії. Тоді можна ввести поняття довжини та оголосити метод `GetLength`, який повертає цю довжину.

Похідний від `Points` клас `Triangle` успадкує цей метод `GetLength`. Однак для трикутника довжина – це периметр. А успадкований `GetLength` не враховує в довжині пряму, що з’єднує останню точку з першою.

Якщо може виникнути потреба у зміні реалізації деякого методу в дочірніх класах, його потрібно оголошувати *віртуальним.* З цією метою використовують модифікатор `virtual`.

Додамо метод `GetLength` до класу `Points`:

```
public class Points : IEnumerable
{
    public double GetDistance(Point pl, Point p2)
    {
        return Math.Sqrt((pl.x – p2.x)  * (pl.x – p2.x) +
                         (pl.y – p2.y) * (pl.y – p2.y));
    }
    public virtual double GetLength()
    {
        double length = 0;
        for (int i = 0; i < count – l; i++)
            length += GetDistance (Points[i],Points[i+l]);
```

```
            return length;
        }
    }
```

Метод GetLength тут оголошено віртуальним, оскільки поняття довжини може змінюватися в класах-нащадках. А от відстань між точками навряд чи потребуватиме переозначення. Тому метод GetDistance не описано як віртуальний.

Перекривання віртуальних методів

Ми вже розглядали перекривання віртуальних методів у класі Point, де перекривається успадкований від object віртуальний метод ToString:

```
public override string ToString()
```

У свою чергу в класі Points з таким самим синтаксисом перекривається успадкований уже від Point віртуальний метод ToString. Щоб перекрити віртуальний метод, означений у базовому класі, необхідно в похідному класі повторити оголошення методу, але модифікатор virtual замінити на модифікатор override. Очевидно, що потрібно також написати нову реалізацію методу. Наприклад:

```
public class Triangle: Points
{
    public override double GetLength()
    {
       return base.GetLength() + GetDistance(points[Count-
1], points[0]);
    }
}
```

C# має модифікатор sealed, який використовують в парі з override і який дає вказівку заборонити перекривання методу в дочірніх класах – *запечатує.*

Поліморфізм

Механізм віртуальних функцій реалізує концепцію поліморфізму об’єктно-орієнтованого програмування.

Розглянемо такі два класи:

```
public class A
{
    public string Method()
    {
        return "A.Method";
    }
    public virtual string VirtualMethod()
    {
        return "A.VirtualMethod";
    }
}
public class B: A
{
    public new string Method()
    {
        return "B.Method";
    }
    public override string VirtualMethod()
    {
        return "B.VirtualMethod";
    }
}
```

Клас B приховує успадкований метод `Method`, оголосивши новий з ідентичним іменем. А віртуальний метод `VirtualMethod` клас A перекриває. Оголосимо змінні:

```
A a = new A ();
B b = new B ();
A x = b;
```

Змінні `a` та `b` містять адреси утворених екземплярів, відповідно, класів A та B. Змінна `x` містить адресу того ж об'єкта, що й `b`, але має тип класу A. Наступний код демонструє особливості віртуальних функцій:

```
string s;
s = a.Method(); //"A.Method"
s = b.Method(); //"B.Method"
```

```
s = x.Method(); //"A.Method"
s = a.VirtualMethod(); //"A.VirtualMethod"
s = b.VirtualMethod(); //"B.VirtualMethod"
s = x.VirtualMethod(); //"B.VirtualMethod"
s = ((A)b).VirtualMethod(); //"B.VirtualMethod"
```

Якщо метод не є віртуальним, компілятор використовує той тип, який змінна мала при оголошенні. У нашому випадку x має тип A. Отож код x.Method() викличе метод класу A, хоча реально x є посиланням на об'єкт класу B. Якщо метод є віртуальним, компілятор згенерує код, який під час виконання перевірятиме, куди насправді вказує посилання, і використовуватиме методи відповідного класу. Хоча x має тип A, викликається метод VirtualMethod класу B. Окрім того, навіть явне приведення типу до A ситуацію не змінює.

Використаємо описану властивість поліморфізму для означення функції, яка повертає довжину об'єкта класу Points або Triangle.

```
public double PointsLength(Points points)
{
    return points.GetLength();
}
```

Оскільки метод GetLength є віртуальним у класах Points та Triangle, то функція PointsLength повертатиме коректні значення довжини для об'єктів різних типів:

```
Points Ps = new Points(3);
Ps[0] .Set (0, 0);
Ps[l] .Set (0, 3);
Ps[2] .Set (4, 0);
Triangle T = new Triangle(Ps[0], Ps[1], Ps [2]);
double pl = PointsLength(Ps);    // pl = 8
double tl = PointsLength(T);     // tl = 12
```

Аудиторні завдання

Завдання № 1

Розробити класи, які б реалізовували малювання геометричних фігур (точки, квадрата) у заданих координатах текстового режиму консолі за допомогою певного символу, наприклад «*». Побудувати ієрархію класів, а базовий клас при цьому об’явити абстрактним, реалізувати механізми наслідування та поліморфізму. У якості поліморфізму виконати перевантаження одного з методів базового класу. Побудувати UML-діаграму класів та діаграму послідовностей засобами Microsoft Visual Studio.

```
using System;

namespace Lab4_2
{
    class Pen // клас містить властивості олівця, зокрема колір
    {
        public ConsoleColor Color;
        public char Symbol;
        public Pen() // конструктор
        {
            Color = ConsoleColor.White; // білий колір
            Symbol = '*'; // за умовчанням малює зірочками
        }
    }

    abstract class Figure // об’єкт цього класу створити не
можна
    {
        public int x;
        public int y;
        public Pen pen = new Pen();
        public abstract void Draw();
    }

    class Point : Figure // клас точки
    {
```

```
    public override void Draw() // перевантаження методу
    {
        Console.ForegroundColor = pen.Color;
        Console.SetCursorPosition(x, y);
        Console.Write(pen.Symbol);
    }
}
class Square : Figure // клас квадрата
{
    public int size = 10; // розмір у символах
    public override void Draw() // перевантаження методу
    {
        int left = x - size / 2;
        int top = y - size / 2;
        for (int i = 0; i <= size; i++)
        {
            Console.ForegroundColor = pen.Color;
            Console.SetCursorPosition(left + i, top);
            Console.Write(pen.Symbol);
            Console.SetCursorPosition(left+i, top + size);
            Console.Write(pen.Symbol);
            Console.SetCursorPosition(left, top + i);
            Console.Write(pen.Symbol);
            Console.SetCursorPosition(left+size, top + i);
            Console.Write(pen.Symbol);
        }
    }
}
class Program
{
    static void Main(string[] args)
    {
        Figure fig = new Point();
        fig.x = 10;
        fig.y = 10;
        fig.Draw();
        Figure fig2 = new Square();
        fig2.x = 20;
        fig2.y = 10;
```

```
            fig2.Draw(); // приклад поліморфізму
            Console.ReadKey();
        }
    }
}
```

Для створення діаграми класів спершу до поточного рішення потрібно додати новий проект обравши пункт меню *Файл* | *Добавить*| *Создать проект…* (рисунок 2.1).

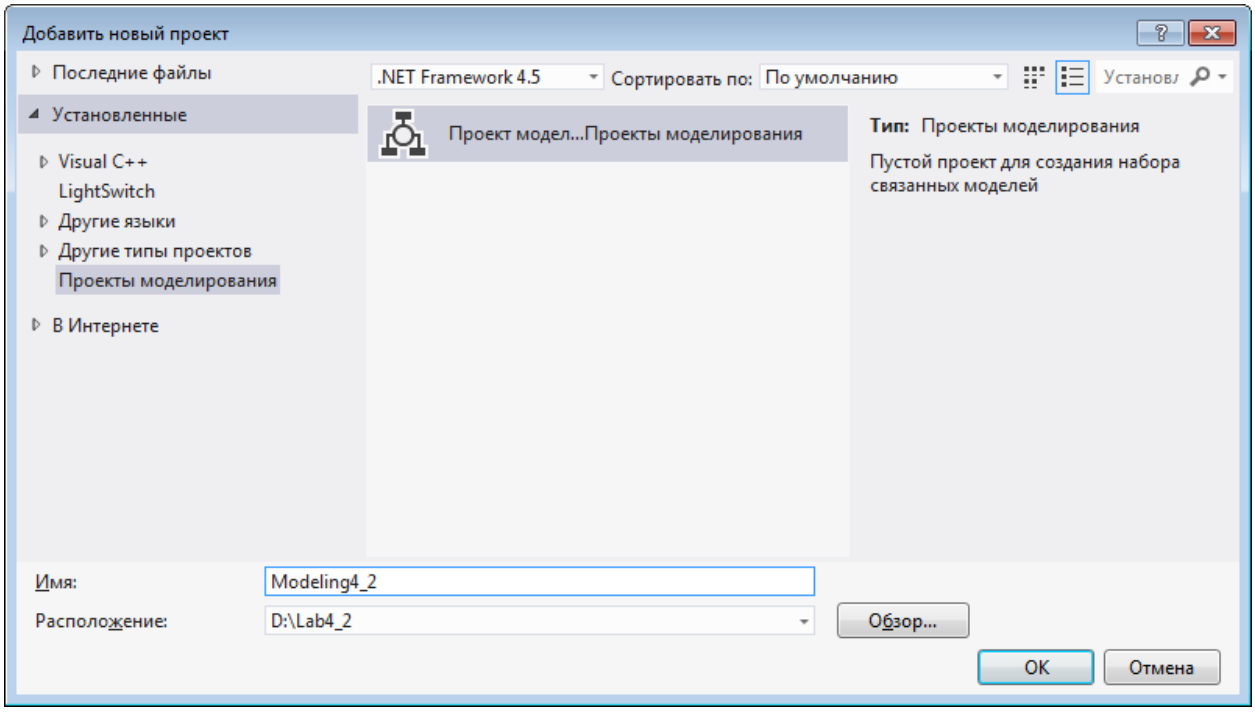

Рисунок 2.1 – Створення проекту моделювання

Наступним кроком потрібно додати діаграму класів UML до проекту, створеного раніше (рисунок 2.2).

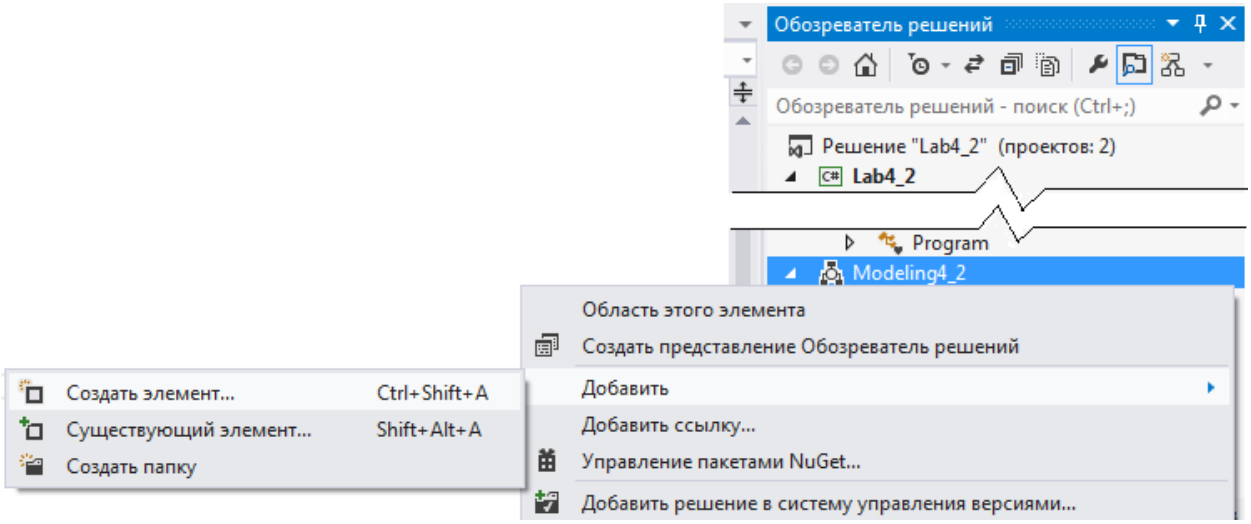

Рисунок 2.2 – Додавання нової діаграми до проекту моделювання

Після перенесення на робочу область всіх класів програмного коду одержимо діаграму, зображену на рисунку 2.3.

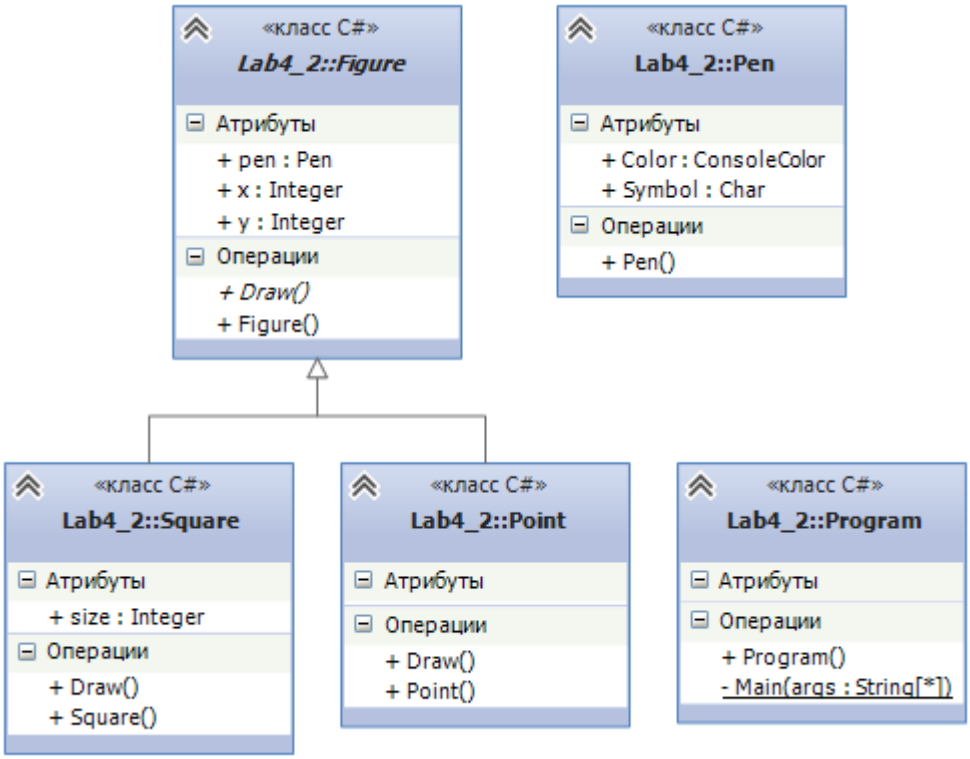

Рисунок 2.3 – Діаграма класів UML

Для відображення часу життя об’єктів та послідовності викликів методів використовується діаграма послідовностей UML. Для її побудови потрібно обрати фрагмент коду (наприклад, точку входу у програмний додаток) та скористатися контекстним меню редактора коду (рисунок 2.4).

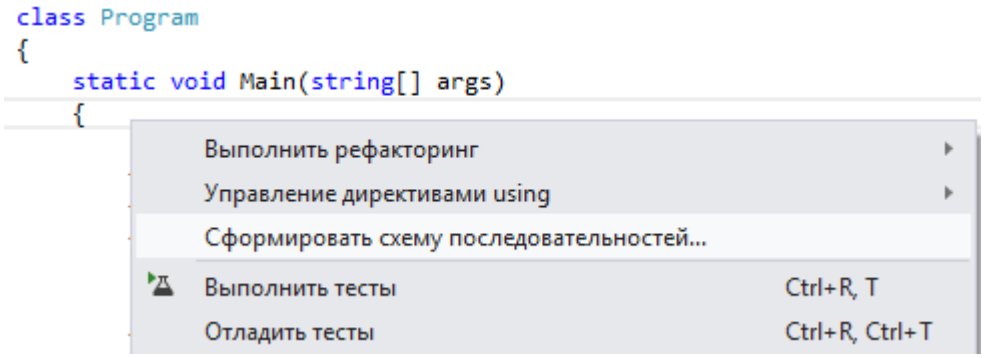

Рисунок 2.4 – Формування діаграми послідовностей UML

У випадку багатопоточних додатків та складній програмній архітектурі такі діаграми можуть бути дуже корисними при

документуванні та презентації розроблених програмістом бібліотек (рисунок 2.5).

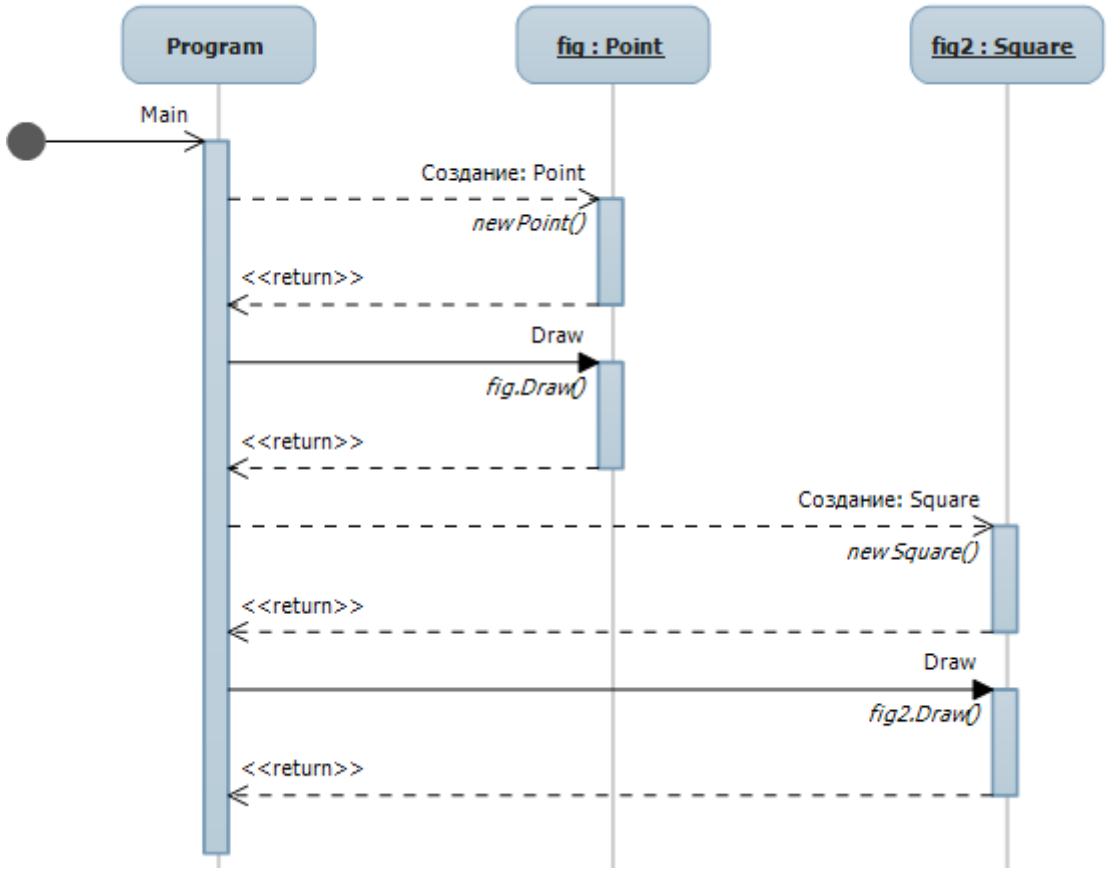

Рисунок 2.5 – Діаграми послідовностей UML для методу `Main`

Завдання № 2

Реалізувати клас, який інкапсулює властивості матриці розміром 3×3 елемента цілого типу. Реалізувати методи випадкового заповнення, обнулення, транспонування та виведення на консоль. Створити нащадка від базового класу, додавши можливість отримання максимального та мінімального значення в матриці. Дослідити послідовність виклику конструкторів при створенні ієрархії класів.

```
using System;

namespace Lab4_3
{
    class Matrix3i // базовий клас матриці
    {
```

```
    public int[,] m = new int[3, 3];
    // формування випадкової матриці, кожний елемент якої
    // знаходиться умежах від min до max
    public void Random(int min, int max)
    {
        Random rnd = new Random();
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                m[i, j] = rnd.Next(min, max);
    }
    public void Print() // друк на консоль
    {
        Console.WriteLine("\nm = ");
        Console.WriteLine("{0}\t{1}\t{2}", m[0, 0], m[0,1],
        m[0, 2]);
        Console.WriteLine("{0}\t{1}\t{2}", m[1, 0], m[1,1],
        m[1, 2]);
        Console.WriteLine("{0}\t{1}\t{2}", m[2, 0], m[2,1],
        m[2, 2]);
    }
    public void Transpose() // транспонування матриці
    {
        int tmp;
        tmp = m[0, 1]; m[0, 1] = m[1, 0]; m[1, 0] = tmp;
        tmp = m[0, 2]; m[0, 2] = m[2, 0]; m[2, 0] = tmp;
        tmp = m[1, 2]; m[1, 2] = m[2, 1]; m[2, 1] = tmp;
    }
    public void SetZero() // заповнення нулями
    {
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                m[i, j] = 0;
    }
    public Matrix3i() // конструктор
    {
        Console.WriteLine("Create Matrix3i");
    }
}
```

```
class Matrix3iEx : Matrix3i // похідний клас
{
    public int MaxValue // властивість тільки для читання
    {
        get
        {
            int max = m[0, 0];
            for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++)
                    if (m[i, j] > max)
                        max = m[i, j];
            return max;
        }
    }
    public int MinValue // властивість тільки для читання
    {
        get
        {
            int min = m[0, 0];
            for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++)
                    if (m[i, j] < min)
                        min = m[i, j];
            return min;
        }
    }
    public Matrix3iEx() // контсруктор
    {
        Console.WriteLine("Create Matrix3iEx");
    }
}
class Program
{
    static void Main(string[] args)
    {
        Matrix3iEx m = new Matrix3iEx();
        m.Print();
        m.Random(-10, 10);
        m.Print();
```

```
            m.Transpose();
            m.Print();
            Console.WriteLine();
            Console.WriteLine("Max = {0}", m.MaxValue);
            Console.WriteLine("Min = {0}", m.MinValue);
            Console.ReadKey();
        }
    }
}
```

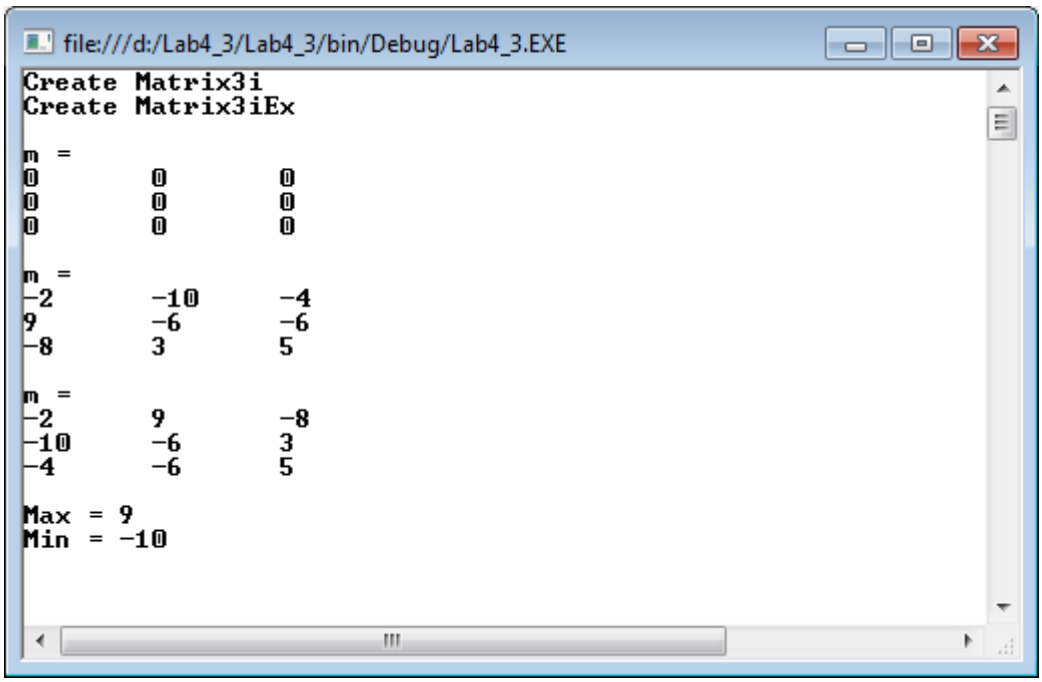

Рисунок 2.6 – Демонстрація роботи програми

Завдання до лабораторної роботи

Відповідно до варіанта та обраного рівня складності виконати наступні завдання.

Рівень завдань	Завдання
Початковий	Відкомпілювати та протестувати аудиторні завдання.
Базовий	До кожного з класів, які наведено нижче, додати по одному полю та методу, побудувати ієрархію класів та реалізувати механізм наслідування.
Високий	До кожного з класів, які наведено нижче, додати по три поля, два метода та по одному конструктору з використання

Рівень завдань	Завдання
	модифікаторів доступу `private, public, protected`. Побудувати ієрархію класів, а базовий клас при цьому об’явити абстрактним, реалізувати механізми наслідування та поліморфізму. У якості поліморфізму виконати перевантаження одного з методів базового класу. Побудувати UML-діаграму класів та діаграму послідовностей засобами Microsoft Visual Studio.

№ варіанта	Завдання
1	Студент, викладач, персона, завідувач кафедри.
2	Службовець, персона, робочий, інженер.
3	Робітник, кадри, інженер, адміністрація.
4	Деталь, механізм, виріб, вузол.
5	Організація, страхова компанія, нафтогазова компанія, завод.
6	Журнал, книга, друковане видання, підручник.
7	Тест, іспит, випускний іспит, випробування.
8	Місце, область, місто, мегаполіс.
9	Іграшка, продукт, товар, молочний продукт.
10	Квитанція, накладна, документ, рахунок.
11	Речовина, мідь, соляна кислота, хлорид натрію, уран.
12	Ртуть, вода, рідина, етиловий спирт.
13	Резистор, світлодіод, елемент, трансформатор.
14	Джойстик, мишка, пристрій введення, клавіатура.
15	Лінія, точка, геометрична фігура, трикутник.
16	Ноутбук, комп’ютер, планшет, нетбук.
17	Гітара, рояль, сопілка, музичний інструмент.
18	Ксерокс, принтер, периферійний пристрій, сканер
19	Маршрутизатор, комутатор, концентратор, мережевий пристрій
20	Автомобіль, поїзд, транспортний засіб, експрес

№ варіанта	Завдання
21	Двигун, двигун внутрішнього згоряння, дизель, реактивний двигун
22	Республіка, монархія, королівство, держава
23	Ссавець, парнокопитне, птиця, тварина
24	Товар, велосипед, гірський велосипед, самокат
25	Лев, дельфін, птиця, синиця, тварина
26	Тест, іспит, випускний іспит, випробування
27	Місце, область, місто, мегаполіс
28	Іграшка, продукт, товар, молочний продукт.
30	Робітник, кадри, інженер, адміністрація

Контрольні питання

1. Що таке клас у розумінні мови C#?
2. Які основні принципи об’єктно-орієнтованого програмування вам відомі?
3. У чому полягає сутність абстрактних методів?
4. Коротко охарактеризувати модифікатори доступу `private, public, protected.`
5. У чому сутність інкапсуляції та поліморфізму?
6. Чим відрізняються змінні за посиланням від змінних за значенням?
7. Які існують правила для конструкторів класів?
8. Яким чином програміст повинен видаляти об’єкти із пам’яті після їх використання?
9. Як відкомпілювати файл програми на мові C# у командному рядку?
10. Для чого призначена уніфікована мова моделювання UML?

ЛАБОРАТОРНА РОБОТА № 3

Тема: базові колекції даних даних .NET Framework на мові C#.

Мета: розробити програми з використанням різних структур даних (масивів, списків, словників, стеків, черг).

Теоретичні відомості

Масив (Array)

Можливо самим простий і відомою структурою даних є `Array`. У C# `Array` є простим масивом об'єктів [8]. Для нього характерно те, що всі його об'єкти одного типу і їх кількість фіксована. Він оголошується і ініціалізується наступним чином:

```
int[] myIntArray = new int[5];
int[] myIntArray2 = { 0, 1, 2, 3, 4 };
```

Як можна бачити з наведеного вище прикладу, масив може ініціалізуватися порожнім або з уже заданими елементами.

Клас `Array` є базовим для мови C#. Тим не менш, явно успадковувати від класу `Array` може тільки система і компілятори. Користувачі повинні застосовувати конструкції масивів, що надаються мовою програмування.

Елементом називається значення, що міститься в об'єкті `Array`. Довжина об'єкта `Array` дорівнює загальному числу елементів, які можуть у ньому міститися. Ранг об'єкта `Array` дорівнює розмірності об'єкта `Array`. Нижня межа виміру об'єкта `Array` є початковим індексом цього виміру об'єкта `Array`; в багатовимірному масиві `Array` у кожного вимірювання можуть бути свої нижні межі. Масив може мати максимум 32 розмірності.

Типовий максимальний розмір `Array` 2 GB. У 64-розрядному середовищі можна уникнути обмеження розміру, встановивши атрибут `enabled` елемента конфігурації `gcAllowVeryLargeObjects` середовища виконання.

Об’єкти Type надають відомості про типи масивів. Об’єкти Array з тим же типом масиву, спільно використовують один і той же об’єкт Type.

Властивість Type.IsArray і метод Type.GetElementType можуть не повертати очікувані результати в об’єкті Array, оскільки у разі приведення масиву до типу Array, результатом є об’єкт, а не масив. Це означає, що typeof(System.Array).IsArray повертає значення false, а typeof(System.Array).GetElementType повертає значення null.

Метод Array.Copy копіює елементи з масиву в масив, не тільки коли типи цих масивів збігаються, але і коли вони різні; приведення типів при цьому виконується автоматично.

Деякі методи, такі як CreateInstance, Copy, CopyTo, GetValue і SetValue, надають перевантаження, які приймають в якості параметрів 64-бітові цілі числа, щоб вмістити масиви великої ємності. LongLength і GetLongLength повертають 64-розрядні цілі числа, які вказують довжину масиву.

В якості введення, давайте подивимося на програму, яка створює і ініціалізує цілий масив з трьох елементів. Будь ласка, зверніть увагу, як елементи задаються та зчитуються. Індексація масиву починається з нуля.

```
using System;

class Program
{
    static void Main()
    {
        // Use an array.
        int[] values = new int[3];
        values[0] = 5;
        values[1] = values[0] * 2;
        values[2] = values[1] * 2;

        foreach (int value in values)
        {
            Console.WriteLine(value);
        }
```

```
    }
}
```

Консольний результат:

```
5
10
20
```

Існує ще один спосіб виділити масив і заповнити його значеннями. Ви можете, використовуючи фігурні дужки {}, присвоїти значення елементів в одному рядку. Довжина масиву визначається автоматично при компіляції програми.

```
using System;

class Program
{
    static void Main()
    {
        // Create an array of three ints.
        int[] array = { 10, 30, 50 };

        foreach (int value in array)
        {
            Console.WriteLine(value);
        }
    }
}
```

Результат роботи:

```
10
30
50
```

Масиви можуть містити не тільки типи значень, такі як цілі. Вони можуть бути використані для рядкових та інших об’єктів. Цікавим моментом тут є те, що самі рядки не зберігаються в масиві. У масиві містяться тільки посилання на рядки. Рядки зазвичай зберігаються в керованій купі.

```
using System;

class Program
{
    static void Main()
    {
        // Create string array of four references.
        string[] array = new string[4];
        array[0] = "DOT";
        array[1] = "NET";
        array[2] = "PERLS";
        array[3] = 2010.ToString();

        // Display.
        foreach (string value in array)
        {
            Console.WriteLine(value);
        }
    }
}
```

Результат роботи програми:

```
DOT
NET
PERLS
2010
```

Як можна передати масив в інший метод, щоб використовувати його там? Нижче подано приклад програми, яка показує правильний синтаксис для передачі цілочисельного масиву. Зверніть увагу, що весь вміст масиву не копіюються, а копіюється лише посилання на масив.

```
using System;

class Program
{
    static void Main()
    {
        // Three-element array.
        int[] array = { -5, -6, -7 };
```

```
        // Pass array to Method.
        Console.WriteLine(Method(array));
    }

    /// <summary>
    /// Receive array parameter.
    /// </summary>
    static int Method(int[] array)
    {
        return array[0] * 2;
    }
}
```

Виведення програми:

```
-10
```

Ви можете повернути масив з методу у якості результату. У цьому прикладі виділяється масив з двох елементів, який потім повертається методом Method.

```
using System;

class Program
{
    static void Main()
    {
        // Write array from Method.
        Console.WriteLine(string.Join(" ", Method()));
    }

    /// <summary>
    /// Return an array.
    /// </summary>
    static string[] Method()
    {
        string[] array = new string[2];
        array[0] = "THANK";
        array[1] = "YOU";
        return array;
    }
```

```
}
```

Перший елемент має індекс 0. Під час доступу до елемента масиву доцільно перевірити, чи не є масив порожнім. У випадку доступу до елементу масиву, який не існує, генерується виняток `IndexOutOfRangeException`.

```
using System;

class Program
{
    static void Main()
    {
        int[] array = new int[2]; // Create an array.
        array[0] = 10;
        array[1] = 20;

        Test(array);
        Test(null); // No output
        Test(new int[0]); // No output
    }

    static void Test(int[] array)
    {
        if (array != null &&
            array.Length > 0)
        {
            int first = array[0];
            Console.WriteLine(first);
        }
    }
}
```

Результат роботи програми

```
10
```

Індекс останнього елемента дорівнює довжині масиву мінус один. Наступна програма демонструє доступ до останнього елемента масиву.

```
using System;
class Program
{
    static void Main()
    {
        string[] arr = new string[]
      {
          "cat",
          "dog",
          "panther",
          "tiger"
      };
        // Get the last string element.
        Console.WriteLine(arr[arr.Length - 1]);
    }
}
```

Результат роботи програми:

```
tiger
```

Список (List)

Масиви не дозволяють динамічно змінювати розмір. Це дозволяє зробити тип `List<>`. Працюючи зі списком, Ви можете змінювати його місткість на власний розсуд. Цей тип ідеально підходить для лінійної колекції. Він надає безліч методів і властивостей. Списки є узагальненнями `generic`. Ви повинні використовувати синтаксис оголошення списку використовуючи тип елементів списку.

Для початку розглянемо, як оголосити новий список цілочисельних значень і додати декілька чисел до нього. Кутові дужки є частиною декларації узагальненого типу і не мають відношення до умовних операторів (менше або більше).

```
using System.Collections.Generic;

class Program
{
    static void Main()
```

```
    {
        List<int> list = new List<int>();
        list.Add(2);
        list.Add(3);
        list.Add(5);
        list.Add(7);
    }
}
```

Ви можете перебрати всі елементи списку за допомогою структури `foreach`. Це звичайна операція при використанні `List`. Синтаксис такий же, як і для масиву.

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        List<int> list = new List<int>();
        list.Add(2);
        list.Add(3);
        list.Add(7);

        // Loop through List with foreach
        foreach (int prime in list)
        {
            Console.WriteLine(prime);
        }

        // Loop through List with for
        for (int i = 0; i < list.Count; i++)
        {
            Console.WriteLine(list[i]);
        }
    }
}
```

Виведення програми:

```
2
3
7
2
3
7
```

Наступна програма демонструє метод очищення списку.

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        List<bool> list = new List<bool>();
        list.Add(true);
        list.Add(false);
        list.Add(true);
        Console.WriteLine(list.Count); // 3

        list.Clear();
        Console.WriteLine(list.Count); // 0
    }
}
```

Вивід програми:

```
3
0
```

Список можна ініціалізувати існуючим масивом.

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
```

```
            int[] arr = new int[3]; //New array with 3 elements
            arr[0] = 2;
            arr[1] = 3;
            arr[2] = 5;
            List<int> list = new List<int>(arr); //Copy to List
            Console.WriteLine(list.Count); //3 elements in List
        }
    }
```

У список можна додати елементи в будь-яку позицію.

```
    using System;
    using System.Collections.Generic;

    class Program
    {
        static void Main()
        {
            List<string> dogs = new List<string>();

            dogs.Add("spaniel");     // Contains: spaniel
            dogs.Add("beagle");     // Contains: spaniel, beagle
            dogs.Insert(1, "dalmatian"); // Contains: spaniel,
dalmatian, beagle
            foreach (string dog in dogs) // Display
            {
                Console.WriteLine(dog);
            }
        }
    }
```

Вивід програми:

```
    spaniel
    dalmatian
    beagle
```

У списків є можливість інвертувати послідовність елементів.

```
    using System;
    using System.Collections.Generic;
    class Program
```

```
{
    static void Main()
    {
        List<string> list = new List<string>();
        list.Add("anchovy");
        list.Add("barracuda");
        list.Add("bass");
        list.Add("viperfish");

        // Reverse List in-place, no new variables required
        list.Reverse();

        foreach (string value in list)
        {
            Console.WriteLine(value);
        }
    }
}
```

Результат роботи програми:

```
viperfish
bass
barracuda
anchovy
```

Словник (Dictionary)

Якщо потрібно здійснювати швидкий пошук елементів у колекції, то доцільно застосувати словник. Словник забезпечує швидкий пошук з ключами для отримання значень. З його допомогою ми використовуємо ключі і значення будь-якого типу, в тому числі цілі і рядки. Словник вимагає особливої форми синтаксису.

```
using System;
using System.Collections.Generic;

class Program
{
```

```
        static void Main()
        {
            Dictionary<string, int> dictionary =
                new Dictionary<string, int>();
            dictionary.Add("cat", 2);
            dictionary.Add("dog", 1);
            dictionary.Add("llama", 0);
            dictionary.Add("iguana", -1);
        }
    }
```

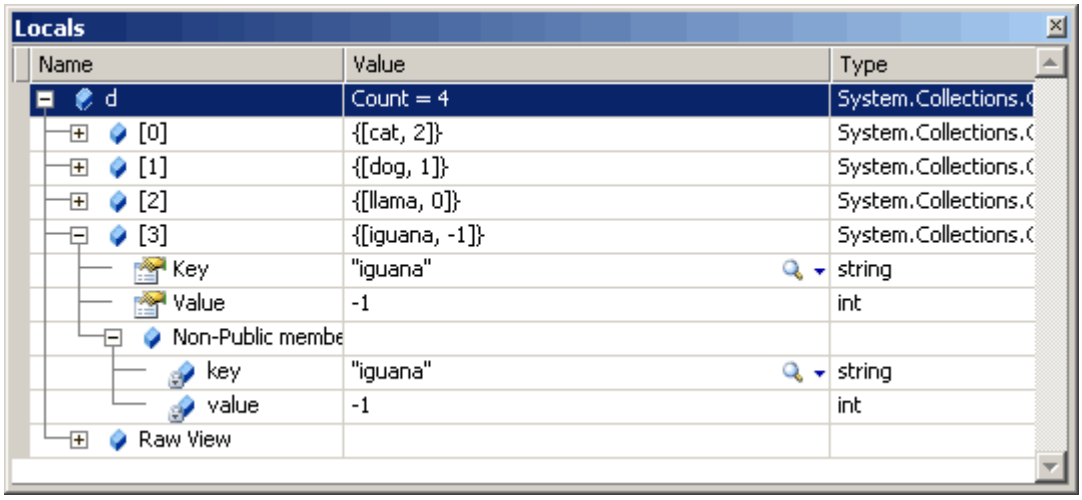

Рисунок 3.1 – Перегляд словника у відладчику Visual Studio

Наступна програма демонструє підхід до перегляду всіх елементів словника.

```
    using System;
    using System.Collections.Generic;

    class Program
    {
        static void Main()
        {
            // Example Dictionary again
            Dictionary<string, int> d = new Dictionary<string,
int>()
            {
                {"cat", 2},
                {"dog", 1},
```

```
            {"llama", 0},
            {"iguana", -1}
        };
        // Loop over pairs with foreach
        foreach (KeyValuePair<string, int> pair in d)
        {
            Console.WriteLine("{0}, {1}",
            pair.Key,
            pair.Value);
        }

        // Use var keyword to enumerate dictionary
        foreach (var pair in d)
        {
            Console.WriteLine("{0}, {1}",
            pair.Key,
            pair.Value);
        }
    }
}
```

Результат роботи програми:

```
cat, 2
dog, 1
llama, 0
iguana, -1

cat, 2
dog, 1
llama, 0
iguana, -1
```

Стек (Stack)

Доступ до елементів *стека* здійснюється за принципом «останнім увійшов – першим вийшов» (LIFO – Las Input First Output).

Стек реалізований у класі `Stack`. Клас налічує такі конструктори:

```
Stack ([capacity]);
```

```
Stack (collection);
```

За замовчуванням місткість стеку `capacity` дорівнює 10. Друга версія конструктора для утворення стека використовує колекцію.

Елементи стека можуть мати довільний тип.

Властивість `Count` повертає кількість елементів у стеку.

Із методів класу Stack зазначимо такі:

Метод	Зміст
`Clear`	вилучає всі елементи зі стека
`Contains`	визначає, чи містить стек заданий елемент
`CopyTo`	копіює елементи стека в одновимірний масив
`Peek`	повертає елемент на вершині стека, проте не вилучає його зі стека
`Pop`	повертає елемент на вершині стека та вилучає його зі стека
`Push`	додає елемент на вершину стека
`ToArray`	копіює елементи стека в масив

Приклад роботи зі стеком подано нижче:

```
using System;
using System.Collections.Generic;

class Program
{
    static Stack<int> GetStack()
    {
        Stack<int> stack = new Stack<int>();
        stack.Push(100);
        stack.Push(1000);
        stack.Push(10000);
        return stack;
    }

    static void Main()
    {
        var stack = GetStack();
        Console.WriteLine("--- Stack contents ---");
```

```
            foreach (int i in stack)
            {
                Console.WriteLine(i);
            }
        }
    }
```

Результат роботи програми:

```
--- Stack contents ---
10000
1000
100
```

Наступний фрагмент коду показує пошук певного значення в стеку:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // An example string array.
        string[] values = { "Dot", "Net", "Perls" };

        // Copy an array into a Stack.
        var stack = new Stack<string>(values);

        // Display the Stack.
        Console.WriteLine("Stack contents");
        foreach (string value in stack)
        {
            Console.WriteLine(value);
        }

        // See if the stack contains "Perls"
        Console.WriteLine("Stack Contains method result");
        bool contains = stack.Contains("Perls");
        Console.WriteLine(contains);
```

```
    }
}
```

Результат:

```
Stack contents
Perls
Net
Dot
Stack Contains method result
True
```

Черга (Queue)

Доступ до елементів черги здійснюється за принципом «першим увійшов – першим вийшов» (FIFO – First Input First Output).

Черга реалізована в класі `Queue`. Клас містить такі конструктори:

```
Queue ([capacity [, growFactor]]);
Queue (collection);
```

За замовчуванням місткість черги `capacity` дорівнює 32, а фактор збільшення `growFactor` – 2,0. Фактор збільшення задає множник, на який потрібно помножити місткість, якщо існуючої недостатньо. Друга версія конструктора використовує для утворення черги колекцію – екземпляр класу з підтримкою інтерфейсу `ICollection`.

Елементи черги можуть мати довільний тип.

Властивість `Count` повертає кількість елементів у черзі.

Із методів класу `Queue` зазначимо такі:

Метод	Пояснення
`Clear`	вилучає всі елементи з черги
`Contains`	визначає, чи містить черга заданий елемент
`CopyTo`	копіює елементи черги в одновимірний масив
`Dequeue`	повертає значення першого елемента черги та вилучає його з черги
`Enqueue`	додає елемент наприкінці черги

Метод	Пояснення
Peek	повертає значення першого елемента черги, однак не вилучає його з черги
ToArray	копіює елементи черги в масив

У наступному прикладі демонструється використання методу Enqueue:

```
using System;
using System.Collections.Generic;
class Program
{
    static void Main()
    {
        // New Queue of integers
        Queue<int> q = new Queue<int>();
        q.Enqueue(5);   // Add 5 to the end of the Queue.
        q.Enqueue(10);  // Then add 10. 5 is at the start.
        q.Enqueue(15);  // Then add 15.
        q.Enqueue(20);  // Then add 20.
    }
}
```

Для того, щоб переглянути всі елементи даної колекції можна скористатися інструкцією foreach:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // Add integers to queue.
        Queue<int> collection = new Queue<int>();
        collection.Enqueue(5);
        collection.Enqueue(6);

        // Loop through queue.
        foreach (int value in collection)
        {
```

```
            Console.WriteLine(value);
        }
    }
}
```

Консольний результат:

```
5
6
```

Елементи черги можна копіювати до масиву:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        // New Queue of integers.
        Queue<int> queue = new Queue<int>();
        queue.Enqueue(5);
        queue.Enqueue(10);
        queue.Enqueue(15);
        queue.Enqueue(20);

        // Create new array with Length equal to Queue's
element count.
        int[] array = new int[queue.Count];
        // Copy the Queue to the int array.
        queue.CopyTo(array, 0);
        // Loop through and display int[] in order.
        Console.WriteLine("Array:");
        for (int i = 0; i < array.Length; i++)
        {
            Console.WriteLine(array[i]);
        }
        // Loop through int array in reverse order.
        Console.WriteLine("Array reverse order:");
        for (int i = array.Length - 1; i >= 0; i--)
        {
```

```
            Console.WriteLine(array[i]);
        }
    }
}
```

Аудиторні завдання

Завдання № 1

Дано список цілих чисел, який складається з n елементів. Обчислити суму та кількість чисел, що діляться без остачі на 10. Якщо таких чисел немає, вивести повідомлення «Чисел, що діляться без остачі на 10, у списку немає».

Розв’язання

Один із варіантів вирішення даного завдання представлено нижче:

```
using System;
using System.Collections.Generic;
namespace Collection
{
    class Program
    {
        const int n = 50;

        static void Main(string[] args)
        {
            List<int> list = new List<int>();
            Random rnd = new Random();
            for (int i = 0; i < n; i++)
            {
                list.Add(rnd.Next(100));
            }
            int sum = 0;
            int count = 0;
            Console.Write("List: ");
            foreach (int a in list)
            {
                Console.Write("{0}; ", a);
```

```
                    if (a % 10 == 0)
                    {
                        count++;
                        sum += a;
                    }
                }
                Console.WriteLine();
                if (count == 0)
                {
                    Console.WriteLine("Чисел, що діляться без
остачі на 10, у списку немає");
                }
                else
                {
                    Console.WriteLine("Count:  {0}\nSum:  {1}",
count, sum);
                }
                Console.ReadKey();
            }
        }
    }
```

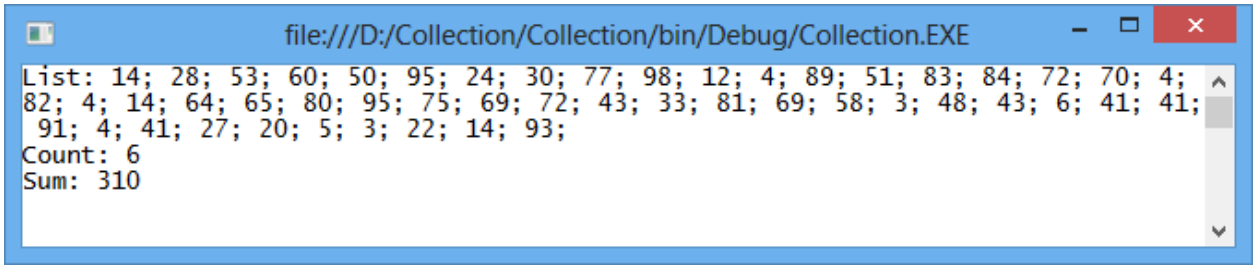

Рисунок 3.2 – Результат роботи програми

Завдання до лабораторної роботи

Відповідно до варіанта та обраного рівня складності виконати наступні завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Завдання № 1	Завдання № 1
2	–	–	Завдання № 2

Завдання № 1

Дано структуру даних (колекцію) відповідно до варіанта. Додати зазначену кількість елементів, які описують відповідну предметну область. Вивести всі елементи на консоль в прямому та зворотному порядку. Вивести кількість елементів у колекції. Очистити колекцію.

№ варіанта	Завдання		
	Колекція	Елементи	Кількість елементів
1	Array	Назви міст України	10
2	List	Назви країн Європи	6
3	SortedList	Назви виробників комп’ютерної техніки	8
4	Dictionary	Назви металів	4
5	ArrayList	Назви хімічних елементів	9
6	Hashtable	Назви нот	7
7	Stack	Назви операційних систем	4
8	Queue	Назви місяців року	5
9	Array	Назви знаків зодіаку	12
10	List	Назви мікропроцесорів	11
11	SortedList	Назви типів даних мови C++	7
12	Dictionary	Назви областей України	6
13	ArrayList	Назви мов програмування	4
14	Hashtable	Назви колекцій .NET Framework	9
15	Stack	Назви типів даних мови C#	10
16	Queue	Назви штатів США	5
17	Array	Назви типів даних мови Pascal	6
18	List	Назви хімічних елементів	8
19	SortedList	Назви нот	4

№ варіанта	Завдання		
	Колекція	Елементи	Кількість елементів
20	Dictionary	Назви операційних систем	9
21	ArrayList	Назви місяців року	7
22	Hashtable	Назви знаків зодіаку	4
23	Stack	Назви мікропроцесорів	5
24	Queue	Назви типів даних мови C++	12
25	Array	Назви областей України	11
26	List	Назви мов програмування	7
27	SortedList	Назви колекцій .NET Framework	6
28	Dictionary	Назви типів даних мови C#	4
30	ArrayList	Назви дисциплін Вашого курсу	9

Завдання № 2

№ варіанта	Завдання
1	Дано стек цілих чисел, який складається з n елементів. Обчислити суму та кількість чисел, що діляться без остачі на 5. Якщо таких чисел немає, вивести повідомлення «Чисел, що діляться без остачі на 5, в стеку немає».
2	Дано список цілих чисел, який складається з n елементів. Обчислити добуток та кількість від’ємних непарних чисел, розташованих на парних позиціях.
3	Дано чергу цілих чисел, яка складається з n елементів. Визначити різницю між максимальним та мінімальним елементами.
4	Дано стек цілих чисел, який складається з n елементів. Обчислити добуток максимального і мінімального елементів стеку.
5	Дано список цілих чисел, який складається з n елементів. Визначити суму додатних елементів списку, що стоять на позиціях, кратних трьом.
6	Дано чергу цілих чисел, яка складається з n елементів. Визначити середнє арифметичне значення чисел черги, що

№ варіанта	Завдання
	стоять на парних позиціях, і середнє арифметичне значення чисел, що стоять на непарних позиціях.
7	Дано стек цілих чисел, який складається з n елементів. Визначити суму додатних парних чисел стеку. У разі відсутності додатних парних чисел вивести повідомлення «Додатних парних чисел в стеку немає».
8	Дано список цілих чисел, який складається з n елементів. Визначити окремо кількість від'ємних чисел, кількість додатних чисел і чисел, рівних нулю.
9	Дано чергу цілих чисел, яка складається з n елементів. Визначити середнє арифметичне значення додатних елементів і середнє арифметичне значення від'ємних елементів.
10	Дано стек цілих чисел, який складається з n елементів. Визначити суму і добуток додатних чисел, що стоять на парних позиціях.
11	Дано список цілих чисел, який складається з n елементів. Визначити середнє арифметичне значення мінімального і максимального елементів списку.
12	Дано чергу цілих чисел, яка складається з n елементів. Обчислити кількість елементів черги, кратних семи. За відсутності таких елементів вивести повідомлення «Елементів кратних 7 немає».
13	Дано стек цілих чисел, який складається з n елементів. Визначити добуток додатних елементів стеку та їх кількість. За відсутності додатних чисел вивести повідомлення «Додатних чисел в стеку немає».
14	Дано список цілих чисел, який складається з n елементів. У ньому серед додатних елементів визначити максимальний елемент.
15	Дано чергу цілих чисел, яка складається з n елементів. Визначити середнє арифметичне значення елементів черги, кратних восьми. Якщо таких елементів немає, вивести повідомлення: «Елементів кратних 8 в черзі немає».
16	Дано стек цілих чисел, який складається з n елементів. Визначити добуток непарних елементів стеку, що стоять на парних позиціях. Якщо таких елементів немає, вивести

№ варіанта	Завдання
	повідомлення: «Непарних елементів, що стоять на парних позиціях в стеку немає».
17	Дано список цілих чисел, який складається з n елементів. У ньому визначити мінімальний елемент та його порядковий номер.
18	Дано чергу цілих чисел, яка складається з n елементів. Визначити середнє арифметичне значення елементів черги, які містяться в діапазоні [-3; 5]. Якщо таких елементів немає, вивести повідомлення: «Елементів, що задовольняють вимозі, в черзі немає».
19	Дано стек цілих чисел, який складається з n елементів. У ньому визначити середнє арифметичне значення чисел, кратних трьом. Якщо таких елементів немає, вивести повідомлення: «Елементів кратних 3 в стеку немає».
20	Дано список цілих чисел, який складається з n елементів. Визначити кількість і добуток елементів списку, які знаходяться в діапазоні [0; 7]. Якщо таких елементів немає, вивести повідомлення: «Елементів з діапазону [0; 7] в списку немає».
21	Дано чергу цілих чисел, яка складається з n елементів. Визначити у ній кількість чисел, кратних 2, і чисел, не кратних 3. Якщо таких елементів немає, вивести повідомлення: «Чисел, кратних 2 і чисел не кратних 3 в черзі немає».
22	Дано стек цілих чисел, який складається з n елементів. Визначити в ньому суму і кількість чисел, кратних 5. Якщо таких елементів немає, вивести повідомлення: «Елементів кратних 5 в стеку немає».
23	Дано список цілих чисел, який складається з n елементів. Визначити суму і кількість непарних чисел списку. Якщо таких елементів немає, вивести повідомлення: «Непарних елементів в списку немає».
24	Дано чергу цілих чисел, яка складається з n елементів. Визначити добуток від'ємних чисел черги, що стоять на парних позиціях. Якщо таких елементів немає, вивести повідомлення: «Від'ємних елементів в черзі немає».
25	Дано стек цілих чисел, який складається з n елементів. Визначити добуток додатних парних чисел стеку Якщо таких

№ варіанта	Завдання
	елементів немає, вивести повідомлення: «Додатних парних елементів в стеку немає».
26	Дано стек цілих чисел, який складається з n елементів. Визначити в ньому суму і кількість чисел, кратних 7. Якщо таких елементів немає, вивести повідомлення: «Елементів кратних 7 в стеку немає».
27	Дано список цілих чисел, який складається з n елементів. Визначити суму і кількість непарних чисел списку. Якщо таких елементів немає, вивести повідомлення: «Непарних елементів в списку немає».
28	Дано чергу цілих чисел, яка складається з n елементів. Визначити добуток від'ємних чисел черги, що стоять на непарних позиціях. Якщо таких елементів немає, вивести повідомлення: «Від'ємних елементів в черзі немає».
30	Дано список цілих чисел, який складається з n елементів. Визначити кількість і добуток елементів списку, які знаходяться в діапазоні [0; 5]. Якщо таких елементів немає, вивести повідомлення: «Елементів з діапазону [0; 5] в списку немає».

Контрольні питання

1. Що таке динамічні структури даних?
2. Які динамічні структури вам відомі?
3. Розкрийте сутність роботи із стеком.
4. Наведіть приклад початкового формування стеку.
5. Наведіть приклад занесення даних в стек.
6. Наведіть приклад вибірки даних із стеку.
7. Розкрийте сутність роботи з чергою.
8. Наведіть приклад початкового формування черги.
9. Наведіть приклад додавання даних в кінець черги.
10. Наведіть приклад вибірки даних з черги.

ЛАБОРАТОРНА РОБОТА № 4

Тема: робота з файлами у C#.

Мета: читати та записувати інформацію у текстові та двійкові файли.

Теоретичні відомості

Файл – іменована інформація на зовнішньому носієві, наприклад на жорсткому або гнучкому магнітному диску. Логічно файл можна представити як кінцеву кількість послідовних байтів, тому такі пристрої, як дисплей, клавіатура і принтер, також можна розглядати як файли. Передача даних із зовнішнього пристрою в оперативну пам'ять називається *читанням*, або *введенням*, зворотний процес – *записом*, або *виведенням* [8].

Введення-виведення в C# виконується за допомогою підсистеми введення-виведення і класів бібліотеки *.NET*. Обмін даними реалізується за допомогою потоків.

Потік (`stream`) – це абстрактне поняття, що належить до будь-якого перенесення даних від джерела до приймача. Потоки забезпечують надійну роботу як із стандартними, так і з визначеними користувачем типами даних. Потік визначається як послідовність байтів і не залежить від конкретного пристрою, з яким проводиться обмін (оперативна пам'ять, файл на диску, клавіатура або принтер).

Обмін з потоком для підвищення швидкості передачі даних проводиться, як правило, через спеціальну область оперативної пам'яті – *буфер*. Буфер виділяється для кожного відкритого файлу. При записі у файл вся інформація спочатку прямує в буфер і там накопичується до тих пір, поки весь буфер не заповниться. Тільки після цього або після спеціальної команди скидання відбувається передача даних на зовнішній пристрій. При читанні з файлу дані спочатку прочитуються в буфер, причому не стільки, скільки запрошується, а скільки поміщається в буфер.

Механізм буферізації дозволяє швидше і ефективно обмінюватися інформацією із зовнішніми пристроями.

Для підтримки потоків бібліотека *.NET* містить ієрархію класів, основна частина якої представлена на рисунку 4.1. Ці класи визначені в просторі імен `System.IO`. Окрім класів там описана велика кількість перелічень для завдання різних властивостей і режимів.

Класи бібліотеки дозволяють працювати в різних режимах з файлами, каталогами і областями оперативної пам'яті. Короткий опис класів приведений в таблиці 4.1.

Як можна бачити з таблиці, виконувати обмін із зовнішніми пристроями можна на рівні:

– двійкового представлення даних (`BinaryReader`, `BinaryWriter`);

– байтів (`FileStream`);

– тексту, тобто символів (`StreamWriter`, `StreamReader`).

У *.NET* використовується кодування *Unicode*, в якому кожен символ кодується двома байтами. Класи, що працюють з текстом, є оболонками класів, що використовують байти, і автоматично виконують те, що кодується з байтів в символи і назад.

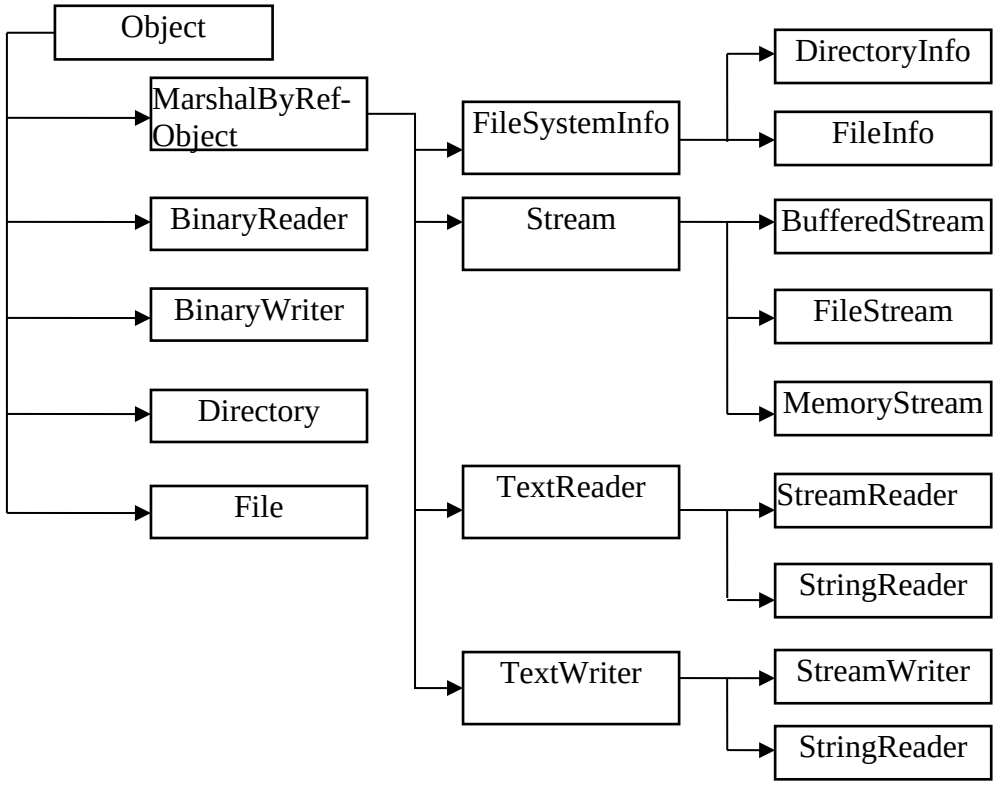

Рисунок 4.1 – Класи бібліотеки *.NET* для роботи з потоками

Двійкові і байтові потоки зберігають дані в тому ж вигляді, в якому вони представлені в оперативній пам'яті, тобто при обміні з файлом відбувається побітове копіювання інформації. Двійкові файли застосовуються не для перегляду їх людиною, а для використання в програмах.

Доступ до файлів може бути *послідовним*, коли черговий елемент можна прочитати (записати) тільки після аналогічної операції з попереднім елементом, і *довільним*, або *прямим*, при якому виконується читання (запис) довільного елементу за заданою адресою. Текстові файли дозволяють виконувати тільки послідовний доступ, в двійкових і байтових потоках можна використовувати обидва методи.

Прямий доступ у поєднанні з відсутністю перетворень забезпечує високу швидкість отримання потрібної інформації.

Методи *форматованого введення*, за допомогою яких можна виконувати введення з клавіатури або з текстового файлу значень арифметичних типів, в C# не підтримуються. Для перетворення з

символьного в числове представлення використовуються методи класу `Convert` або метод `Parse`, розглянуті в попередніх розділах.

Таблиця 4.1 – Основні класи простору імен `System.IO`

Клас	Опис
`BinaryReader`, `BinaryWriter`	Читання і запис значень простих вбудованих типів (цілочисельних, логічних, рядкових і т. п.) у внутрішній формі уявлення
`BufferedStream`	Тимчасове зберігання потоку байтів (наприклад, для подальшого перенесення в постійне сховище)
`Directory`, `Directorylnfo`, `File`, `Filelnfo`	Робота з каталогами або фізичними файлами: створення, видалення, придбання властивостей. Можливості класів `File` і `Directory` реалізовані в основному у вигляді статичних методів. Аналогічні класи `DirectoryInfo` і `FileInfo` використовують звичайні методи
`FileStream`	Довільний (прямий) доступ до файлу, представленого як потік байтів
`MemoryStream`	Довільний доступ до потоку байтів в оперативній пам’яті
`StreamWriter`, `StreamReader`	Читання з файлу і запис у файл текстової інформації (довільний доступ не підтримується)
`StringWriter`, `StringReader`	Робота з текстовою інформацією в оперативній пам'яті

Форматоване виведення, тобто перетворення з внутрішньої форми представлення числа в символьну, зрозумілу людині, виконується за допомогою перевантажених методів `ToString`.

Окрім перерахованих класів в бібліотеці .NET є класи `XmlTextReader` і `XmlTextWriter`, призначені для формування і читання коду у форматі *XML*.

Розглянемо прості способи роботи з файловими потоками. Використання класів файлових потоків в програмі припускає наступні операції:

1) Створення потоку і зв’язування його з фізичним файлом.
2) Обмін (уведення-виведення).
3) Закриття файлу.

Кожен клас файлових потоків містить декілька варіантів конструкторів, за допомогою яких можна створювати об'єкти цих класів різними способами і в різних режимах.

Наприклад, файли можна відкривати тільки для читання, тільки для запису або для читання і запису. Ці режими доступу до файлу містяться в переліченні `FileAccess`, визначеному в просторі імен `System.IO`. Константи перелічення приведені в таблиці 4.2.

Таблиця 4.2 – Значення `FileAccess`

Значення	Опис
`Read`	Відкрити файл тільки для читання
`ReadWrite`	Відкрити файл для читання і запису
`Write`	Відкрити файл тільки для запису

Можливі режими відкриття файлу визначені в переліченні `FileMode` (таблиця 4.3).

Таблиця 4.3 – Значення `FileMode`

Значення	Опис

`Append`	Відкрити файл, якщо він існує, і встановити поточний покажчик в кінець файлу. Якщо файл не існує, створити новий файл
`Create`	Створити новий файл. Якщо в каталозі вже існує файл з таким же ім'ям, він буде стертий
`CreateNew`	Створити новий файл. Якщо в каталозі вже існує файл з таким же ім'ям, виникає виключення `IOException`
`Open`	Відкрити існуючий файл
`OpenOrCreate`	Відкрити файл, якщо він існує. Якщо немає, створити файл з таким ім'ям
`Truncate`	Відкрити існуючий файл. Після відкриття вміст файлу видаляється

Режим `FileMode.Append` можна використовувати тільки спільно з доступом типу `FileAccess.Write`, тобто для файлів, що відкриваються для запису.

Режими сумісного *використання* файлу різними користувачами визначає перелічення `FileShare` (таблиця 4.4).

Таблиця 4.4 – Значення `FileShare`

Значення	Опис
`None`	Сумісне використання відкритого файлу заборонене. Запит на відкриття даного файлу завершується повідомленням про помилку
`Read`	Дозволяє відкривати файл для читання одночасно декільком користувачам. Якщо цей прапор не встановлений, запити на відкриття файлу для читання завершуються повідомленням про помилку
`ReadWrite`	Дозволяє відкривати файл для читання і запису одночасно декільком користувачам
`Write`	Дозволяє відкривати файл для запису одночасно декільком користувачам

Потоки байтів

Уведення-виведення у файл на рівні байтів виконується за допомогою класу `FileStream`, який є спадкоємцем абстрактного класу `Stream`, що визначає набір стандартних операцій з потоками. Елементи класу `Stream` описані в таблиці 4.5.

Таблиця 4.5 – Елементи класу `Stream`

Елемент	Опис
`BeginRead`, `BeginWrite`	Почати асинхронне введення або виведення
`CanRead`, `CanSeek`, `CanWrite`	Властивості, що визначають, які операції підтримує потік: читання, прямий доступ і/або запис
`Close`	Закрити поточний потік і звільнити пов'язані з ним ресурси (сокети, покажчики на файли і т. п.)
`EndRead`, `EndWrite`	Чекати завершення асинхронного введення; закінчити асинхронне виведення
`Flush`	Записати дані з буфера в пов'язане з потоком джерело даних і очистити буфер. Якщо для даного потоку буфер не використовується, то цей метод нічого не робить
`Length`	Повернути довжину потоку в байтах
`Position`	Повернути поточну позицію в потоці
`Read`, `ReadByte`	Підрахувати послідовність байтів (або один байт) з поточного потоку і перемістити покажчик в потоці на кількість лічених байтів
`Seek`	Встановити поточний покажчик потоку на задану позицію
`SetLength`	Встановити довжину поточного потоку
`Write`, `WriteByte`	Записати послідовність байтів (або один байт) в поточний потік і перемістити покажчик в потоці на кількість записаних байтів

Клас `FileStream` реалізує ці елементи для роботи з дисковими файлами. Для визначення режимів роботи з файлом використовуються стандартні перелічення `FileMode`, `FileAccess` і

FileShare. Значення цих перелічень приведені в таблицях 4.2–4.4. У лістингу 4.1 представлений приклад роботи з файлом. У прикладі демонструються читання і запис одного байта і масиву байтів, а також позиціонування в потоці.

Лістинг 4.1 – Приклад використання потоку байтів

```
using System;
using System.IO;

namespace FileTest
{
    class Class1
    {
        static void Main(string[] args)
        {
            FileStream f = new FileStream("test.txt",
FileMode.Create, FileAccess.ReadWrite);
            // у початок файлу записується число 100
            f.WriteByte(100);
            byte[] x = new byte[10];
            // записується 10 чисел від 0 до 9
            for (byte i = 0; i < 10; ++i)
            {
                x[i] = (byte)(10 - i);
                f.WriteByte(i);
            }
            // записується 5 елементів масиву
            f.Write(x, 0, 5);
            byte[] y = new byte[20];
            f.Seek(0, SeekOrigin.Begin); // поточний покажчик -
на початок
            f.Read(y, 0, 20); // читання з файлу в масив
            foreach (byte elem in y)
            {
                Console.Write(" " + elem);
            }
            Console.WriteLine();
```

```
            f.Seek(5, SeekOrigin.Begin); // поточний покажчик -
на 5-й елемент
            int a = f.ReadByte();        // читання 5-го
елементу
            Console.WriteLine(a);
            a = f.ReadByte();            // читання 6-го
елементу
            Console.WriteLine(" Поточна позиція в потоці " +
f.Position);
            f.Close();
        }
    }
}
```

Результат роботи програми:

```
100   01234   56789   10 98760000
4
5
Поточна позиція в потоці 7
```

Поточна позиція в потоці спочатку встановлюється на початок файлу (для будь-якого режиму відкриття, окрім `Append`) і зрушується на одну позицію при записі кожного байта.

Для установки бажаної позиції читання використовується метод `Seek`, що має два параметри: перший задає зсув в байтах щодо точки відліку, що задається другим. Точки відліку задаються константами перелічення `SeekOrigin`: початок файлу – `Begin`, поточна позиція – `Current` і кінець файлу – `End`.

У даному прикладі файл створювався в поточному каталозі. Можна вказати і повний шлях до файлу, наприклад:

```
FileStream f = new FileStream( @"D:\C#\test.txt",
FileMode.Create, FileAccess.ReadWrite );
```

Операції по відкриттю файлів можуть завершитися невдало, наприклад, при помилці в імені існуючого файлу або за відсутності

вільного місця на диску, тому рекомендується завжди контролювати результати цих операцій.

У разі непередбачених ситуацій середовище виконання генерує різні виключення, обробку яких слід передбачити в програмі, наприклад:

– `FileNotFoundException`, якщо файлу у вказаному каталозі не існує;

– `DirectoryNotFoundException`, якщо не існує вказаний каталог;

– `ArgumentException`, якщо невірно заданий режим відкриття файлу;

– `IOException`, якщо файл не відкривається із-за помилок введення-виведення.

Можливі і інші виняткові ситуації. Зручно обробляти найбільш вірогідні помилки роздільно, щоб надати користувачеві програми в повідомленні, що виводиться, найбільш точну інформацію. У приведеному далі прикладі окремо перехоплюється помилка в імені файлу, а потім обробляється решта всіх можливих помилок:

```
try
{
    FileStream f = new FileStream(@"d:\test.tx",
      FileMode.Open, FileAccess.Read );
    // дії з файлом
    f.Close();
}
catch( FileNotFoundException e)
{
    Console.WriteLine( e.Message ) ;
    Console.WriteLine( "Перевірте правильність імені
файлу!" );
    return;
}
catch( Exception e )
{
    Console.WriteLine( "Error:  " + e.Message );
    return;
```

```
}
```

При закритті файлу звільняються всі пов'язані з ним ресурси, наприклад, для файлу, відкритого для запису, у файл вивантажується вміст буфера. Тому рекомендується завжди закривати файли після закінчення роботи, особливо файли, відкриті для запису. Якщо буфер потрібно вивантажити, не закриваючи файл, використовується метод `Flush`.

Потоки символів

Символьні потоки `StreamWriter` і `StreamReader` працюють з *Unicode-символами.* Ці потоки є спадкоємцями класів `TextWriter` і `TextReader`. У таблицях 4.6 і 4.7 приведені найбільш важливі елементи цих класів. Як бачите, довільний доступ для текстових файлів не підтримується.

Таблиця 4.6 – Найбільш важливі елементи базового класу `TextWriter`

Елемент	Опис
`Close`	Закрити файл і звільнити пов'язані з ним ресурси. Якщо в процесі запису використовується буфер, він буде автоматично очищений
`Flush`	Очистити всі буфери для поточного файлу і записати накопичені в них дані в місце їх постійного зберігання. Сам файл при цьому не закривається
`NewLine`	Використовується для завдання послідовності символів, що означають початок нового рядка. За умовчанням використовується послідовність «повернення каретки» – «переведення рядка» (\r \n)
`Write`	Записати фрагмент тексту в потік
`WriteLine`	Записати рядок в потік і перейти на інший рядок

Таблиця 4.7 – Найбільш важливі елементи класу `TextReader`

Елемент	Опис
`Peek`	Повернути наступний символ, не змінюючи позицію покажчика у файлі
`Read`	Рахувати дані з вхідного потоку
`ReadBlock`	Рахувати з вхідного потоку вказану користувачем кількість символів і записати їх в буфер, починаючи із заданої позиції
`ReadLine`	Рахувати рядок з поточного потоку і повернути його як значення типу *string*. Порожній рядок (null) означає кінець файлу (EOF)
`ReadToEnd`	Рахувати всі символи до кінця потоку, починаючи з поточної позиції, і повернути дані як один рядок типу *string*

Ви вже знайомі з деякими методами, приведеними в цих таблицях: впродовж всього посібника постійно використовувалися методи читання з текстових потоків і запису в текстові потоки, але не для дискових файлів, а для консолі, яка є їх окремим випадком.

У лістингу 4.3 створюється текстовий файл, в який записуються два рядки. Другий рядок формується з перетворених чисельних значень змінних і пояснюючого тексту. Вміст файлу можна подивитися в будь-якому текстовому редакторові. Файл створюється в тому ж каталозі, куди середовище записує виконуваний файл. За умовчанням це каталог *...\ConsoleApplication1\bin\Debug*.

Лістинг 4.3 – Виведення в текстовий файл

```
using System;
using System.IO;

namespace FileTest
{
    class Program
    {
        static void Main(string[] args)
        {
            try
```

```
            {
                StreamWriter f = new StreamWriter("text.txt");
                f.WriteLine("ABCDEF:");
                double a = 12.234;
                int b = 29;
                f.WriteLine(" a = {0,6:C} b = {1,2:X}", a, b);
                Console.WriteLine(" a={0,6:C} b={1,2:X}", a,
b);
                f.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
                return;
            }
        }
    }
}
```

У лістингу 4.4 файл, створений в попередньому прикладі, виводиться на екран.

Лістинг 4.4 – Читання текстового файлу

```
using System;
using System.IO;

namespace FileTest
{
    class Class1
    {
        static void Main(string[] args)
        {
            try
            {
                StreamReader f = new StreamReader("1.txt");
                string s = f.ReadToEnd();
                Console.WriteLine(s);
                f.Close();
```

```
            }
            catch (FileNotFoundException e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("Проверьте правильность имени
файла!");
                return;
            }
            catch (Exception e)
            {
                Console.WriteLine("Error:" + e.Message);
                return;
            }
        }
    }
}
```

У цій програмі весь файл прочитується за один прийом за допомогою методу *ReadToEnd*. Частіше виникає необхідність прочитувати файл порядково, такий приклад приведений в лістингу 4.5.

Лістинг 4.5 – Порядкове читання текстового файлу

```
using System;
using System.IO;

namespace FileTest
{
    class Class1
    {
        static void Main(string[] args)
        {
            try
            {
                StreamReader f = new StreamReader("1.txt");
                string s;
                long i = 0;
                while ((s = f.ReadLine()) != null)
```

```
                    Console.WriteLine(" {0} : {1} ", ++i, s);
                f.Close();
            }
            catch (FileNotFoundException e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine(" Перевірте правильність
імені файлу!");
                return;
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
                return;
            }
        }
    }
}
```

Приклад перетворення чисел, що містяться в текстовому файлі, в їх внутрішню форму представлення приведений в лістингу 4.6. У програмі обчислюється сума чисел в кожному рядку.

На вміст файлу накладаються строгі обмеження: числа мають бути розділені рівно одним пропуском, після останнього числа в рядку пропуску бути не повинно, файл не повинен закінчуватися символом переведення рядків. Методи розбиття рядка і перетворення в цілочисельне представлення розглядалися раніше.

Лістинг 4.6 – Перетворення рядків в числа

```
using System;
using System.IO;

namespace FileTest
{
    class Class1
    {
        static void Main(string[] args)
```

```
        {
            try
            {
                StreamReader f = new StreamReader("1.txt");
                string s;
                const int n = 20;
                int[] a = new int[n];
                string[] buf;
                while ((s = f.ReadLine()) != null)
                {
                    buf = s.Split(' ');
                    long sum = 0;
                    for (int i = 0; i < buf.Length; ++i)
                    {
                        a[i] = Convert.ToInt32(buf[i]);
                        sum += a[i];
                    }
                    Console.WriteLine("{0} сумма: {1}", s,sum);
                }
                f.Close();
            }
            catch (FileNotFoundException e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine("Перевірте правильність
                                   імені файлу!");
                return;
            }
            catch (Exception e)
            {
                Console.WriteLine("Error; " + e.Message);
                return;
            }
        }
    }
}
```

Результат роботи програми:

```
1  2  4   сума:  7
```

```
3  44  -3  6  сума:  50
8  1  1   сума:  10
```

Двійкові потоки

Двійкові файли зберігають дані в тому ж вигляді, в якому вони представлені в оперативній пам'яті, тобто у внутрішній формі представлення. Двійкові файли застосовуються не для перегляду їх людиною, а для використання в програмах. Вихідний потік *BinaryWriter* підтримує довільний доступ, тобто є можливість виконувати запис в довільну позицію двійкового файлу.

Двійковий файл відкривається на основі базового потоку, який найчастіше використовує потік *FileStream*.

Основні методи двійкових потоків приведені в таблицях 4.8 і 4.9.

Таблиця 4.8 – Найбільш важливі елементи класу `BinaryWriter`

Елемент	Опис
`BaseStream`	Базовий потік, з яким працює об'єкт `BinaryWriter`
`Close`	Закрити потік
`Flush`	Очистити буфер
`Seek`	Встановити позицію в потоці
`Write`	Записати значення в потік

Таблиця 4.9 – Найбільш важливі елементи класу `BinaryReader`

Елемент	Опис
`BaseStream`	Базовий потік, з яким працює об'єкт `BinaryReader`
`Close`	Закрити потік
`PeekChar`	Повернути наступний символ без переміщення внутрішнього покажчика в потоці
`Read`	Рахувати потік байтів або символів і зберегти в масиві
`ReadXXXX`	Рахувати з потоку дані певного типу (наприклад, `Readboolean`, `Readbyte`, `ReadInt32` і т. д.)

У лістингу 4.7 приведений приклад формування двійкового файлу. У файл записується послідовність дійсних чисел, а потім для демонстрації довільного доступу третє число замінюється числом 8888.

Лістинг 4.7 – Формування двійкового файлу

```
using System;
using System.IO;

namespace FileTest
{
    class Class1
    {
        static void Main(string[] args)
        {
            try
            {
                BinaryWriter fout = new BinaryWriter(new
FileStream(@"D:\1.txt", FileMode.Create));
                double d = 0;
                while (d < 4)
                {
                    fout.Write(d);
                    d += 0.33;
                };
                fout.Seek(16, SeekOrigin.Begin); // другий
елемент файлу
                fout.Write(8888d);
                fout.Close();
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
                return;
            }
        }
    }
```

```
}
```

При створенні двійкового потоку в нього передається об'єкт базового потоку. При установці покажчика поточної позиції у файлі враховується довжина кожного значення типу `double` – 8 байт.

У лістингу 4.8 приводиться програма, яка за допомогою екземпляра `BinaryReader` прочитує вміст файлу в масив дійсних чисел, а потім виводить цей масив на екран.

При читанні береться до уваги, що метод `ReadDouble` при виявленні кінця файлу генерує виключення `EndOfStreamException`. Оскільки в даному випадку це не помилка, тіло обробника виключень порожнє.

Лістинг 4.8 – Читання двійкового файлу

```
using System;
using System.IO;

namespace FileTest
{
    class Class1
    {
        static void Main()
        {
            try
            {
                FileStream f = new FileStream(@"D:\1.txt",
                                              FileMode.Open);
                BinaryReader fin = new BinaryReader(f);
                long n = f.Length/8; // кількість чисел у файлі
                double[] x = new double[n];
                long i = 0;
                try
                {
                    while (true)
                    {
                        x[i++] = fin.ReadDouble(); // читання
                    }
                }
```

```
                catch (EndOfStreamException e)
                {
                }
                foreach (double d in x)
                {
                    Console.Write(" " + d); // виведення
                }
                fin.Close();
                f.Close();
            }
            catch (FileNotFoundException e)
            {
                Console.WriteLine(e.Message);
                Console.WriteLine(" Перевірте правильність
імені файлу!");
                return;
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
                return;
            }
        }
    }
}
```

Результат роботи програми:

```
    0    0,33    8888    0,99    1,32    1,65    1,98    2,31
2,64    2,97    3,3    3,63    3,96
```

Завдання до лабораторної роботи

Відповідно до варіанта та обраного рівня складності виконати наступні завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати декілька завдань представлених вище	Завдання № 1	Завдання № 1
2	–	–	Завдання № 2

Завдання № 1

№ варіанта	Завдання
1	**Задача А:** Створити файл, який містить інформацію про особисту колекцію книголюба. Структура запису: шифр книги, автор, рік видання, місцезнаходження (номер стелажу, шафи та т.і.). Кількість записів довільна. **Задача В:** Написати програму, яка видає наступну інформацію: — місцезнаходження книги автора *A* назви *B*. Значення *A,B* ввести з клавіатури; — список книг автора *C*, які знаходяться в колекції; — кількість книг видання *X* року, які знаходяться в колекції.
2	**Задача А:** Створити файл-довідник, який містить дані про біполярні транзистори. Структура запису: марка, провідність (n-p-n, p-n-p), максимальний струм колектора, максимальна напруга колектор-емітер, мінімальний і максимальний коефіцієнти підсилення, максимальна робоча частота. Кількість записів довільна. **Задача В:** Написати програму, яка дозволяє шукати у довіднику: — всю інформацію по введеній марці транзистора з клавіатури; — по введеному з клавіатури струму, напрузі і коефіцієнту підсилення видати всі підходящі транзистори; — видати всі комплементарні пари транзисторів (у яких параметри однакові, а провідність різна).

№ варіанта	Завдання
3	**Задача А:** Утворити файл, який містить інформацію про співробітників університету. Структура запису: прізвище працюючого, назва відділу, рік народження, стаж роботи, посада, оклад. Кількість записів довільна. **Задача В:** Написати програму, яка видає дозволяє отримати наступну інформацію: — список працівників пенсійного віку на сьогоднішній день з зазначенням стажу роботи; — середній стаж працюючих у відділі *X*.
4	**Задача А:** Створити файл, який містить значення функції sin(x), cos(x), tg(x) коли *x* змінюється від 0 до 314 з кроком 0.5. **Задача В:** Написати програму, яка у файлі, шукає від'ємні елементи, і коли вони є, то виводить їх на екран. Коли від'ємних елементів немає, на екран вивести перший та останній елементи.
5	**Задача А:** Створити файл, який містить інформацію про наявність квитків і рейсів Аерофлоту. Структура запису: номер рейсу, пункт призначення, час вильоту, час прибуття, кількість вільних місць у салоні. Кількість записів довільна. **Задача В:** Написати програму, яка видає інформацію наступного типу: — час відправлення літаків у місто *X*; — наявність вільних місць на рейс у місто *X* ;з часом відправлення *Y*. ***Вказівки:*** значення *X,Y* вводиться по запиту з клавіатури.
6	**Задача А:** Написати програму яка створює файл, що містить інформацію про розклад телепрограм на день. Структура запису: назва програми, час початку програми, час закінчення програми. **Задача В:** Написати програму, яка дозволяє отримати наступну інформацію: — усю програму телепередач на день; — по введеному з клавіатури часу видати назву програми, котра буде транслюватися в цей час;

№ варіанта	Завдання
	— назву самої довгої та самої короткої (за тривалістю) телепрограми.
7	**Задача А:** Утворити файл, який містить інформацію про здачу студентами сесії. Структура запису: індекс групи, прізвище студента, оцінки з п`яти екзаменів та п`яти заліків («*З*» –зараховано, «*Н*» – не зараховано). Кількість записів довільна. **Задача В:** Написати програму, яка видає наступну інформацію: — прізвища невстигаючих студентів з вказівкою індексів груп та кількостей заборгованостей; — середній бал, отриманий кожним студентом групи *X*, та всією групою в цілому.
8	**Задача А:** Створити файл-довідник, який містить дані про напівпровідникові діоди. Структура запису: марка, максимальний струм, максимальна зворотна напруга, падіння напруги у відкритому стані, максимальна робоча частота. Кількість записів довільна. **Задача В:** Написати програму, яка дозволяє шукати у довіднику: — всю інформацію по введеній марці діода з клавіатури; — по введеному з клавіатури струму, зворотній напрузі і частоті видати всі підходящі діоди; — видати всю інформацію про діоди з падінням напруги у відкритому стані менше, ніж введено з клавіатури.
9	**Задача А:** Утворити файл, який містить інформацію про асортимент взуття в крамниці фірми. Структура запису: артикул, назва, кількість, ціна однієї пари. Кількість записів довільна. Артикул починається з літери *Ж* для жіночого взуття, *Ч* – чоловічого, *Д* – дитячого. **Задача В:** Написати програму, яка видає наступну інформацію: — про наявність та ціну взуття артикула *X*; — асортиментний список жіночого взуття з вказівкою назви та кількості пар кожної моделі, яка є у продажу.

№ варіанта	Завдання
10	**Задача А:** Утворити файл, який містить інформацію про пацієнтів дитячої клініки. Структура запису: прізвище пацієнта, стать, вік, місце проживання (місто), діагноз. Кількість записів довільна. **Задача В:** Написати програму, яка видає наступну інформацію: — кількість пацієнтів, які прибули до клініки з іншого міста; — список пацієнтів старших *X* років з діагнозом *Y*. Значення *X,Y* ввести з клавіатури.

Завдання № 2

Розробити клас, який містить метод для роботи з текстовою інформацією: зчитування тексту із файлу «Input.txt», форматування тексту та запис результату у файл «Output.txt».

№ варіанта	Завдання
1	Виконати коригування тексту, видаляючи символи переходу на новий рядок, які не завершують абзац. Вважати, що абзац починається з червоного рядка – символа табуляції.
2	Виконати коригування тексту, видаляючи повторювані й зайві знаки пробілів. Зайвими вважати відступи, поставлені перед абзацом.
3	Виконати коригування тексту, вставляючи символи пробілу для правильного розділення слів та знаків пунктуації у реченнях. Якщо пробіл вже є, то коригувати не потрібно.
4	Виконати коригування тексту, видаляючи з нього всі знаки переносів, враховуючи, що вони зустрічаються тільки в кінці рядка.
5	Виконати коригування тексту, додаючи до назв розділів виділення пропусками перед і після рядка. Якщо пропуск рядка вже є, то коригувати даний фрагмент не потрібно. Вважати, що назва розділу починається зі слова «розділ» і відповідної цифри.

№ варіанта	Завдання
6	Виконати коригування тексту таким чином, щоб у кожному рядку було до N символів і при цьому були відсутні розриви слів.
7	Виконати коригування тексту таким чином, щоб всі назви глав були написані прописними буквами. Вважати, що назви глав в тексті виділено пропусками перед і після одного рядка.
8	Виконати коригування тексту, видаляючи з нього всі зайві відступи рядків. Відступи, що використовуються для виділення назв глав, зайвими не вважаються. Назви глав починаються зі слова «глава» і відповідної цифри.
9	Виконати коригування тексту таким чином, щоб усі абзаци починалися з червоного рядка – символу табуляції.
10	Виконати коригування тексту, видаляючи з нього всі символи, які не відносяться до текстової інформації.
11	Виконати коригування тексту таким чином, щоб всі слова могли починатися з великої або малої літери, а всі інші літери були тільки малими.
12	Виконати коригування тексту, видаляючи всі тире, що поставлені перед початком рядка, крім тих, що починають абзац. Вважати, що абзаци починаються з червоного рядка – символу табуляції.
13	Виконати коригування тексту таким чином, щоб усі цифри в тексті були виділені пробілами. Для цифр, які починають речення, коригування не потрібне.
14	Виконати коригування тексту, видаляючи з нього всі зайві крапки. Крапки і трикрапки в кінці речень видаляти не потрібно.
15	Виконати коригування тексту, видаляючи всі символи табуляцій, крім тих, що починають абзаци і є червоним рядком.
16	Виконати для тексту формування в його кінці змісту без зазначення номерів сторінок (перелік глав). Вважати, що назви глав починаються зі слова «глава» і відповідної цифри.
17	Виконати для тексту нормалізацію всіх виносок таким чином, щоб вони містили число символів «*», яке постійно

№ варіанта	Завдання
	збільшується. Коригування повинно проходити окремо для виносок і їх розшифровок.
18	Виконати коригування тексту, додаючи в нього номери сторінок зліва знизу. Вважати, що один лист може містити 65 рядків, включаючи номер.
19	Виконати коригування тексту, додаючи для кожного рядка поле по 5 символів, так щоб в рядок містилося N символів і були відсутні розриви слів.
20	Виконати коригування тексту, видаляючи з нього всі назви глав. Вважати, що назви глав починаються зі слова «глава» і відповідної цифри.
21	Виконати коригування тексту, виділяючи всі слова, що містять нелітеральні символи за допомогою квадратних дужок.
22	Виконати для тексту формування на початку, після назви, списку номерів глав із зазначенням номерів сторінок, вважаючи, що лист може містити 70 рядків.
23	Виконати для тексту на початку кожного листа вставку колонтитулів, що містять назву тексту. Вважати, що один лист містить 68 рядків.
24	Виконати коригування тексту таким чином, щоб кожна глава починалася з нового аркуша, вважаючи, що глави починаються зі слова «глава» і відповідної цифри. Лист може містити до 60 рядків.
25	Виконати коригування тексту, видаляючи з нього всі виноски та їх розшифровки. Вважати, що виноска – це сукупність декількох символів «*», а розшифровка – речення, що починається з декількох символів «*», написане після тире.

Контрольні питання

1. Які засоби для роботи з файлами є у бібліотеці .NET Framework?
2. Чим відрізняються класи `BinaryReader` та `BinaryReader`?
3. Що таке потік при роботі з файлами?

4. Чим істотно відрізняються двійкові та текстові формати файлів?

5. Що таке серіалізація та десеріалізація об’єктів?

6. Які класи слугуть для роботи з форматом XML?

7. Яке місце в архітектурі .NET Framework займають класи `StreamReader` та `StreamWriter`?

8. Для чого призначена конструкція `using {...}` при роботі з класами `StreamReader` та `StreamWriter`?

9. Які виняткові ситуації можуть виникати при роботі з файлами?

10. Перерахуйте основні властивості та методи класу `File`.

ЛАБОРАТОРНА РОБОТА № 5

Тема: структурована обробка виняткових ситуацій (помилок).

Мета: розробити програми на мові C# та реалізувати генерування та перехоплення виняткових ситуацій різного типу.

Теоретичні відомості

Розробка програмного забезпечення є досить складним процесом. Тому завдяки такій складності у програмі можуть з'являтися помилки. Серед найбільш розповсюджених програмних аномалій можна виділити:

- програмні помилки;
- помилки користувача;
- виключні ситуації (зайнятий файл, відсутність зв'язку з БД).

Мова програмування C# пропонує чотири ключових слова (`try`, `catch`, `throw` та `finally`), які дозволяють генерувати і обробляти виключні ситуації.

Оператор `try` задає блок коду, призначений для обробки помилок або вивільнення пам'яті. За блоком `try` повинні розміщуватися блок `catch`, блок `finally` або обидва блока. Блок `catch` виконується, коли у блоці `try` виникає помилка. Блок `finally` виконується після того, як потік керування виходить із блока `try` (або блока `catch`). Він призначений для вивільнення пам'яті, незалежно від того, виникла помилка чи ні.

Оператор `try` виглядає таким чином:

```
try
{
... //В этом блоке можно генерировать исключение
}
catch (ExceptionA ex)
... // Обработка исключения типа ExceptionA
}
catch (ExceptionB ex)
{
```

```
... // Обработка исключения типа ExceptionB }
finally
{
... // Освобождения памяти
}
```

Розглянемо такий код:

```
int x = 3, y = 0;
Console.WriteLine (x / y);
```

Оскільки змінна *y* рівна нулю, середовище виконання генерує виняток типу `DivideByZeroException` і програма завершується. Це можна попередити за допомогою такого коду:

```
try
{
  int x = 3, y = 0;
  Console.WriteLine (x / y);
}
catch (DivideByZeroException ex)
{
  Console.Write ("y не может быть равным нулю. ");
  // После обработки исключения выполнение
  // возобновляется с этой точки...
}
```

Аудиторні завдання

Задання № 1

Обчислити значення виразу, передбачивши обробку помилок та видачу відповідних повідомлень користувачу. Параметри виразу *x*, *y*, *z* користувач вводить у консольному вікні з клавіатури.

$$\sum_{i=1}^{x} \frac{z}{i-y}$$

Розв’язання

Проаналізувавши математичний вираз, можна побачити, що коли $i = y$ виникає помилка ділення на нуль. Крім того, користувач може ввести нечислове значення у вікні консолі.

```
    static void Main(string[] args)
    {
        int x, y, z;
        try
        {
            Console.Write("X = ");
            x = Convert.ToInt32(Console.ReadLine());
            Console.Write("Y = ");
            y = Convert.ToInt32(Console.ReadLine());
            Console.Write("Z = ");
            z = Convert.ToInt32(Console.ReadLine());
            double Sum = 0;
            for (int i = 1; i <= x; i++)
            {
                Sum += z / (i - y);
            }
            Console.WriteLine("Result: {0}", Sum);
        }
        catch (FormatException)
        {
          Console.WriteLine("Error. You must input integer
    value");
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Divide by zero error");
        }
        Console.ReadKey();
    }
```

Задання № 2

Продемонструвати появу в програмі винятку `System.ArgumentOutOfRangeException`, використовуючи клас `List<>` і не користуючись ключовим словом `throw`.

Розв'язання

Переглянувши офіційну довідку Microsoft для методів класу `List<>` http://msdn.microsoft.com/en-us/library/0ebtbkkc(v=vs.110).aspx

(рисунок 5.1), можна виявити, що виняток `System.ArgumentOutOfRangeException` виникає, наприклад, при зверненні до невірного елемента списку. Скористуємося цією умовою.

List<T>.Item Property

.NET Framework 4.5 | Other Versions | 1 out of 10 rated this helpful - Rate this topic

Gets or sets the element at the specified index.

Namespace: System.Collections.Generic
Assembly: mscorlib (in mscorlib.dll)

Syntax

C# | C++ | F# | VB

```
public T this[
        int index
] { get; set; }
```

Parameters

index
Type: System.Int32
The zero-based index of the element to get or set.

Property Value
Type: T
The element at the specified index.
Implements
IList<T>.Item
IReadOnlyList<T>.Item

Exceptions

Exception	Condition
ArgumentOutOfRangeException	*index* is less than 0. -or- *index* is equal to or greater than Count.

Рисунок 5.1 – Фрагмент довідки про методи класу `List<>`

Один із можливих варіантів рішення подано нижче.

```
class Program
{
    static void Main(string[] args)
    {
        List<int> list = new List<int>();
        list.Add(10);
        list.Add(20);
        Console.WriteLine(list[20]); // error in this line
        Console.ReadKey();
    }
}
```

Завдання до лабораторної роботи

Відповідно до варіанта та обраного рівня складності виконати наступні завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Завдання № 1	Завдання № 1
2	–	–	Завдання № 2

Завдання № 1

Обчислити значення виразу, передбачивши обробку помилок та видачу відповідних повідомлень користувачу. Параметри виразу *x*, *y*, *z* користувач вводить у консольному вікні з клавіатури.

№ варіанта	Завдання
1	$\sum_{i=1}^{x-1} \frac{z \sin x}{i - y^2}$
2	$\prod_{i=1}^{y} \sqrt{\frac{i+z}{x-y+i^2}}$

№ варіанта	Завдання
16	$\sum_{i=1}^{zy} \frac{x^z}{x^2 + y - iz}$
17	$\sum_{i=1}^{x} (i - xy) \Big/ \sum_{i=1}^{y} (i - xz)$

№ варіанта	Завдання
3	$\sum_{i=0}^{y-x} \frac{\cos y \sin x}{i^2 - y^2}$
4	$\prod_{i=1}^{x+y+z} \sqrt{\frac{i+10}{x-y+i}}$
5	$\sum_{i=1}^{x} \prod_{j=1}^{y} \frac{i-j}{i+j+z}$
6	$\sum_{i=1}^{x} \sum_{j=2}^{y} \frac{z+x}{i+yz}$
7	$\prod_{i=1}^{x+y} \sqrt{\frac{2+i-z}{x^3-i+i^2}}$
8	$\sum_{i=1}^{z} \prod_{j=1}^{10} \frac{x-i+\cos j}{\sin i+j+z}$
9	$\sum_{i=4}^{z-x+y} \frac{z+x^i}{x+i-y}$
10	$\sum_{i=1}^{zy} \frac{x^y}{x^i+y-z}$
11	$\sum_{i=1}^{x} (i+x) \Big/ \sum_{i=1}^{y} (i+xz)$
12	$\sum_{i=1}^{z} \prod_{j=1}^{i} \frac{x-i}{i+j+z}$
13	$\sum_{i=0}^{xyz} \frac{\cos y \sin x}{i-z}$
14	$\sum_{i=1}^{x} \prod_{j=1}^{i} \frac{xyz}{i+j+z}$
15	$\sum_{i=1}^{x-1} \frac{z \sin x}{i-y^2} + \sum_{i=1}^{8} \sqrt{\frac{1}{xyz}}$

№ варіанта	Завдання
18	$\sum_{i=1}^{x-y} \frac{z \sin x}{i-y} + \sum_{i=1}^{x+z} \sqrt{\frac{1}{xy-z}}$
19	$\sum_{i=1}^{y+2} \frac{zx}{i-y^2}$
20	$\sum_{i=0}^{x+y+z} \frac{tg(y) \cdot ctg(x)}{i-2x}$
21	$\sum_{i=1}^{z} \frac{x^y+2}{x^i+y-z}$
22	$\sum_{i=1}^{z} \prod_{j=1}^{10} \frac{x-i+\cos j}{\sin i+j+z}$
23	$\sum_{i=1}^{x-1} \frac{z+\sin x}{i-y^2} + \sum_{i=1}^{4} \sqrt{\frac{1}{x+yz}}$
24	$\sum_{i=1}^{x} \sum_{j=2}^{y} \frac{x+y+2z}{i+z}$
25	$\sum_{i=0}^{xyz} \frac{i+x+y+z+2}{i-z}$
26	$\sum_{i=0}^{yx+z} \frac{\cos y \sin x}{i^2-y^2}$
27	$\prod_{i=x}^{x+y+z} \sqrt{\frac{i+10}{x-y+i}}$
28	$\sum_{i=1}^{x} \prod_{j=1}^{y} \frac{i-j}{i+j+z}$
29	$\sum_{i=1}^{x} \sum_{j=z}^{y} \frac{z+x}{i+yz}$
30	$\prod_{i=1}^{x+y} \sqrt{\frac{2i-zx}{x^3-i+i^2}}$

Завдання № 2

Виконати оброблення не менше двох виняткових ситуацій, які можуть виникати у програмі.

№ варіанта	Завдання
1	Користувач вводить назву файлу на жорсткому диску. Програма знаходить добуток 1-го, 3-го та 5-го чисел з цього файлу і виводить результат на консоль
2	Користувач вводить дату народження. Розрахувати вік користувача у днях. Якщо введено дату невірно, то згенерувати свій власний тип винятку `MyDateError`

3	Дано фрагмент коду. Виявити помилки, які можуть виникнути при роботі даного коду. Перерахувати відповідні винятки.
	`class Manager`
	`{`
	`  int Sum(object collection, int count)`
	`  {`
	`  List<int> list = (List<int>)collection;`
	`  int s = 0;`
	`  for (int i = 0; i < count; i++)`
	`  {`
	`  s += list[i];`
	`  }`
	`  return s;`
	`  }`
	`}`

4	У файлі знаходиться телефонний довідник, який містить прізвища абонентів та номери телефонів. У кожного абонента може бути декілька телефонних номерів. Користувач вводить назву файлу-довідника та прізвище клієнта, програма знаходить кількість номерів телефонів заданого клієнта

№ варіанта	Завдання
5	Користувач вводить свій номер телефону. Якщо введений номер не відповідає формату +38XX-XXX-XX-XX, то згенерувати свій власний тип винятку `MyPhoneError`
6	Користувач вводить назву файлу на жорсткому диску. Програма знаходить частку 4-го та 8-го чисел з цього файлу і виводить результат на консоль
7	Користувач вводить адреси сайтів, які додаються до файлу. Якщо користувач вводить сайт не у доменах .com або .net, згенерувати свій власний виняток до додати у файл запис «Error»
8	Продемонструвати появу в програмі винятків `FormatException`, `InvalidCastException` не користуючись ключовим словом `throw`
9	Користувач вводить назву файлу на жорсткому диску. Програма перевіряє, чи є у файлі одиниці вимірювання довжин, наприклад, 10 м, 5,35 мм тощо. Файл може містити довільний текст
10	Дано масив рядків. Користувач вводить через кому номери рядків, які потрібно видалити з масиву. Вивести результат на консоль

Контрольні питання

1. Що таке виняткова ситуація?
2. Які властивості має виключення `System.Exception`?
3. Які підходи існують для обробки програмних помилок?
4. Яким чином використовувати вкладені блоки `try…catch…finally`?
5. Які виключні ситуації можуть створювати методи класу `List`?

ЛАБОРАТОРНА РОБОТА № 6

Тема: перевантаження операцій та методи розширення.

Мета: розробити програми на мові C# та реалізувати перевантаження унарних та бінарних операцій.

Теоретичні відомості

Перевантажувати можна такі операції:

+ – * / ++ – – ! ~ % & | ^

== != < > << >>

У наступному прикладі об’явлено структуру `Note`, яка представляє музичну ноту, а потім виконано перевантаження операції +:

```
public struct Note
{
  int value;
  public Note (int semitonesFromA)
  {
    value = semitonesFromA;
  }
  public static Note operator x (Note x, int semitones)
  {
    return new Note (x.value + semitones);
  }
}
```

Це перевантаження дозволяє додавати об’єкт типу `Note` та змінну типу `int`.

```
Note B = new Note (2);
Note CSharp = B + 2;
```

Розширяючи даний приклад, виконаємо перевантаження операції порівняння для типу `Note`.

```
public static bool operator == (Note n1, Note n2)
{
  return n1.value == n2.value;
}
public static bool operator != (Note n1, Note n2)
{
```

```
  return !(n1.value == n2.value);
}
public override bool Equals (object othe rNote)
{
  if (KotherNote is Note)) return false;
  return this == (Note)otherNote;
}
public override int GetHashCode()
{
  return value.GetHashCodeO; // Используется хеш-код поля value
}
```

Аудиторні завдання

Задання № 1

Створити клас, який представляє комплексне число. Реалізувати перевантаження основних арифметичних операцій (+, –, *, /). Продемонструвати роботу даних операцій.

Розв’язання

Як відомо з курсу математики, комплексне число має дійсну та уявну частину.

```
class Complex
{
    public double Re;   // real part
    public double Im;   // imagine part
    public Complex() { }
    public Complex(double re, double im)
    {
        Re = re;
        Im = im;
    }
    public static Complex operator +(Complex a, Complex b)
    {
        return new Complex(a.Re + b.Re, a.Im + b.Im);
    }
    public static Complex operator -(Complex a, Complex b)
```

```
    {
        return new Complex(a.Re - b.Re, a.Im - b.Im);
    }
    public static Complex operator *(Complex a, Complex b)
    {
        return new Complex(a.Re * b.Re - a.Im * b.Im,
                           a.Re * b.Im + a.Im * b.Re);
    }
    public static Complex operator /(Complex a, Complex b)
    {
        double val = 1 / (b.Re * b.Re - b.Im * b.Im);
        return new Complex(val * (a.Re * b.Re + a.Im * b.Im),
                           val * (a.Re * b.Im - a.Im * b.Re));
    }
    public override string ToString()
    {
        return String.Format("{0} + {1}*j", Re, Im);
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Complex class testing");
        Complex a = new Complex(3, 4);
        Complex b = new Complex(2, -3);
        Complex c;
        Console.WriteLine("a = {0}", a);
        Console.WriteLine("b = {0}", b);
        c = a + b;
        Console.WriteLine("c = a + b = {0}", c);
        c = a - b;
        Console.WriteLine("c = a - b = {0}", c);
        c = a * b;
        Console.WriteLine("c = a * b = {0}", c);
        c = a / b;
        Console.WriteLine("c = a / b = {0}", c);
        Console.ReadKey();
```

```
    }
}
```

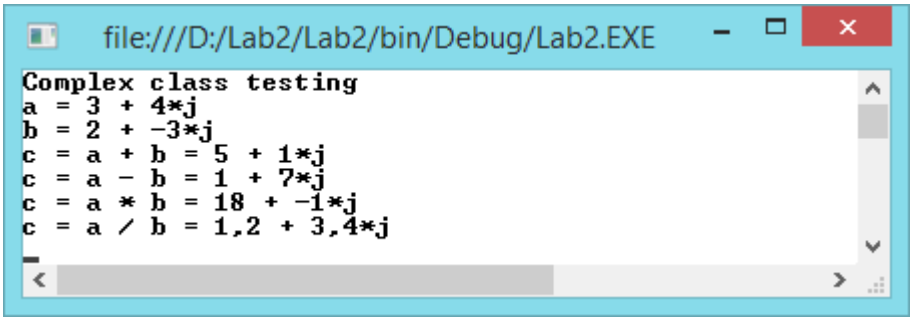

Рисунок 6.1 – Приклад виконання програми

Задання № 2

За допомогою методів розширення додати до типу `int` можливість зведення його в квадрат та куб.

Розв'язання

```
static class IntEx
{
    public static int Sqr(this int val)
    {
        return val * val;
    }
    public static int Cube(this int val)
    {
        return val * val * val;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(25.Sqr());
        Console.WriteLine((5 + 3).Cube());
        Console.ReadKey();
    }
}
```

Завдання до лабораторної роботи

Відповідно до варіанта та обраного рівня складності виконати наступні завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Завдання № 1	Завдання № 1
2	–	–	Завдання № 2

Завдання № 1

Створити клас, який представляє відповідний математичний об’єкт. Реалізувати перевантаження вказаних операцій. Продемонструвати роботу даних операцій у консольному додатку.

№ варіанта	Завдання	
	Математичний об’єкт	Операції для перевантаження
1	Вектор у двовимірному просторі	вектор + вектор вектор + число вектор – вектор вектор * вектор вектор * число
2	Матриця цілочисельних елементів 2×1	матриця + матриця матриця – матриця матриця * число матриця / число
3	Комплексне число	компл. число + число компл. число – число компл. число * число компл. число / число
4	Матриця цілочисельних елементів 2×2	матриця – матриця матриця + число матриця * число матриця / число

№ варіанта	Завдання	
	Математичний об’єкт	Операції для перевантаження
5	Вектор у тривимірному просторі	порівняння векторів >, <, <=, >=, ==, !=
6	Матриця дійсних елементів 2×2	матриця * матриця матриця + матриця матриця % число порівняння матриць ==, !=
7	Вектор у тривимірному просторі	вектор + вектор вектор + число вектор – вектор вектор * вектор вектор * число
8	Матриця дійсних елементів 1×2	порівняння матриць >, <, <=, >=, ==, !=
9	Вектор у двовимірному просторі	порівняння векторів >, <, <=, >=, ==, !=
10	Бітова множина на 8 бітів	зміщення бітів >>, << порівняння ==, !=
11	Вектор у двовимірному просторі	порівняння модулів векторів >, <, <=, >=, ==, !=
12	Матриця цілочисельних елементів 2×1	матриця * матриця матриця + матриця матриця * число порівняння матриць ==, !=
13	Комплексне число	компл. число + компл. число компл. число – компл. число порівняння ==, !=
14	Матриця цілочисельних елементів 2×2	матриця – матриця матриця + число матриця * матриця
15	Вектор у тривимірному просторі	вектор – число

№ варіанта	Завдання	
	Математичний об’єкт	Операції для перевантаження
		вектор + число векторний добуток векторів
16	Матриця дійсних елементів 2×2	порівняння детермінантів >, <, <=, >=, ==, !=
17	Вектор у тривимірному просторі	вектор – вектор вектор + вектор зміщення вектора >>, <<
18	Матриця дійсних елементів 3×2	матриця * матриця матриця + матриця матриця * число порівняння матриць ==, !=
19	Вектор у чотиривимірному просторі	вектор + вектор вектор + число вектор – вектор
20	Бітова множина на 10 бітів	зміщення бітів >>, << порівняння множин >, <, <=, >=, ==, !=

Завдання № 2

№ варіанта	Завдання
1	За допомогою методів розширення додати до типу `int` можливість визначення квадратного кореня та довільного степеню
2	За допомогою методів розширення додати до типу `string` можливість визначення кількості квадратних та круглих дужок, які мітяться у рядку тексту
3	За допомогою методів розширення додати до типу `string` можливість визначення кількості знаків пунктуації у реченні

№ варіанта	Завдання
4	За допомогою методів розширення додати до типу `double` можливість визначення квадратного кореня та довільного степеню
5	До класу `string` додати методи розширення для друку поточної дати та часу
6	До класу `string` додати методи розширення для видалення всіх зайвих пробілів (на початку, вкінці та в середині тексту)
7	За допомогою методів розширення додати до типу `float` можливість визначення квадратного кореня та довільного степеню
8	До класу `string` додати методи розширення для перестановки слів у випадковому порядку
9	За допомогою методів розширення додати до типу `int` можливість додавання рядка тексту, який представляє число
10	За допомогою методів розширення додати до типу `sbyte` можливість інвертування всіх бітів числа
11	Розширити клас `int` для розрахунку факторіала
12	Розширити клас `Array` для сортування елементів за спаданням
13	До класу `string` додати методи розширення для друку масиву
14	До класу `string` додати методи розширення для зведення до верхнього регістра всіх початкових літер слів
15	За допомогою методів розширення додати до типу `double` можливість визначення квадратного кореня та довільного степеню
16	Розширити клас `int` для визначення чи є число паліндромом

№ варіанта	Завдання
17	До класу `string` додати методи розширення для друку обсягу оперативної пам’яті та розміру розділів жорсткого диску
18	Розширити клас `int` для інвертування цифр числа місцями
19	За допомогою методів розширення додати до типу `string` можливість визначення кількості голосних літер у реченні
20	До класу `Array` додати методи розширення для друку масиву чисел

Контрольні питання

1. Що таке перевантаження операцій?
2. Які правила існують для методів розширення?
3. Як виконується перевантаження скорочених арифметичних операторів?
4. Чи можна виконати додавання методів у статичний клас?
5. Наведіть декілька класів, для яких доцільно виконувати перевантаження операцій >>, <<?

ЛАБОРАТОРНА РОБОТА № 7

Тема: робота з інтерфейсами.

Мета: розробити програми на мові C# із застосуванням інтерфейсів та похідних класів.

Теоретичні відомості

Інтерфейс – це список оголошень методів, властивостей, подій та індексаторів. Оголошення інтерфейсу подібне до класу, однак не містить модифікаторів доступу для членів і реалізацій. Інтерфейс не може мати конструкторів. Отож об'єкт інтерфейсу не можна утворити [8].

Наприклад, інтерфейс `IEnumerator` із простору імен `System.Collections` оголошений так:

```
interface IEnumerator
{
    //властивість
    object Current {get;}
    //методи
    bool MoveNext();
    void Reset();
}
```

Кажуть, що клас підтримує інтерфейс, якщо він містить реалізацію усіх оголошень інтерфейсу. Зокрема, клас підтримує інтерфейс `IEnumerator`, якщо він містить реалізацію властивості `Current` і методів `MoveNext` і `Reset`. За домовленістю назва інтерфейсу починається літерою I.

У попередньому пункті ми оголосили індексатор для класу `Points` і зазначили, що до нього не можна застосувати цикл `foreach`. Для того щоб клас `Points` підтримував колекції, він повинен виконати наперед оголошену домовленість: містити метод з назвою `GetEnumerator`, який повертає об'єкт деякого класу з підтримкою інтерфейсу `IEnumerator`. Це правило формалізується інтерфейсом `IEnumerable`, оголошеним у просторі імен `System.Collections` так:

```
    public interface IEnumerable
    {
        IEnumerator GetEnumerator();
    }
```

Перед тим як додати підтримку цього інтерфейсу до класу `Points`, утворимо допоміжний клас `PointsEnum`:

```
class PointsEnum: IEnumerator
{
    int location = -1;
    Points points;
    //конструктор класу
    public PointsEnum(Points points)
    {
        this.points = points;
        location = -1;
    }
    //реалізація членів інтерфейсу
    IEnumerator public object Current
    {
        get
        {
            if (location < 0 || location >= points.Count)
                throw new InvalidOperationException(
                  "Некоректний індекс");
            return points[location];
        }
    }
    public bool MoveNext()
    {
        ++location;
        return (location >= points.Count) ? false:true;
    }
    public void Reset()
    {
        location = -1;
    }
}
```

Код `class PointsEnum: IEnumerator` вказує, що клас підтримує інтерфейс `IEnumerator`, тобто містить реалізацію його членів. Якщо необхідно, щоб клас підтримував декілька інтерфейсів, то після двокрапки треба перелічити назви цих інтерфейсів, розділені комами. І, відповідно, реалізувати всі члени цих інтерфейсів. Клас `PointsEnum` працює з об'єктом типу `Points`, який передається параметром конструктора.

Додамо тепер до класу `Points` підтримку інтерфейсу `IEnumerable`:

```
public class Points : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        return new PointsEnum(this);
    }
}
```

Код `new PointsEnum(this)` утворює об'єкт типу `PointsEnum`, а метод повертає значення типу `IEnumerator`. Оскільки клас `PointsEnum` підтримує цей інтерфейс, то протиріччя тут не буде.

Тепер компілятор не заперечуватиме проти використання `foreach`:

```
Points q = new Points(5);
string s = "";
foreach (Point p in q) s += p.ToString();
```

Аудиторне завдання

Реалізувати ієрархію класів: квадрат, коло, трикутник. Визначити та використати інтерфейс `IFigure`. До інтерфейсу додати методи визначення площі та периметру геометричної фігури.

Розв'язання

Для роботи з координатами геометричних фігур доцільно створити відповідний клас Point.

```
class Point
{
    public double X { get; set; }
    public double Y { get; set; }
    public Point(double x, double y)
    {
        X = x;
        Y = y;
    }
}

interface IFigure
{
    double Square();
    double Perimeter();
}

class SquareObj : IFigure
{
    public Point Center { get; set; }
    public double Size { get; set; }
    public SquareObj(Point center, double size)

    {
        Center = center;
        Size = size;
    }
    public double Square()
    {
        return Size * Size;
    }
    public double Perimeter()
    {
        return 4 * Size;
    }
}

class CircleObj : IFigure
{
```

```
    public Point Center { get; set; }
    public double Radius { get; set; }

    public CircleObj(Point center, double r)
    {
        Center = center;
        Radius = r;
    }
    public double Square()
    {
        return Math.PI * Radius * Radius;
    }
    public double Perimeter()
    {
        return 4 * Math.PI * Radius;
    }
}

class TriangleObj : IFigure
{
    public Point P0 { get; set; }
    public Point P1 { get; set; }
    public Point P2 { get; set; }
    public TriangleObj(Point p0, Point p1, Point p2)
    {
        P0 = p0;
        P1 = p1;
        P2 = p2;
    }
    private double Dist(Point p0, Point p1)
    {
        double dx = p1.X - p0.X,
                dy = p1.Y - p0.Y;
        return Math.Sqrt(dx * dx + dy * dy);
    }
    public double Square()
    {
        double a = Dist(P0, P1),
               b = Dist(P1, P2),
```

```
                c = Dist(P2, P0),
                hp = 0.5 * (a + b + c);
            return Math.Sqrt(hp * (hp - a) * ( hp – b) * (hp - c));
        }
        public double Perimeter()
        {
            return Dist(P0, P1) + Dist(P1, P2) + Dist(P2, P0);
        }
}

class Program
{
        static void Main(string[] args)
        {
            Console.WriteLine("Interface testing");
            IFigure fig1 = new SquareObj(new Point(1.5, 2.0), 5.0);
            IFigure fig2 = new CircleObj(new Point(1.5, 2.0),
10.0);
            IFigure fig3 = new TriangleObj(new Point(1.0, 1.0),
                                           new Point(-2.0, -1.0),
                                           new Point(0.0, 2.0));
            Console.WriteLine("Fig1: S = {0:f2}; P = {1:f2}",
                              fig1.Square(), fig1.Perimeter());
            Console.WriteLine("Fig2: S = {0:f2}; P = {1:f2}",
                              fig2.Square(), fig2.Perimeter());
            Console.WriteLine("Fig3: S = {0:f2}; P = {1:f2}",
                              fig3.Square(), fig3.Perimeter());
            Console.ReadKey();
        }
}
```

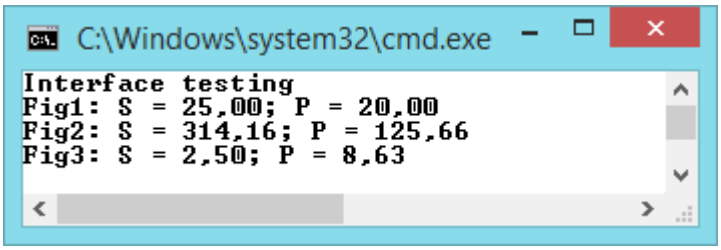

Рисунок 7.1 – Результат роботи програми

Завдання до лабораторної роботи

Відповідно до варіанта та обраного рівня складності виконати наступні завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Відкомпілювати та протестувати аудиторні завдання	Завдання № 1
2	–	Завдання № 1	Завдання № 2

Завдання № 1

Дати короткий опис методів та загальне призначення інтерфейсу.

№ варіанта	Завдання
1	`IEnumerable`
2	`IEnumerator`
3	`ICloneable`
4	`IComparable`
5	`ICollection`
6	`IDictionary`
7	`IDisposable`
8	`ISerializable`
9	`IConvertible`
10	`IEquatable`

№ варіанта	Завдання
11	`IQueryable`
12	`IAsyncResult`
13	`IXmlSerializable`
14	`IUnknown`
15	`IGrouping`
16	`IOrderedQueryable`
17	`IDeviceContext`
18	`IHtmlString`
19	`IHttpModule`
20	`ICollectData`

Завдання № 2

Реалізувати ієрархію класів. Визначити та використати інтерфейс `IFigure` (фігура). До інтерфейсу додати методи згідно з варіантом, а також властивості, пов’язані з відповідними даними класів.

№ варіанта	Завдання	
	Методи інтерфейсу `IFigure`	Класи геометричних фігур
1	визначення площі поверхні, наявності перетину	сфера, куб, тетраедр
2	визначення довжини, точки перетину	точка, відрізок, півколо
3	визначення наявності перетину, об’єму	конус, сфера, куб
4	розрахунок площі, периметру	квадрат, шестикутник, еліпс
5	розрахунок об’єму, наявності перетину	еліпсоїд, куб
6	розрахунок площі, всіх точок перетину	кільце, трапеція, ромб
7	розрахунок площі, периметру, наявності перетину	еліпс, коло, паралелограм
8	розрахунок довжини, всіх точок перетину	ламана лінія із 3 сегментів, відрізок, коло
9	розрахунок площі, повернення кількості сторін, точок перетину	трикутник, квадрат, ромб
10	розрахунок площі, периметру	трикутник, коло, прямокутник
11	розрахунок площі, наявності перетину	коло, прямокутник, трикутник
12	визначення площі поверхні, наявності перетину	сфера, куб, тетраедр
13	визначення довжини, точки перетину	точка, відрізок, півколо
14	визначення наявності перетину, об’єму	конус, сфера, куб
15	розрахунок площі, периметру	квадрат, шестикутник, еліпс

№ варіанта	Завдання	
	Методи інтерфейсу `IFigure`	Класи геометричних фігур
16	розрахунок об’єму, наявності перетину	конус, куб
17	розрахунок площі, всіх точок перетину	кільце, трапеція, ромб
18	розрахунок площі, периметру, наявності перетину	еліпс, паралелограм, квадрат
19	розрахунок довжини, всіх точок перетину	ламана лінія із 4 сегментів, відрізок, трикутник
20	розрахунок площі, повернення кількості сторін, точок перетину	паралелограм, квадрат, трикутник

Контрольні питання

1. Що таке інтерфейс?
2. Які правила існують для об’явлення методів інтерфейсів?
3. Як використовуються інтерфейси у .NET Framework?
4. Дайте коротку характеристику інтерфейсу `ICloneable.`
5. Порівняйте застосування абстрактних класів з інтерфейсами.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Основна література

1. Троелсен Э., Джепикс Ф. Язык программирования C# 7, платформы .NET и .NET Core : 8-е изд., пер. с англ. СПб. : Диалектика, 2018. 1328 с.
2. Скит Джон. C# для профессионалов. Тонкости программирования. М. : Вильямс, 2019. 608 с.
3. Тузовский А. Ф. Объектно-ориентированное программирование. М. : Юрайт, 2018. 206 с.
4. Албахари Джозеф, Албахари Бен. C# 7.0. Справочник. Полное описание языка 7-е изд. М. : Вильямс, 2018. 1220 с.
5. Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# : 4-е изд. СПб. : Питер, 2018. 896 с.
6. Васильев А. Программирование на C# для начинающих. Основные сведения. М. : Эксмо, 2018. 592 с.
7. Тепляков С. Паттерны проектирования на платформе .NET. СПб. : Питер, 2015. 320 с.

Додаткова література

8. Troelsen Andrew, Japikse Phil. Pro C# 8 with .NET Core 3. Foundational Principles and Practices in Programming : 9th ed. USA : Apress, 2020. 1262 p.
9. Jon Skeet. C# in Depth : 4th ed. USA : Manning Publications, 2019. 528 p.
10. John Sharp. Microsoft Visual C# Step by Step. USA : Microsoft Press, 2018. 832 p.
11. Nagel Christian. Professional C# 7 and .NET Core 2.0. Birmingham : Wrox, 2018. 1440 p.

12. Solis Daniel, Schrotenboer Cal. Illustrated C# 7. The C# Language Presented Clearly, Concisely, and Visually : 5th ed. USA : Apress, 2018. 817 p.

13. Perkins Benjamin, Jacob Vibe Hammer, Jon D. Reid. Beginning C# 7 Programming with Visual Studio 2017. Birmingham : Wrox, 2018. 912 p.

14. Sarcar Vaskaran. Design Patterns in C#. A Hands-on Guide with Real-World Examples. USA : Apress, 2018. 465 p.

15. MacDonald M. Pro .NET 2.0 Windows Forms and Custom Controls in C#. USA : Apress, 2006. 1081 p.

16. Албахари, Джозеф, Албахари, Бен. C# 8.0. Карманный справочник. СПб. : Диалектика, 2020. 240 с.

17. Прайс М. Дж. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов : 3-е изд. СПб. : Питер, 2018. 640 с.

18. Вагнер Билл. Эффективное программирование на C#. 50 способов улучшения кода. М. : Вильямс, 2017. 224 с.

19. Бойко Б. І., Омельчук Л. Л., Русіна Н. Г. Об’єктно-орієнтоване програмування. Лабораторний практикум : навч. посіб. К. : Київський національний університет ім. Тараса Шевченка, 2016. 90 с.

20. Гаско Рик. Объектно-ориентированное программирование. М. : Солон-Пресс, 2016. 298 с.

21. Фленов М. Е. Библия C# : 3-е изд., перераб. и доп. СПб. : БХВ-Петербург, 2016. 544 с.

22. Бублик В. В. Об’єктно-орієнтоване програмування : підручник К. : ІТ-книга, 2015. 624 с.

23. Петцольд Ч. Программирование с использованием Microsoft Windows Forms. Мастер-класс. М. : Русская Редакция; СПб. : Питер, 2006. 432 с.

ДОДАТОК А
Правила оформлення звіту

1. Перед виконанням лабораторної роботи необхідно уважно вивчити теоретичну частину до неї.

2. Створити документ Microsoft Word.

2.1. Встановити параметри сторінки: відступ зліва – 2 см; відступ справа – 1,5 см; відступ знизу – 2 см; відступ зверху – 1,5 см.

2.2. Розмір шрифту – 14, міжрядковий інтервал – одинарний.

2.3. Нумерація звіту повинна бути наскрізною, починаючи з титульного аркуша. Номер сторінки зазначають посередині верхньої частини аркуша над текстом. На титульному аркушу номер сторінки не ставлять, але рахують.

2.4. У звіті можуть бути наведені переліки, перед якими ставлять двокрапку. Перед кожною позицією переліку ставлять риску (–) або рядкову літеру з дужкою. Для подальшої деталізації переліків використовують арабські цифри з дужкою, наприклад:

а) . . . . . .

б) . . . . . .

1) . . . . . .

2) . . . . .

в) . . . . . .

2.5. У тексті звіту не дозволяється: вживати звороти розмовної мови; вживати застарілі та жаргонні терміни і вислови; вживати скорочені слова, крім встановлених стандартами скорочень.

2.6. Якщо в тексті наводиться ряд числових значень в однакових одиницях, то позначення одиниці виміру зазначають тільки після останнього числового значення, наприклад: 1, 2, 3 м; або від 5 до 10 мм. Одиниці вимірювання від числових величин відокремлюють нерозривним пробілом (Ctrl+Shift+Space).

2.7. Числові значення величин треба відокремлювати від десяткової частини комою, наприклад: 7,5; 8,75; 10,00. У необхідних випадках треба використовувати математичне округлення, наприклад: вірно...розмір файлу складає 20 МБ; невірно … розмір файлу складає 20,036 МБ.

2.8. Послідовність розміщення матеріалу у звіті повинна бути наступною:

− титульний аркуш (додаток А);

− зміст;

− лабораторні роботи:

- номер лабораторної роботи (стиль *Заголовок1)*;
- тема та мета роботи;
- анотовані теоретичні відомості, які застосовуються при розв'язанні індивідуальних завдань;
- умова завдання та його розв'язок;
- матеріали самостійної роботи студента згідно виданого завдання (матеріали повинні бути оброблені та систематизовані, не допускається прямого копіювання з джерел інформації);
- висновок до лабораторної роботи;
- список використаних джерел.

− загальний висновок до лабораторного курсу.

2.9. Оформлення ілюстрацій.

2.9.1. Усі ілюстрації у звіті у вигляді креслень, ескізів, схем, графіків, діаграм, фотографій та ін. називаються рисунками.

2.9.2. Рисунки можуть бути виконані олівцем, пастою, тушшю, фломастером та розташовані на окремих аркушах або безпосередньо в тексті записки, якщо рисунки невеликі.

2.9.3. Рисунки нумеруються в межах кожної лабораторної роботи двома цифрами – номером роботи і порядковим номером рисунку в роботі, розділеними крапкою.

2.9.4. Кожний рисунок повинен мати найменування. Слово «Рисунок», його номер та найменування розміщують під рисунком та записують таким чином:

Рисунок 1.3 – Алгоритм роботи програми

2.9.5. Після номеру ставиться тире (–), а після найменування крапка не ставиться.

2.9.6. На усі рисунки повинні бути посилання у тексті роботи, наприклад: ... наведено на рисунку 2.6.

2.9.7. Графіки повинні мати координатні осі та координатну сітку. На координатних осях необхідно наносити числові значення змінних

величин; найменування фізичної величини, яка пишеться текстом паралельно відповідній осі, та через кому позначають одиницю виміру фізичної величини. Напис розміщують поза полем графіка, у кінці напису крапка не ставиться.

2.10. Оформлення таблиць.

2.10.1. Таблиці нумерують у межах кожної роботи арабськими цифрами, розділеними крапкою, та розташовують над таблицею ліворуч. Кожна таблиця повинна мати назву, яку пишуть над таблицею. Перед назвою таблиці пишуть слово «Таблиця» і її номер, який складається з номера розділу і порядкового номера таблиці в межах розділу. Номер таблиці від назви виділяють тире, наприклад:

Таблиця 4.1 – Технічні характеристики модему ADE-4400

2.10.2. Якщо висота таблиці перевищує одну сторінку, її продовження переноситься на наступну сторінку. При цьому лінію, що обмежує першу частину таблиці знизу, не проводять, а над продовженням таблиці на наступній сторінці пишуть «Продовження таблиці 4.1». При переносі таблиці допускається її головку замінювати номерами граф, відповідно до їх номерів у першій частині таблиці.

2.10.3. На всі таблиці повинні бути посилання у тексті записки, наприклад: ... наведено в таблиці 4.1.

2.11. Оформлення формул.

2.11.1. Формули і математичні рівняння подаються у тексті окремим рядком і розташовуються на його середині. Переносити формулу на наступний рядок дозволяється тільки по знаках операцій, який повторюють на початку наступного рядка.

2.11.2. Формули нумерують у межах розділу арабськими цифрами. Номер складається з номера розділу та порядкового номера формули, розділених крапкою. Номер формули записують у круглих дужках на рівні праворуч формули. Посилання на формули у тексті записки дають у дужках, наприклад: ... у формулі (2.1).

2.11.3. Пояснення символів і числових коефіцієнтів, які входять у формулу, необхідно подавати безпосередньо під формулою. Пояснення кожного символу треба давати з нового рядка, причому перший рядок пояснення повинен починатися зі слова «де» без двокрапки після нього.

2.12. Оформлення списку використаних джерел.

2.12.1. Посилання на джерело наводиться у вигляді порядкового номера джерела, взятого в квадратні дужки. Якщо необхідно посилатися одночасно на декілька джерел, їх номери зазначають через кому чи тире, наприклад: [12]; [1,4,7]; [5-9]; [2 с. 4]; [3 таблиця 2.1].

2.12.2. Перелік літературних джерел розміщують у порядку їх згадування у роботі (найзручніший спосіб) або в алфавітному порядку.

2.12.3. Бібліографічний опис джерела в переліку має відповідати вимогам ДСТУ ГОСТ 7.1:2006 «Система стандартів з інформації, бібліотечної та видавничої справи. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання». Бібліографічний опис дається мовою оригіналу. Прізвища авторів від їх ініціалів відокремлюють нерозривним пробілом (Ctrl+Shift+Space).

2.12.4. Дозволяється також у якості джерел інформації використовувати ресурси глобальної мережі інтернет, проте тільки офіційні сайти виробників обладнання, інтернет-магазинів для складання кошторису тощо. Заборонено включати до переліку використаних джерел такі сайти як www.google.com, www.yandex.ru, www.yahoo.com та ін., які є загальними пошуковими сервісами, а також www.wikipedia.org, www.ukrreferat.com, www.referat.ru та ін., де інформація може мати неперевірений характер і додаватися на сайт нефахівцями.

3. До захисту поточної лабораторної роботи допускаються студенти, які надали звіт в електронному варіанті і захистили попередні лабораторні роботи.

4. Кожен студент захищає лабораторну роботу індивідуально.

5. На захисті викладач може задати будь-яке питання, що стосується теоретичної частини, і будь-яке завдання іншої бригади або підсумкове.

6. На останньому занятті викладачу подається роздрукований звіт з лабораторних робіт та його електронний варіант.

7. До заліку допускаються студенти, які захистили всі лабораторні роботи.

ЗМІСТ

М е т о д и ч н і в к а з і в к и
до виконання лабораторних робіт
з дисципліни «Об’єктно-орієнтоване програмування»
для студентів спеціальності
123 «Комп’ютерна інженерія»
усіх форм навчання.
Частина 1

УКЛАДАЧІ: Музика Іван Олегович
Кузнєцов Денис Іванович

Реєстраційний № ___

Підписано до друку _____ 2021 р.
Формат _____А5_____
Обсяг _____155_____ стор.
Тираж _____ прим.

Видавничий центр
Криворізького національного університету,
вул. Віталія Матусевича, 11, м. Кривий Ріг