ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА КІБЕРНЕТИКИ КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Л.Л. Омельчук
А.О. Свистунов

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Лабораторний практикум

Навчальний посібник

Київ 2023

УДК 519.85 (075.8)

ББК 32.973.2-018я73

О57

Рецензенти:

д-р фіз.-мат. наук *С.С. Шкільняк,*

к-т фіз.-мат. наук *О.В. Шишацька*.

Рекомендовано до друку вченою радою факультету комп'ютерних наук та кібернетики (протокол № 1 від "05" вересня 2023 року)

Омельчук Л.Л.

О57 Об'єктно-орієнтоване програмування. Лабораторний практикум: навчальний посібник / Л.Л. Омельчук, А.О. Свистунов. – Київ: 2023. – 320 с.

УДК 519.85 (075.8)

ББК 32.973.2-018я73

О57

Викладено матеріали до лабораторного практикуму з обов'язкової навчальної дисципліни «Об'єктно-орієнтоване програмування». В посібнику наведено умови лабораторних робіт, порядок розрахунку семестрової оцінки, теоретичний матеріал лабораторного практикуму, завдання для самостійної роботи та лабораторних занять, перелік контрольних запитань до кожної теми.

Для студентів другого курсу факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, які навчаються за освітньо-професійною програмою «Інформатика» спеціальності 122 «Комп'ютерні науки».

© Омельчук Л.Л., 2023

ЗМІСТ

ВСТУП

Навчальна дисципліна «Об'єктно-орієнтоване програмування» входить до переліку обов'язкових дисциплін освітньо-професійної програми «Інформатика» підготовки фахівців за освітньо-кваліфікаційним рівнем «бакалавр» в галузі знань 12 «Інформаційні технології» за спеціальністю 122 «Комп'ютерні науки». Дисципліна «Об'єктно-орієнтоване програмування» викладається у 3 семестрі 2 курсу в обсязі – 120 год.

(4 кредити ECTS) зокрема: лекції – 28 год., лабораторні заняття– 28 год., консультації – 2 год., самостійна робота – 62 год. У курсі передбачено 2 частини та 2 контрольні роботи. Завершується дисципліна – іспитом.

Мета дисципліни – засвоєння базових знань з основ об'єктно-орієнтованого програмування, включаючи основні поняття, парадигми та принципи об'єктно-орієнтованого програмування. Оволодіння базовими навичками проєктування програмних систем, роботи з найбільш вживаними шаблонами проєктування, набуття навичок об'єктно-орієнтованого програмування та оволодіння мовою програмування C#.

Попередні вимоги до опанування або вибору навчальної дисципліни (*за наявності*):

1. Знати: основні поняття програмування та принципи розробки програм; базові класичні алгоритми та різновиди структур даних

2. Вміти: проєктувати, розробляти та тестувати програми на базовому рівні.

3. Володіти елементарними навичками: програмування мовами C, C++.

Завдання (навчальні цілі):

набуття знань, умінь та навичок (компетентностей) на рівні новітніх досягнень у програмуванні, відповідно освітньої кваліфікації «Бакалавр з комп'ютерних наук». Зокрема, розвивати:

- здатність застосовувати знання у практичних ситуаціях;
- здатність оцінювати та забезпечувати якість виконуваних робіт;
- здатність проєктувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування:

узагальненого, об'єктноорієнтованого, функціонального, логічного, з відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмами управління.

Місце дисципліни. Нормативна навчальна дисципліна «Об'єктно-орієнтоване програмування» є складовою циклу професійної підготовки фахівців освітньо-кваліфікаційного рівня «бакалавр».

Зв'язок з іншими дисциплінами. Дисципліна "Об'єктно-орієнтоване програмування" є базовою для засвоєння всіх інших курсів та спецкурсів програміського спрямування, окремих розділів теорії алгоритмів та математичної логіки.

Результати навчання за дисципліною:

Результат навчання (1. знати; 2. вміти; 3. комунікація; 4. автономність та відповідальність)		**Форми (та/або методи і технології) викладання і навчання**	**Методи оцінювання та пороговий критерій оцінювання (за необхідності)**	**Відсоток у підсумковій оцінці з дисципліни**
Код	**Результат навчання**			
РН1.1	*Знати основні поняття об'єктно-орієнтованого програмування.*	*Лекція, лабораторне заняття*	*Контрольна робота, домашні завдання, Іспит*	16%
РН1.2	*Знати основні етапи життєвого циклу ПС.*	*Лекція, лабораторне заняття*	*Контрольна робота, домашні завдання, Іспит*	15%
РН1.3	*Знати шаблони, антишаблони та принципи об'єктно-орієнтованого проєктування програмного забезпечення.*	*Лекція, лабораторне заняття*	*Контрольна робота, домашні завдання, Іспит*	20%

РН2.1	*Вміти застосовувати на практиці інструментальні програмні засоби проєктування та розробки програмного забезпечення.*	*Лабораторне заняття, самостійна робота*	*Захист лабораторної роботи, Іспит*	23%
РН3.1	*Обґрунтовувати власний погляд на задачу, спілкуватися з колегами з питань проєктування та розробки програм, складати письмові звіти*	*Лабораторне заняття*	*Захист ЛР*	10%
РН4.1	*Організовувати свою самостійну роботу для досягнення результату*	*Самостійна робота*	*Захист лабораторної роботи*	8%
РН4.2	*Відповідально ставитися до виконуваних робіт, нести відповідальність за їх якість*	*Лабораторне заняття*	*Захист лабораторної роботи*	8%

Співвідношення результатів навчання дисципліни із програмними результатами навчання

Результати навчання дисципліни **Програмні результати навчання**	**РН 1.1**	**РН 1.2**	**РН 1.3**	**РН 2.1**	**РН 3.1**	**РН 4.1**	**РН 4.2**
(з опису освітньої програми)							
ПРН9. Розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв’язання задач в галузі комп’ютерних наук.	+			+			+

ПРН15. Застосовувати знання методології та CASE-засобів проєктування складних систем, методів структурного аналізу систем, об'єктно-орієнтованої методології проєктування при розробці і дослідженні функціональних моделей організаційно-економічних і виробничо-технічних систем.		+	+		+	+	

Схема формування оцінки

Форми оцінювання студентів:

До третього тижня семестру студент повинен визначитися з формою роботи на лабораторних заняттях, яка буде відповідати трьом рівням підготовки: високий (перший) рівень, середній (другий) рівень чи початковий (третій) рівень та підтвердити свій вибір проходженням відповідного опитування. У разі, якщо здобувач вчасно не зробив вибір свого рівня викладач здійснює цей вибір самостійно враховуючи рейтинг здобувача з дисципліни «Програмування» 1-2 семестри.

- семестрове оцінювання (за рівнями):

Високий (перший) рівень

1. Контрольна робота (тест) 1: РН 1.1, РН 1.2 — 10 балів / 6 балів.

2. Контрольна робота (тест) 2: РН1.3 – 10 балів / 6 бали.

3. Домашні завдання 1, 2, 8: РН 1.1., РН 1.2, РН1.3- 6 балів (2+2+2) / 4,8 балів.

4. Захист лабораторної роботи 1 (проєкту): РН 2.1, РН3.1, РН4.1, РН4.2 – 20 балів / 16 балів.*

5. Захист лабораторної роботи 2 (проєкту): РН 2.1, РН3.1, РН4.1, РН4.2 – 14 балів / 11,2 балів.*

Завдання на додаткові бали:

1. *Неформальна освіта:* сертифікат. Для реалізації цього права студент повинен до 1 жовтня поточного року написати та прикріпити в Classroom заяву (в якій зазначено форму контролю, яку замінюємо та інформація про обраний курс) зі своїм персональним підписом. Курс повинен бути попередньо узгоджений з викладачем. Дата отримання сертифікату повинна бути в інтервалі від 01 вересня до 01 грудня поточного року – *максимум 5 балів (1 сертифікат).*

2. *Публічна доповідь на тему «Шаблони проєктування» – максимум 3 бали.*

3. *Виконання лабораторних робіт інших рівнів.*

Студент має право замінити виконання та захист лабораторних робіт 1* та 2* участю в реалізації командного проєкту в межах ІТ-проєкту міждисциплінарної інтеграції (*для допуску необхідно пройти оцінювання рівня знань та умінь з програмування*).

Середній (другий) рівень

1. Контрольна робота (тест) 1: РН 1.1, РН 1.2 — 10 балів / 6 балів.

2. Контрольна робота (тест) 2: РН1.3 – 10 балів / 6 бали.

3. Домашні завдання 1, 2, 6, 7, 8: РН 1.1, РН 1.2, РН1.3 – 13 балів (2+2+3+4+2) / 7,8 балів.

4. Захист лабораторної роботи 1 (проєкту): РН 2.1, РН3.1, РН4.1, РН4.2 – 16 балів / 11 балів.

5. Захист лабораторної роботи 2 (проєкту): РН 2.1, РН3.1, РН4.1, РН4.2 – 11 балів / 8,8 балів.

Завдання на додаткові бали:

1. *Неформальна освіта:* сертифікат. Для реалізації цього права студент повинен до 1 жовтня поточного року написати та прикріпити в Classroom заяву (в якій зазначено форму контролю, яку замінюємо та інформація про обраний курс) зі своїм персональним підписом. Курс повинен бути попередньо узгоджений з викладачем. Дата отримання сертифікату повинна бути в інтервалі від 01 вересня до 01 грудня поточного року – *максимум 5 балів (1 сертифікат).*

2. *Публічна доповідь на тему «Шаблони проєктування» – максимум 3 бали.*

3. Виконання лабораторних робіт інших рівнів.

Початковий (третій) рівень

1. Контрольна робота (тест) 1: РН 1.1., РН 1.2 — 10 балів / 6 балів.

2. Контрольна робота (тест) 2: РН1.3 – 10 балів / 6 бали.

3. Домашні завдання 1-8: РН 1.1, РН 1.2, РН1.3 – 21 бал (2+2+3+2+3+3+4+2) / 12,6 балів.

4. Захист лабораторної роботи 2 (проєкту): РН 2.1, РН3.1, РН4.1, РН4.2 – 11 балів / 8,8 балів.

5. Захист лабораторної роботи 3 (проєкту): РН 2.1, РН3.1, РН4.1, РН4.2 – 8 балів / 6,4 балів.

Завдання на додаткові бали:

1. *Неформальна освіта:* сертифікат. Для реалізації цього права студент повинен до 1 жовтня поточного року написати та прикріпити в Classroom заяву (в якій зазначено форму контролю, яку замінюємо та інформація про обраний курс) зі своїм персональним підписом. Курс повинен бути попередньо узгоджений з викладачем. Дата отримання

сертифікату повинна бути в інтервалі від 01 вересня до 01 грудня поточного року – *максимум 5 балів.*

2. *Публічна доповідь на тему «Шаблони проєктування» – максимум 3 бали.*

3. Виконання лабораторних робіт інших рівнів.

Максимально можна отримати 10 додаткових балів. Максимально за семестр можна отримати 60 балів.

Напередодні запланованої дати захисту завдання студент повинен прикріпити матеріали із виконаним завданням в Classroom.

- підсумкове оцінювання (у формі іспиту):

- *максимальна кількість балів які можуть бути отримані студентом: 40 балів;*

- *результати навчання які будуть оцінюватись: РН1.1, РН1.2, РН1.3, РН2.1;*

- *форма проведення і види завдань: письмова робота.*

Види завдань: 8 тестових та 6 письмових завдань.

Критерії оцінювання на іспиті

Завдання	**Тема завдання**	**Максимальний бал**	**Всього балів**
Завдання 1, 2, 3	Письмове запитання з S.O.L.I.D.	По 2 бали	15 балів
Завдання 4, 5, 6	Письмові запитання з ООП та C#	По 3 бали	
Завдання 7, 8, 9, 10, 12, 13, 14, 15, 16	Тестові завдання з матеріалів частини 1 та частини 2	По 1-4 бали	19 балів
Завдання 17	Письмове практичне завдання з ООП	6 балів	6 балів
			40 балів

Умови домашніх завдань:

1. Use-case-діаграма та діаграма класів для заданої предметної області.

2. Діаграма класів (доопрацювання) та діаграма послідовності використання для заданої предметної області.

3. Тестові завдання на теми: поліморфізм, спадкування, exception. Практичні завдання з програмування.

4. Тестові завдання на теми: поліморфізм (продовження), абстрактні класи, інкапсуляція. Практичні завдання з програмування.

5. Тестові завдання до теми: тип var, dynamic, структури і класи, properties, enum.

6. Тестові та практичні завдання на теми: JSON, Dictionary, LINQ.

7. Тестові завдання на тему шаблони проєктування. Практичні завдання з програмування на тему шаблони проєктування.

8. Практичні завдання з програмування на тему S.O.L.I.D.

Умови лабораторних робіт:

Лабораторна робота 1:	Розробка програми для роботи з електронними таблицями (аналіз та обчислення виразів). З використанням MAUI. З реалізацією збереження та читання файлів з таблицями, що розміщені у локальному доступі.
Лабораторна робота 1*:	Розробка програми для роботи з електронними таблицями (аналіз та обчислення виразів). З використанням MAUI. З реалізацією збереження та читання файлів з таблицями, що розміщені в Google Grive.
Лабораторна робота 2:	Робота з XML-документами. З використанням MAUI. З реалізацією збереження та читання XML-файлів, що розміщені у локальному доступі.
Лабораторна робота 2*:	Робота з XML-документами. З використанням MAUI. З реалізацією збереження та читання XML-файлів з таблицями, що розміщені в Google Grive.
Лабораторна робота 3:	Розробка програм обробки файлів у форматі JSON (диспетчер для JSON).

Організація оцінювання:

Терміни проведення форм оцінювання:

1. Контрольна робота 1 (тест): до 7 тижня семестру.

2. Контрольна робота 2 (тест): до 14 тижня семестру.

3. Домашні завдання 1: до 2 тижня семестру.

4. Домашні завдання 2: до 3 тижня семестру.

5. Домашні завдання 3: до 4 тижня семестру.

6. Домашні завдання 4: до 5 тижня семестру.

7. Домашні завдання 5: до 6 тижня семестру.

8. Домашні завдання 6: до 8 тижня семестру.

9. Домашні завдання 7: до 9 тижня семестру.

7. Домашні завдання 8: до 13 тижня семестру.

10. Захист лабораторної роботи 1 (проєкту): до 8 тижня семестру (UML-діаграми до проєкту до 8 тижня семестру).

11. Захист лабораторної роботи 1 (проєкту): до 9 тижня семестру (UML-діаграми до проєкту до 7 тижня семестру).*

12. Захист лабораторної роботи 2 (проєкту): до 12 тижня семестру (UML-діаграми до проєкту до 10 тижня семестру).

12. Захист лабораторної роботи 3 (проєкту): до 13 тижня семестру (UML-діаграми до проєкту до 11 тижня семестру).

13. Захист лабораторної роботи 3 (проєкту): до 12 тижня семестру (UML-діаграми до проєкту до 10 тижня семестру).

15. Неформальна освіта: сертифікат. Для реалізації цього права студент повинен до 1 жовтня поточного року написати та прикріпити в Classroom заяву (в якій зазначено форму контролю, яку замінюємо та інформація про обраний курс) зі своїм персональним підписом. Курс повинен бути попередньо узгоджений з викладачем. Дата отримання сертифікату повинна бути в інтервалі від 01 вересня до 01 грудня поточного року.

16. Публічна доповідь на тему «Шаблони проєктування» – до 12 тижня семестру.

Студент має право на одне перескладання кожної контрольної роботи із можливістю отримання максимально 80% початково визначених за цю контрольну роботу балів. Термін перескладання визначається викладачем.

У разі неякісного виконання лабораторної роботи, або домашнього завдання, викладач має право не зарахувати це завдання, або знизити за нього бали.

Студент має право захистити лабораторні роботи після закінчення визначеного для них терміну, але з втратою 1 балу за кожен тиждень, який пройшов з моменту закінчення терміну їх здачі.

Студент має право здавати домашні завдання роботи після закінчення визначеного для них терміну, але з втратою 0,5 балів за кожен тиждень, який пройшов з моменту закінчення терміну їх здачі.

Шкала відповідності оцінок

Відмінно / Excellent	90-100
Добре / Good	75-89
Задовільно / Satisfactory	60-74
Незадовільно / Fail	0-59

Усі види робіт приймаються ВИКЛЮЧНО під час лабораторних занять.

Принаймні за два тижні до здачі **кожної з лабораторних робіт** обов'язковим є демонстрація трьох діаграм UML (1 діаграма прецедентів (Use case diagram), 1 діаграма класів (Class diagram) та 1 діаграма послідовності (Sequence diagram)) з можливістю отримання 1 балу за кожен набір діаграм.

Умови домашніх завдань оголошуються викладачем на лабораторному занятті, що передує терміну здачі.

Студент допускається до складання іспиту, якщо кількість набраних за семестр балів не менше 36.

Підсумкова екзаменаційна робота – 40 балів.

УМОВИ ЛАБОРАТОРНИХ РОБІТ

Наведена нижче умова завдання не є вичерпною чи абсолютно точною. Автору розробки надається можливість розширювати поставлене завдання та виходити за його рамки.

Метою виконання будь-якого проєкту є одержання нових знань, навичок та умінь. Поряд з цим він має носити ще й дослідницький та творчий характер. Так, наприклад у ньому має бути не тільки підтверджене уміння використовувати визначену інструментальну систему, але експериментально доведено, що якісь частини проєкту могли б мати іншу (напр. більш ефективну) реалізацію.

Під "обробкою текстів" тут розуміється клас задач обробки інформації нечислового характеру, наприклад редагування тексту, форматування документів, пошук інформації, сортування файлів даних, лексичний і синтаксичний аналіз текстів, друк документів, перетворення формату документів, що містять тексти, таблиці і.т.ін.

Має бути головне меню та кнопки, що дублюють основні його елементи. Повинна бути подана коротка інформація про програму та надаватися допомога користувачу. Діалог україномовний.

Перевірка виконаних робіт здійснюється виключно за розкладом. Разом з проєктом має бути набір тестових файлів та інших матеріалів, що свідчать про правильність створених програм. Автор проєкту повинен щотижня інформувати викладача про виконані етапи робіт.

Лабораторна робота № 1

РОЗРОБКА ПРОГРАМИ ДЛЯ РОБОТИ З ЕЛЕКТРОННИМИ ТАБЛИЦЯМИ (АНАЛІЗ ТА ОБЧИСЛЕННЯ ВИРАЗІВ)

Наведена нижче умова завдання не є вичерпною чи абсолютно точною. Автору розробки надається можливість розширювати поставлене завдання та виходити за його рамки.

Використання антишаблону магічна кнопка (англ. *Magic pushbutton*) штрафується 3 балами.

В проєкті має бути реалізований зручний україномовний інтерфейс. Повинна бути подана коротка інформація про програму та надаватися допомога користувачу. Діалог україномовний.

Для реалізації користувацького інтерфейсу застосунку рекомендовано використання MAUI.

Під "аналізом та обчисленням" тут розуміється клас задач, що часто зустрічаються в системному програмуванні у процесі обробки інформації операційними системами, макроасемблерами, асемблерами, трансляторами. Усі вони передбачають завдання вхідної інформації (виразів) із використанням формалізованих описів (БНФ, діаграм Вірта та ін.). Перший етап визначає синтаксичну правильність виразів, подальші – перетворення формату даних, заключний – обчислення значення виразу.

Варіант роботи видається викладачем.

Реалізувати проєкт з роботи з електронними таблицями, надати можливість їх введення, збереження, обробки, редагування кількості рядків та стовпчиків таблиці.

Клітини електронної таблиці містять вирази, що складаються із знаків операцій, констант та посилань на інші клітинки таблиці. Синтаксис посилання на клітину таблиці може бути запропонований виконавцем роботи за аналогією з відомими системами. Перевірити синтаксичну правильність виразу та знайти його значення (передбачити виконання двох етапів: синтаксична перевірка з локалізацією помилок та власне обчислення значення). Відображення інформації в таблиці на формі має підтримувати два режими: ВИРАЗ/ЗНАЧЕННЯ.

Для додаткових двох балів замість виразів у клітинках таблиці - оператори спрощеної мови програмування.

Вважати, що при побудові виразів використовуються:

1) цілі числа довільної довжини;
2) круглі дужки;
3) імена клітинок (наприклад, A3);
4) операції та функції, які для кожного із варіантів лабораторної роботи визначаються окремо.

Символи пропуску у виразах використовуються звичайним чином.

Перелік варіантів операцій та функцій:

1)	+, -, *, / (бінарні операції);	7)	mmax(x1,x2,...,xN), mmin(x1,x2,...,xN) (N>=1);
2)	mod, div;	8)	=, <, >;
3)	+, - (унарні операції);	9)	<=, >=, <>;
4)	^ (піднесення у степінь);	10)	not;
5)	inc, dec;	11)	or, and;
6)	max(x,y), min(x,y);	12)	eqv.

У варіантах 1-19 вираз вважати арифметичним (значення такого виразу повинно бути числом), а у варіантах 20-81 вираз вважати логічним.

Варіанти лабораторних робіт (за наборами операцій та функцій):

1) 1,2,3,4;	21) 1,2,3,8,10;	41) 1,3,8,10,11;	61) 2,4,9,10,11;
2) 1,2,3,5;	22) 1,2,4,8,9;	42) 1,4,5,8,9;	62) 2,5,8,10,11;
3) 1,2,3,6;	23) 1,2,4,8,10;	43) 1,4,5,8,10;	63) 2,6,9,10,11;
4) 1,2,3,7;	24) 1,2,5,8,9;	44) 1,4,6,8,9;	64) 2,7,8,10,11;
5) 1,2,4,5;	25) 1,2,5,9,10;	45) 1,4,6,8,10;	65) 2,9,10,11,12;
6) 1,2,4,6;	26) 1,2,6,8,9;	46) 1,4,7,8,9;	66) 3,4,8,10,11;
7) 1,2,4,7;	27) 1,2,6,8,10;	47) 1,4,7,8,10;	67) 3,5,9,10,11;

8) 1,3,4,5;	28) 1,2,7,8,9;	48) 1,4,8,9,10;	68) 3,7,9,10,11;
9) 1,3,4,6;	29) 1,2,7,9,10;	49) 1,4,8,10,11;	69) 3,8,10,11,12;
10) 1,3,4,7;	30) 1,2,8,9,10;	50) 1,5,6,8,10;	70) 4,5,9,10,11;
11) 1,4,5,6;	31) 1,2,8,10,11;	51) 1,5,7,8,10;	71) 4,6,8,10,11;
12) 1,4,5,7;	32) 1,3,4,8,9;	52) 1,5,8,10,11;	72) 4,7,8,10,11;
13) 1,5,6,7;	33) 1,3,4,9,10;	53) 1,6,7,8,10;	73) 4,8,10,11,12;
14) 2,3,4,5;	34) 1,3,5,8,9;	54) 1,6,8,10,11;	74) 5,6,7,8,9;
15) 2,3,4,6;	35) 1,3,5,8,10;	55) 1,7,8,9,10;	75) 5,6,8,10,11;
16) 2,3,4,7;	36) 1,3,6,8,9;	56) 1,7,8,10,11;	76) 5,7,8,10,11;
17) 3,4,5,6;	37) 1,3,6,9,10;	57) 1,8,9,10,11;	77) 5,8,10,11,12;
18) 3,4,5,7;	38) 1,3,7,8,9;	58) 1,8,10,11,12;	78) 6,7,8,10,11;
19) 3,4,6,7;	39) 1,3,7,8,10;	59) 2,3,4,8,10;	79) 6,8,10,11,12;
20) 1,2,3,8,9;	40) 1,3,8,9,10;	60) 2,3,8,10,11;	80) 7,8,10,11,12;
			81) 8,9,10,11,12.

Лабораторна робота № 1*

РОЗРОБКА ПРОГРАМИ ДЛЯ РОБОТИ З ЕЛЕКТРОННИМИ ТАБЛИЦЯМИ (АНАЛІЗ ТА ОБЧИСЛЕННЯ ВИРАЗІВ), ЩО ВКЛЮЧАЄ РОБОТУ З GOOGLE DRIVE

Використання антишаблону магічна кнопка (англ. *Magic pushbutton*) штрафується 3 балами.

Умова лабораторної роботи №1 розширюється необхідністю реалізації збереження та зчитування файлів з таблицями (за бажанням можна використовувати Google-таблиці, або власний формат), що розміщені на Google Drive.

Лабораторна робота № 2
РОБОТА З ФАЙЛАМИ XML

Використання антишаблону магічна кнопка (англ. *Magic pushbutton*) чи невикористання шаблонів штрафується 3 балами.

Для реалізації користувацького інтерфейсу застосунку рекомендовано використання MAUI.

При реалізації лабораторної роботи **ОБОВ'ЯЗКОВИМ** є використання шаблонів проєктування. Наприклад, використання шаблону «Стратегія» Strategy.

Є XML-файл на **БУДЬ-ЯКУ тематику (рекомендовані тематики наведено нижче в оопису умови лабораторної роботи)** за власним бажанням. Необхідно забезпечити обробку цих документів.

Вимоги до вхідного файлу XML:

1) кількість атрибутів у головного вузла (по якому здійснюється пошук) повинна бути різна у різних таких вузлів;

2) XML файл має мати не менше ніж 2-рівневу структуру.

Обробка включає в себе дві задачі:

1) аналіз вмісту документу (пошук інформації за ключовими словами та динамічну генерацію запитів);

2) трансформацію у файл HTML.

Вхідні дані для аналізу та трансформації надаються у вигляді *файлу-прикладу *.xml*. Трансформація документу в HTML файл виконується на основі *XSL-документа *.xsl*.

Аналіз вмісту документа повинен бути виконаний трьома способами:

1) за допомогою SAX API;

2) за допомогою DOM API;

3) за допомогою LINQ to XML.

Необхідно реалізувати всі три способи і при цьому використати шаблон «Стратегія» Strategy.

Вимоги до інтерфейсу користувача:

Для реалізації користувацького інтерфейсу застосунку рекомендовано використання MAUI.

1) повинна бути оброблена «підгрузка даних», тобто при виборі файла з яким програма працює пошук *хоча б одного атрибута* має здійснюватися за вибором користувача серед **тих значень атрибутів які присутні в файлі**.
2) повинно бути підтвердження при виході з програми («Чи дійсно ви хочете завершити роботу з програмою?» – «Так», «Ні»);
3) повинна бути кнопка Clear яка очищує поле виводу і задані параметри пошуку.

Обробка включає в себе дві задачі: аналіз вмісту документу (пошук інформації за ключовими словами та динамічну генерацію запитів) та трансформацію у файл HTML. Вхідні дані для аналізу та трансформації надаються у вигляді файлу-прикладу *.xml. Трансформація документу в HTML-код виконується на основі XSL-документа *.xsl. Аналіз вмісту документа повинен бути виконаний трьома способами – за допомогою SAX API, DOM API та LINQ to XML. Необхідно реалізувати всі три способи!

Варіанти інформаційних систем, які надають вхідний xml-файл (саму ІС на цьому етапі реалізовувати не потрібно).

1. "Наукові роботи кафедри"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення) (департамент, відділення), кафедра, лабораторія, посада (із зазначенням початку та кінця перебування на посаді), назва виконуваної наукової чи дослідної роботи, замовник, його адреса, підпорядкування, галузь та ін.

2. "Кафедральна бібліотека"

Таблиці містять таку інформацію: П.І.П. автора, назва, анотація, кваліфікаційні ознаки видання, для читачів - П.І.П., факультет, кафедра, посада та ін.

3. "Гуртожиток"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра , курс, дані про місце проживання (із датами)та ін.

4. "Електронний архів факультету"

Таблиці містять таку інформацію: П.І.П. автора, назва матеріалу, факультет (департамент, відділення), кафедра, вид матеріалу, обсяг, дата створення та ін.

5. "Розклад"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра викладача, дані про аудиторії, учбовий план та склад студентів.

6. "Успішність студентів"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, дані про навчальні дисципліни та успішність студентів та ін.

7."Інформаційні ресурси кафедри"

Таблиці БД містять таку інформацію: Назва ресурсу (ресурс - будь-яка інформація навчального та наукового спрямування), анотація, вид, автор (П.І.П., факультет (департамент, відділення), кафедра….), умови використання, адреса та ін.

8. "Розклад роботи дисплейних класів факультету та АРМ кафедр"

Таблиці БД містять таку інформацію: Інформація про класи та окремі робочі місця, інформація про користувачів, дані про розклад планових занять та викладачів, час вільного доступу

9. "Програмне забезпечення на мережі факультету"

Таблиці БД містять таку інформацію: Назва програмного засобу, анотація, вид, версія, автор (….), умови використання, де записано дистрибутив

10. "Інформаційні сервіси мережі факультету "

Таблиці БД містять таку інформацію: Назва сервісу, анотація, вид, версія, автор (….), умови та првила використання, інформація при реєстрації користувачів

11. "Мережна газета факультету"

Таблиці БД містять таку інформацію: Назва статті чи графічного матеріалу , анотація, автор (….), де записано файли, інформація про читачів та їх відгуки

12. "Індивідуальна робота кафедри зі студентами"

Таблиці БД містять таку інформацію: Інформація про студентів та викладачів (…), теми та графік роботи, виконання завдань, допоміжні інформаційні матеріали до тем, запитання-відповіді

13. " Спорт на факультеті"

Таблиці БД містять таку інформацію: Інформація про студентів та викладачів (….), що займаються у спортивних секціях, графік роботи секцій, план проведення змагань

14. "Кадри науковців (Аналіз штатного розпису)"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення) (департамент, відділення), кафедра, лабораторія, посада (із зазначенням початку та кінця перебування на посаді) та ін.

15. "Кадри (Освіта)"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, освіта (базова, додаткова, аспірантура, докторантура), навчальний заклад, де здобувалась освіта з датами, та ін.

16. "Кадри науковців (Звання)"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра , науковий ступінь, вчене звання (із датами).

17. "Кадри науковців (Пенсія)"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, дата народження, стать та ін.

18. "Кадри науковців (Публікації)"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, дані про публікації.

19. "Кадри науковців (Зарплата)"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, посада, посадовий оклад, час перебування на посаді.

20. "Конференції"

Таблиці містять інформацію про наукові конференції: назва, терміни проведення, тематика (для якого підрозділу цікаво), організатор, місце проведення, термін подачі рукописів, вартість участі та ін.

21. "Аспірантура"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, фах, форма навчання, початок навчання, графік іспитів та заліків, форми та терміни звітності на кафедрі та ін. Використати допоміжні таблиці (класифікатори, довідники, словники та ін.)

22. "Студентські гуртки та семінарами"

Таблиці містять таку інформацію: Назва гуртка/ семінару, факультет (департамент, відділення), кафедра, день та час проведення засідань, староста, орієнтація (на який курс та яку тематику), П.І.П. керівника.

23. "Кадри науковців"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, посада, посадовий оклад. прийом та звільнення, переміщення з посади на посаду, зміни посадового окладу та ін.

24. "Облік випускників"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, спеціальність, час вступу та закінчення, переміщення з посади на посаду та з одного місця роботи на інше.

25. "Міжнародні контакти"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, спеціальність, часові рамки виду співробітництва (перебування у науковому відрядженні, читання лекцій, передача наукової інформації за кордон, одержання інформації із-за кордону, проведення семінару.

26. "День факультету"

Таблиці містять таку інформацію: назва заходу на день факультету, П.І.П., факультет (департамент, відділення), кафедра, посада відповідального за проведення, виконавців окремих доручень щодо підготовки та проведення дня факультету, терміни підготовки, час, місце проведення, неформальна характеристика та ін. на розсуд виконавця лабораторної роботи.

27. "Наукове товариство студентів та аспірантів "

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, спеціальність, час вступу до товариства його членів, план проведення заходів (що, де, коли, неформальна характеристика) на інше.

28. "Студентський парламент"

Таблиці містять таку інформацію: П.І.П., факультет (департамент, відділення), кафедра, спеціальність, часові рамки виду заходу, що проводить студентський парламент.

Лабораторна робота № 2*

РОБОТА З ФАЙЛАМИ XML, ЩО ВКЛЮЧАЄ РОБОТУ З GOOGLE DRIVE

Умова лабораторної роботи №2 розширюється необхідністю реалізації збереження та зчитування XML та XSLT-файлів, що розміщені на Google Drive. В тому числі збереження відфільтрованих даних у форматі XML та HTML на Google Drive.

Виконати такі завдання (відповідно до варіанту).

1. Застосування шаблону «Фабричний метод» (Factory Method) для збереження відфільтрованого фрагмента на Google Drive в різних форматах: html, .xml. Створіть базовий клас, де буде код відкритого файлу і фабричний метод для створення конкретного об’єкту, який буде зберігати дані в цей файл.

2. Використовуючи шаблон «Одинак» (Singleton), розробіть систему протоколювання подій програми. Протокол зберігати на локальному диску. Система повинна:

- підтримувати 3 рівня важливості подій (фільтрація, трансформація, збереження);
- забезпечити фіксацію події (з подією фіксуються час, важливість, текстове повідомлення).

Наприклад:

21.09.2023 13:02:22 Фільтрація. Параметри: «Name» – aaa, «DateTime» – 01/01/2020-01/01/2023

21.09.2023 14:02:10 Трансформація. Збережено у файл aaa.html

21.09.2022 14:02:21. Збереження. Збережено відфільтрований фрагмент у файл aaa.xml

Лабораторна робота № 3

РОЗРОБКА ПРОГРАМ ОБРОБКИ ФАЙЛІВ У ФОРМАТІ JSON (ДИСПЕТЧЕР ДЛЯ JSON)

Для предметної області, яка залежить від варіанту завдання, реалізувати проєкт для роботи з даними поданими у форматі JSON.

Проєкт має містити:

- відкриття файлу типу JSON (через диспетчер);
- зчитування інформації з файла;
- візуалізацію списку;
- додавання / редагування / видалення даних до списку (додавання / редагування через окрему форму);
- збереження до / вивантаження з файлу (незмінного);
- серіалізацію / десеріалізацію даних формата JSON (відповідно до варіанту);
- мінімум 3 критерії пошуку по списку з використанням LINQ (визначається на етапі створення діаграм);
- форму "Про програму" (ПІБ, курс, група, рік, короткий опис).
- десеріалізація даних;
- використання Grid (таблиці) для роботи з даними і подальшим їх відображенням.

Проєкт повинен бути реалізовано за шаблоном MAUI (або Windows Forms Application чи WEB-проєкт за попереднім узгодженням з викладачем).

ТЕМАТИЧНИЙ ПЛАН ЛЕКЦІЙ, ЛАБОРАТОРНИХ ЗАНЯТЬ ТА САМОСТІЙНОЇ РОБОТИ

ЧАСТИНА 1. ОБ'ЄКТНО-ОРІЄНТОВАНА МЕТОДОЛОГІЯ

ТИЖДЕНЬ 1

Тема 1. ОСНОВНІ ПОНЯТТЯ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Огляд, мета і призначення теми: огляд основних парадигм програмування, вивчення основних понять об'єктно-орієнтованого програмування.

Вивчивши матеріал даної теми, студент зможе: знати основні парадигми програмування, охарактеризувати основні етапи життєвого циклу ПС, знати та уміти будувати діаграми UML (діаграму класів, прецедентів, послідовності використання).

Лабораторний практикум 1

Організаційні моменти (контактні дані викладачів, умови проведення занять, схема формування оцінки, умови лабораторних робіт, розподіл варіантів лабораторних робіт, тест-знайомство (0 балів)) (30 хв.). Призначення та використання мови UML. Опрацювання побудови діаграми прецедентів (Use-Case Diagram) та діаграми класів (Class Diagram) (60 хв.).

Завдання для опрацювання матеріалу

Завдання 1.1. Побудуйте діаграму прецедентів (Use-case-діаграму) для адміністратора системи продажу квитків на потяги.

Завдання 1.2. Побудуйте діаграму прецедентів (Use-case-діаграму) для користувача з роллю пасажир сервісу продажу квитків на потяги.

Завдання 1.3. Розробити діаграму класів (Class) для системи продажу квитків на потяги.

Завдання 2.1. Побудуйте діаграму прецедентів (Use-case-діаграму) для адміністратора системи поліклініки.

Завдання 2.2. Побудуйте діаграму прецедентів (Use-case-діаграму) для лікаря системи поліклініки.

Завдання 2.3. Розробити діаграму класів (Class) для системи поліклініки.

Завдання 3.1. Побудуйте діаграму прецедентів (Use-case-діаграму) для викладача системи online навчання.

Завдання 3.2. Побудуйте діаграму прецедентів (Use-case-діаграму) для студента системи online навчання.

Завдання 3.3. Розробити діаграму класів (Class) для системи online навчання.

Завдання 4.1. Побудуйте діаграму прецедентів (Use-case-діаграму) для адміністратора сервісу придбання квитків в кінотеатр.

Завдання 4.2. Побудуйте діаграму прецедентів (Use-case-діаграму) для користувача з роллю глядач сервісу придбання квитків в кінотеатр.

Завдання 4.3. Розробити діаграму класів (Class) для системи «кінотеатр».

Завдання 5.1. Побудуйте діаграму прецедентів (Use-case-діаграму) для індивідуального варіанту лабораторної роботи №1.

Завдання 5.2. Побудуйте діаграму класів для індивідуального варіанту лабораторної роботи №1.

Запитання до теми:

1. Які моделі життєвого циклу Ви знаєте?
2. Що таке парадигма програмування?
3. Які парадигми програмування Ви знаєте?
4. Що таке об’єктно-орієнтоване програмування?
5. Які основні елементи об’єктної моделі Ви знаєте?
6. Що таке абстрагування?
7. Що таке інкапсуляція?
8. Що спільного та чим відрізняються абстракція і інкапсуляція?
9. Що таке модульність?
10. Що таке ієрархічність?
11. Які види ієрархічності Ви знаєте?
12. Що таке типізація?
13. Що таке паралелізм?
14. Що таке збережуваність?
15. Що таке інкапсуляція?
16. Наведіть визначення та опишіть зв’язок між поняттями класу, об’єкту та екземпляру класу.

17. Що таке спадкування (наслідування)?
18. Що таке поліморфізм?

Завдання для самостійної роботи. Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторного заняття. Практичне застосування UML для об'єктно-орієнтованого моделювання ПС.

Побудова діаграм UML (прецедентів та класів) для лабораторної роботи №1. Виконання домашнього завдання №1.

Ознайомлення з умовою власного проєкту лабораторного практикуму № 1. Для виконання лабораторних робіт та опрацювання лекційного матеріалу з курсу "Об'єктно-орієнтоване програмування" необхідно встановити Visual Studio 2022: https://visualstudio.microsoft.com/downloads/

При встановленні обов'язковими є наступні компоненти:

- ASP.NET and web development (для другого семестра та домашнього завдання 3).
- NET Multy-platform App UI development (для виконання лабораторних робіт першого семестра та домашнього завдання 3).
- .NET desktop development (для виконання домашнього завдання 3).
- Data storage and processing (для другого семестра).

ТИЖДЕНЬ 2
Тема 2. ОСНОВНІ ПРИНЦИПИ МОДЕЛЮВАННЯ ПРОГРАМНИХ СИСТЕМ. ОГЛЯД МОВИ UML

Огляд, мета і призначення теми: продовження огляду основних понять об'єктно-орієнтованого програмування, огляд основних етапів та моделей життєвого циклу програмних систем (ПС), розуміння місця мови UML в життєвому циклі ПС, огляд основних видів діаграм UML.

Вивчивши матеріал даної теми, студент зможе: охарактеризувати основні етапи життєвого циклу ПС, охарактеризувати основні моделі ЖЦ ПС, знати та уміти будувати діаграми UML (діаграму класів, прецедентів, послідовності використання).

Лабораторний практикум 2

Опрацювання побудови діаграми класів (Class Diagram) та діаграми послідовності (Sequence diagram) (60 хв.). Захист домашнього завдання №1 (30 хв.)

Опрацювання лекційного матеріалу та власного проєкту лабораторного практикуму № 1.

Завдання для опрацювання матеріалу

Завдання 1.1. Побудуйте діаграму послідовності (Sequence) для прецеденту: Актор "Пасажир" має два квитки на потяг №772, що їде з Хмельницького до Києва і здає їх.

Завдання 1.2. Доопрацювати з урахуванням розробленої Sequuence-діаграми діаграму класів (Class) для системи продажу квитків на потяги.

Завдання 2.1. Побудуйте діаграму послідовності (Sequence) для прецеденту: Актор "Пацієнт" записується до лікаря-терапевта Іванова І.І. на 10:00, 10 жовтня 2023 року.

Завдання 2.2. Доопрацювати з урахуванням розробленої Sequuence-діаграми діаграму класів (Class) для системи поліклініки.

Завдання 3.1. Побудуйте діаграму послідовності (Sequence) для прецеденту: Актор "Студент Іваненко І." авторизується як студент, реєструється на курс "ООП" де є один матеріал (завдання) і прикріплює відповідь, що складається з одного файлу "a.pdf".

Завдання 3.2. Доопрацювати з урахуванням розробленої Sequuence-діаграми діаграму класів (Class) для системи online навчання.

Завдання 4.1. Побудуйте діаграму послідовності (Sequence) для прецеденту: Актор "Адміністратор" додає інформацію про фільм "Тенет", для цього фільму він додає два сеанси 20.09.2023 о 15:00 та о 19:00 в залі №1.

Завдання 4.2. Доопрацювати з урахуванням розробленої Sequence-діаграми діаграму класів (Class) для системи «Кінотеатр».

Завдання 5.1. Побудуйте діаграму послідовності використання (Sequence-діаграму) для індивідуального варіанту лабораторної роботи №1.

Завдання 5.2. Доопрацювати з урахуванням розробленої Sequuence-діаграми діаграму класів (Class) для індивідуального варіанту лабораторної роботи №1.

Запитання до теми:

1. Перелічити і охарактеризувати основні види діаграм UML.
2. Що таке діаграми станів і діяльності та яке їхнє призначення?
3. Що таке діаграми компонентів і розгортання та яке їхнє призначення?
4. Опишіть спрощену стратегію використання UML-діаграм при моделюванні ПС.
5. Які моделі життєвого циклу Ви знаєте?
6. Що таке сутності й відношення UML?
7. Що таке прикордонні й керівні класи, класи-сутності, абстрактні й конкретні класи?
8. Охарактеризувати відношення узагальнення й залежності між класами.
9. Охарактеризувати відношення асоціації, агрегації й композиції між класами.
10. Що таке діаграми прецедентів і для чого вони використовуються?
11. Які відношення між акторами й прецедентами Ви знаєте?
12. Які відношення залежності між прецедентами Ви знаєте?

Завдання для самостійної роботи. Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів

лабораторного заняття. Практичне застосування UML для об’єктно-орієнтованого моделювання програмних систем.

Побудова діаграм UML (класів та послідовності) для лабораторної роботи №1. Виконання домашнього завдання №2.

ТИЖДЕНЬ 3
Тема 3. ОСНОВНІ ПАРАДИГМИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Огляд, мета і призначення теми: огляд основних парадигм об'єктно-орієнтованого програмування. Огляд основних принципів моделювання та його місце в життєвому циклі розробки програмних систем. Використання мови UML, огляд основних видів діаграм UML.

Вивчивши матеріал даної теми, студент зможе: охарактеризувати основні етапи життєвого циклу ПС, охарактеризувати основні моделі ЖЦ ПС, знати та уміти будувати діаграми UML (діаграму класів, прецедентів, послідовності використання). Створити та запустити проєкт в Visual Studio.

Лабораторний практикум 3

Знайомство з інтегрованим середовищем розробки Visual Studio. Перший проєкт в Visual Studio (45 хв.).

Захист домашнього завдання №2 (45 хв.)

Опрацювання лекційного матеріалу і умови лабораторної роботи № 1 та 1*.

Завдання для опрацювання матеріалу

Завдання 1.1 Створіть проєкти, які виводять текст «Hello World!» за шаблонами Console Application, Windows Forms Application, WPF Application та .Net MAUI App.

Завдання 1.2

1) Необхідно створити новий solution
2) Додати новий проєкт Console Application, в якому буде можливість вводити ціле число, після чого буде відображено текст "Ви ввели число N".
3) Додати ще один проект .Net MAUI App для Windows, в якому на форму необхідно додати поле для введення цілих чисел, та компонент кнопку, при натисненні на яку, буде відображатися повідомлення з текстом "Ви ввели число N".

Завдання 2.1. Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Необхідно створити консольний проєкт, який містить щонайменше 4 класи:

1) абстрактний клас Student (з атрибутами name та state (рядки), конструктором з параметром ім'я, що ініціалізує ім'я та порожній state, публічними методами void Relax(), Read(), Write() (до state додають підрядок "Relax", "Read", "Write" відповідно) та абстрактним методом Study).

2, 3) класи GoodStudent та BadStudent, що успадковуються від Student (з конструктором з параметром name, який викликає базовий конструктор та до state додає "good" чи "bad" відповідно, перевизначеним методом Study() що для GoodStudent викликає по черзі методи Read(), Write() Read(), Write() та Relax(), а для BadStudent методи Relax(), Relax(), Relax(), Relax(), Read()).

4) клас Group (з атрибутами назва групи (рядок) та список студентів (List<Student>), конструктор з параметром назва групи, методи AddStudent(Student st) (додавання студента), GetInfo() - видає рядок з назвою групи та перерахованими іменами студентів), GetFullInfo() - видає рядок з назвою групи, перерахованими іменами студентів та їх статусами),

Організувати можливість введення груп та студентів до груп та виведення інформації про них.

Перелік методів та класів не є вичерпним.

Завдання 2.2.

1) Створити новий проект Console Application
2) Додати в проект новий абстрактний класс Worker, в якому будуть:
   - конструктор, який приймає на вхід ім'я
   - параметри: Name, Position, WorkDay
   - методи : Call(), WriteCode(), Relax() - які відповідають за дії співробітника протягом робочого дня
   - абстрактний метод FillWorkDay();
3) Додати новий клас Developer, що є нащадком класу Worker, в якому будуть:
   - конструктор, в якому визначається параметр Position як "Розробник"
   - перевизначення метода FillWorkDay(), в якому послідовно викликаються методи: WriteCode(), Call(), Relax(), WriteCode().

4) Додати новий класс Manager, що є нащадком класу Worker, в якому будуть:
    - конструктор, в якому визначається параметр Position як "Менеджер"
    - приватна змінна типу Random
    - перевизначення метода FillWorkDay(), в якому метод Call() викликається рандомну кількість разів (від 1 до 10), потім метод Relax(), після чого знову метод Call() викликається рандомну кількість разів (від 1 до 5)
5) Додати в проєкт новий класс Team, в якому будуть:
    - конструктор, який приймає на вхід ім'я
    - приватна змінна з переліком співробітників
    - метод додавання нового співробітника у команду
    - метод виведення інформації про команду (назва команди і ПІБ співробітників з нового рядку кожне)
    - метод виведення детальної інформації про команду (назва команди і інформація про співробітників у вигляді: <Name> - <Position> - <WorkDay>)

6) Організувати можливість введення команд та співробітників до команд та виведення інформації про них

Завдання 3.

1) Створіть проект за шаблоном Console Application
2) Додайте новий клас Converter, який містить:
    - параметри, що відповідають курсу долара та євро по відношенню до гривні
    - конструктор, який приймає на вхід 2 аргументи типу decimal, та ініціалізують параметри
3) Реалізуйте програму, яка буде виконувати конвертацію з гривні в одну з зазначених валют та навпаки

Завдання 4. Створити консольний додаток TestPolymorphism і клас TestOverload з трьома методами DisplayOverload. Запустіть його. Який результат?

```
class TestOverload
{
    1 reference
    public void DisplayOverload(int a)
    {
        Console.WriteLine("DisplayOverload " + a);
    }
    1 reference
    public void DisplayOverload(string a)
    {
        Console.WriteLine("DisplayOverload " + a);
    }
    1 reference
    public void DisplayOverload(string a, int b)
    {
        Console.WriteLine("DisplayOverload " + a + b);
    }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        TestOverload to = new TestOverload();

        to.DisplayOverload(100);
        to.DisplayOverload("aaa");
        to.DisplayOverload("bbb", 2);
        Console.ReadKey();
    }
}
```

Завдання 5. Створити консольний додаток TestPolymorphism і клас TestOverload з двома методами DisplayOverload. Запустіть його. Який результат?

```
class TestOverload
{

    2 references
    public void DisplayOverload()
    {
        Console.WriteLine("DisplayOverload void");
    }
    2 references
    public int DisplayOverload()
    {
        Console.WriteLine("DisplayOverload int");
        return 0;
    }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        TestOverload to = new TestOverload();

        to.DisplayOverload();
        int i = to.DisplayOverload();
        Console.ReadKey();
    }
}
```

Завдання 6. Створити консольний додаток TestPolymorphism і клас TestOverload з двома методами DisplayOverload. Запустіть його. Який результат?

```
3 references
class TestOverload
{
    2 references
    static public void DisplayOverload(int i)
    {
        Console.WriteLine("DisplayOverload static int");
    }
    2 references
    public void DisplayOverload(int i)
    {
        Console.WriteLine("DisplayOverload void int");
    }
    1 reference
    public void DisplayOverload(string a)
    {
        Console.WriteLine("DisplayOverload void string");
    }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {

        TestOverload to = new TestOverload();
        TestOverload.DisplayOverload(1);
        to.DisplayOverload(1);
        to.DisplayOverload("aaa");
        Console.ReadKey();
    }
}
```

Завдання 7. Створити консольний додаток TestPolymorphism і клас TestOverload з методами DisplayOverload. Запустіть його. Який результат?

```
2 references
class TestOverload
{
    1 reference
    public void DisplayOverload()
    {
        Console.WriteLine("DisplayOverload public");
    }
    1 reference
    protected void DisplayOverload()
    {
        Console.WriteLine("DisplayOverload protected");
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {

        TestOverload to = new TestOverload();
        to.DisplayOverload();
        Console.ReadKey();
    }
}
```

Завдання 8. Створити консольний додаток TestPolymorphism і клас TestOverload з методами DisplayOverload. Запустіть його. Який результат?

```
2 references
class TestOverload
{
    1 reference
    public void DisplayOverload(int a, string a)
    {
        double a = 1.1;
        Console.WriteLine("DisplayOverload int a string a");
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {

        TestOverload to = new TestOverload();
        to.DisplayOverload(1, "a");
        Console.ReadKey();
    }
}
```

Завдання 9. Які висновки можна зробити? Оберіть усі правильні відповіді:

- ☐ Метод ідентифікується не тільки за іменем, але і за його параметрами.
- ☐ Метод не ідентифікується за результатом.
- ☐ Модифікатори на кшталт static не є властивостями, які ідентифікують метод.
- ☐ На ідентифікатор методу впливають тільки його ім'я і параметри (їх тип, кількість). Модифікатори доступу не впливають.
- ☐ На ідентифікатор методу впливають його ім'я, параметри (їх тип, кількість) та модифікатори доступу.
- ☐ Метод ідентифікується за результатом.
- ☐ Метод ідентифікується не тільки по імені та модифікатору доступу, але не ідентифікується за його параметрами.
- ☐ Імена параметрів повинні бути унікальні.
- ☐ Не можуть бути однаковими ім'я параметра методу та ім'я змінної, створеної в цьому ж методі.
- ☐ Ім'я параметра методу та ім'я змінної, створеної в цьому ж методі можуть бути однаковими.
- ☐ Імена параметрів методу можуть повторюватися.

Завдання на опрацювання спадкування

Завдання 10. Створити консольний додаток і додайте два класи, з назвами ClassA і ClassB, як показано нижче. Запустіть його. Який результат?

```
2 references
class ClassA
{

}

0 references
class ClassB
{
    public string str = "aaa";
    0 references
    public void Method1()
    {
        Console.WriteLine("ClassB Method1");
    }
    0 references
    public void Method2()
    {
        Console.WriteLine("ClassB Method2");
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {

        ClassA a = new ClassA();
        a.Method1();
        Console.ReadKey();
    }
}
```

Завдання 11. Створити консольний додаток і класи. Зверніть увагу на зміну в першому рядку. Запустіть його. Який результат?

```
2 references
class ClassA : ClassB
{

}

1 reference
class ClassB
{
    public string str = "aaa";
    1 reference
    public void Method1()
    {
        Console.WriteLine("ClassB Method1");
    }
    0 references
    public void Method2()
    {
        Console.WriteLine("ClassB Method2");
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {

        ClassA a = new ClassA();
        a.Method1();
        Console.ReadKey();
    }
}
```

Завдання 12. Створити консольний додаток з двома класами. Запустіть його. Зверніть увагу на новий метод в ClassA. Який результат?

```
2 references
class ClassA : ClassB
{
    1 reference
    public void Method1()
    {
        System.Console.WriteLine("ClassA Method1");
    }
}

1 reference
class ClassB
{
    public string str = "aaa";
    0 references
    public void Method1()
    {
        Console.WriteLine("ClassB Method1");
    }
    0 references
    public void Method2()
    {
        Console.WriteLine("ClassB Method2");
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {

        ClassA a = new ClassA();
        a.Method1();
        Console.ReadKey();
    }
}
```

Завдання 13. Внесіть зміни в попередній код. Запустіть його. Який результат?

```
2 references
class ClassA : ClassB
{
    1 reference
    public void Method1()
    {
        System.Console.WriteLine("ClassA Method1");
        base.Method1();
    }
}
```

Завдання 14. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
class ClassA : ClassB
{
    0 references
    public void Method1()
    {
        System.Console.WriteLine("ClassA Method1");
    }
}

3 references
class ClassB
{
    public string str = "aaa";

    0 references
    public void Method2()
    {
        Console.WriteLine("ClassB Method2");
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {

        ClassB b = new ClassB();
        b.Method1();
        Console.ReadKey();
    }
}
```

Завдання 15. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
class ClassA : ClassB, ClassC
{
    0 references
    public void Method1()
    {
        System.Console.WriteLine("ClassA Method1");
    }
}

3 references
class ClassB
{
    public string str = "aaa";

    1 reference
    public void Method2()
    {
        System.Console.WriteLine("ClassB Method2");
    }
}

1 reference
class ClassC
{

    0 references
    public void Method3()
    {
        System.Console.WriteLine("ClassC Method3");
    }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        ClassB b = new ClassB();
        b.Method2();
        Console.ReadKey();
    }
}
```

Завдання 16. Створити консольний додаток. Запустіть його. Який результат?

```
1 reference
class ClassA : ClassB
{
    0 references
    public void Method1()
    {
        System.Console.WriteLine("ClassA Method1");
    }

}

3 references
class ClassB : ClassC
{
    public string str = "aaa";

    1 reference
    public void Method2()
    {
        System.Console.WriteLine("ClassB Method2");
    }
}

1 reference
class ClassC : ClassA
{
    0 references
    public void Method3()
    {
        System.Console.WriteLine("ClassC Method3");
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        ClassB b = new ClassB();
        b.Method2();
        Console.ReadKey();
    }
}
```

Завдання 17. Створити консольний додаток. Запустіть його. Який результат?

```
3 references
class ClassA : ClassB
{
    1 reference
    public void Method1()
    {
        System.Console.WriteLine("ClassA Method1");
    }
}

3 references
class ClassB
{
    public string str = "aaa";

    0 references
    public void Method2()
    {
        System.Console.WriteLine("ClassB Method2");
    }
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        ClassB classB = new ClassB();
        ClassA classA = new ClassA();
        classB = classA;
        classA = (ClassA)classB;

        classA.Method1();
        Console.ReadKey();
    }
}
```

Завдання 18. Які висновки можна зробити? Виберіть усі правильні відповіді.

- ☐ Дочірній клас має параметри батьківського.
- ☐ Методи дочірніх класів мають пріоритет при виконанні.
- ☐ Ключове слово base може бути використано для звернення до методів класу-предку.
- ☐ Успадкування не працює в зворотному напрямку.
- ☐ Клас може мати тільки один батьківський клас, множинне спадкування класів в C # не підтримує.
- ☐ Класи не можуть успадковуватися циклічно (1-й від 2-го, 2-й від 3-го 3-й від 1-го);
- ☐ Класи можуть успадковуватися циклічно (1-й від 2-го, 2-й від 3-го 3-й від 1-го).
- ☐ Клас може мати декілька батьківських класів.
- ☐ Ключове слово base може бути використано для звернення до методів класу-нащадку.
- ☐ Методи батьківських класів мають пріоритет при виконанні.
- ☐ Ви можете присвоїти змінній батьківського типу об'єкт дочірнього, але не навпаки.
- ☐ Ви можете присвоїти змінній дочірнього типу об'єкт батьківського, але не навпаки.
- ☐ Ви можете присвоїти змінній батьківського типу об'єкт дочірнього та навпаки.

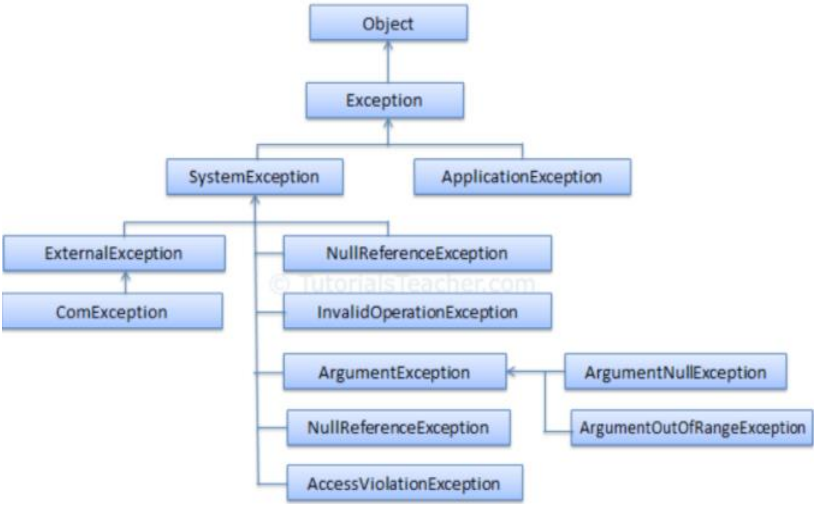
Завдання на опанування виключень в мові C#

Виняток – це будь-який стан помилки або непередбачена поведінка, що виникає при виконанні програми. Винятки можуть виникати через збій у вашому коді, або у коді, що викликається (наприклад, в загальній бібліотеці), недоступність ресурсів ОС, несподіваних станів, що виникають у середовищі виконання (наприклад, код, який неможливо перевірити) і з інших причин. Після деяких з цих станів додаток може відновитися, після інших - ні.

В .NET виняток – це об'єкт, що успадковується від класу System.Exception. Виняток створюється з області коду, де сталася проблема. Виняток передається вгору по стеку до того часу, поки його не обробить додаток або програма не завершиться.

Для кожної групи виняткових ситуацій існує свій клас, члени якого забезпечують обробку ситуацій цієї групи.

Ієрархія деяких виключень



Завдання 19. Створіть консольний додаток. Запустіть його. Який результат?

```
public static void Main()
{
    int x = 0;
    try
    {
        int y = 100 / x;
    }
    catch (ArithmeticException)
    {
        Console.WriteLine("ArithmeticException");
    }
    catch (Exception)
    {
        Console.WriteLine("Exception");
    }
    Console.ReadKey(true);
}
```

Завдання 20. Створіть консольний додаток. Запустіть його. Який результат?

```
public static void Main()
{
    int x = 0;
    try
    {
        int y = 100 / x;
    }
    catch (Exception)
    {
        Console.WriteLine("Exception");
    }
    catch (ArithmeticException)
    {
        Console.WriteLine("ArithmeticException");
    }

    Console.ReadKey(true);
}
```

Завдання 21. Створіть консольний додаток. Запустіть його. Який результат?

```
public static void Main()
{
    try
    {
        int[] numbers = new int[4];
        numbers[7] = 9;      // IndexOutOfRangeException

        int x = 5;
        int y = x / 0;  // DivideByZeroException
        Console.WriteLine($"Результат: {y}");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("DivideByZeroException");
    }
    catch (IndexOutOfRangeException)
    {
        Console.WriteLine("IndexOutOfRangeException");
    }

    Console.ReadKey(true);
}
```

Завдання 22. Створіть консольний додаток. Запустіть його. Який результат?

```
public static void Main()
{
    try
    {
        object obj = "you";
        int num = (int)obj;     // InvalidCastException
        Console.WriteLine($"Результат: {num}");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("DivideByZeroException");
    }
    catch (IndexOutOfRangeException)
    {
        Console.WriteLine("IndexOutOfRangeException");
    }
    Console.ReadKey(true);
}
```

Завдання 23. Створіть консольний додаток. Запустіть його. Який результат?

```
public static void Main()
{
    try
    {
        object obj = "you";
        int num = (int)obj;     // InvalidCastException
        Console.WriteLine($"Результат: {num}");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("DivideByZeroException");
    }
    catch (IndexOutOfRangeException)
    {
        Console.WriteLine("IndexOutOfRangeException");
    }
    catch (Exception ex)
    {
        Console.WriteLine("{0}", ex.Message);
    }

    Console.ReadKey(true);
}
```

Завдання 24. Створіть консольний додаток. Запустіть його. Який результат?

```
public static void Main()
{
    Console.OutputEncoding = Encoding.UTF8;
    try
    {
        int x = 5;
        int y = x / 0;
        Console.WriteLine("Результат: {0}", y);
    }
    catch
    {
        Console.WriteLine("Виключення!");
    }
    finally
    {
        Console.WriteLine("Блок finally");
    }
    Console.WriteLine("Завершення програми");

    Console.ReadKey(true);
}
```

Завдання 25. Створіть консольний додаток. Запустіть його. Який результат?

```
public static void Main()
{
    Console.OutputEncoding = Encoding.UTF8;
    int x = 1;
    int y = 0;
    try
    {
        int result = x / y;
    }
    catch (DivideByZeroException) when (y == 0 && x == 0)
    {
        Console.WriteLine("y та x не повинні бути 0 одночасно");
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine(ex.Message);
    }
    finally
    {
        Console.WriteLine("Блок finally");
    }
    Console.WriteLine("Завершення програми");

    Console.ReadKey(true);
}
```

Завдання 26. Створіть консольний додаток. Запустіть його. Який результат?

```
public static void Main()
{
    Console.OutputEncoding = Encoding.UTF8;
    int x = 0;
    int y = 0;
    try
    {
        int result = x / y;
    }
    catch (DivideByZeroException) when (y == 0 && x == 0)
    {
        Console.WriteLine("y та x не повинні бути 0 одночасно");
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine(ex.Message);
    }
    finally
    {
        Console.WriteLine("Блок finally");
    }
    Console.WriteLine("Завершення програми");

    Console.ReadKey(true);
}
```

Завдання 27. Створіть консольний додаток. Запустіть його. Який результат?

```
class StudentException : ArgumentException
{
    2 references
    public int Value { get; }
    1 reference
    public StudentException(string message, int val)
        : base(message)
    { Value = val; }
}
2 references
class Student
{
    1 reference
    public string Name { get; set; }
    private int score;
    1 reference
    public int Score
    {
        get { return score; }
        set
        {
            if (value < 20)
                throw new StudentException("З балом <20 недопуск до іспиту", value);
            else
                score = value;
        }
    }
}
0 references
class ExceptionTestClass
{
    0 references
    public static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        try
        {
            Student st = new Student { Name = "Ivan", Score = 10 };
        }
        catch (StudentException ex)
        {
            Console.WriteLine($"Помилка: {ex.Message}");
            Console.WriteLine($"Кількість балів: {ex.Value}");
        }
        Console.ReadKey(true);
    }
}
```

Завдання 28. Створіть консольний додаток. Запустіть його. Який результат?

```
static void Main(string[] args)
{
    Console.OutputEncoding = Encoding.UTF8;
    try
    {
        TestClass.Method1();
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine($"Catch в Main : {ex.Message}");
    }
    finally
    {
        Console.WriteLine("Блок finally в Main");
    }
    Console.WriteLine("Конец метода Main");
    Console.Read();
}

ference
ss TestClass

1 reference
public static void Method1()
{
    try
    {
        Method2();
    }
    catch (IndexOutOfRangeException ex)
    {
        Console.WriteLine($"Catch в Method1 : {ex.Message}");
    }
    finally
    {
        Console.WriteLine("Блок finally в Method1");
    }
    Console.WriteLine("Конец метода Method1");
}
1 reference
static void Method2()
{
    try
    {
        int x = 8;
        int y = x / 0;
    }
    finally
    {
        Console.WriteLine("Блок finally в Method2");
    }
    Console.WriteLine("Конец метода Method2");
}
```

Завдання 29. Зробіть висновки з виконаних завдань.

Запитання до теми:

1. Які парадигми програмування Ви знаєте? Охарактеризуйте їх.
2. Що таке декларативне програмування?
3. Що таке імперативне програмування?
4. Які парадигми імперативного програмування Ви знаєте?
5. Які парадигми об'єктно-орієнтованого програмування Ви знаєте?
6. Охарактеризуйте інкапсуляцію.
7. Охарактеризуйте спадкування.
8. Охарактеризуйте поліморфізм.
9. Що таке абстракція?
10. Що таке надсилання повідомлень в ООП?
11. Повторне використання в ООП.
12. Що таке клас в ООП?
13. Що таке об'єкт в ООП?

Завдання для самостійної роботи. Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторного заняття. Виконання домашнього завдання №3. Реалізація лабораторної роботи №1 та 1*.

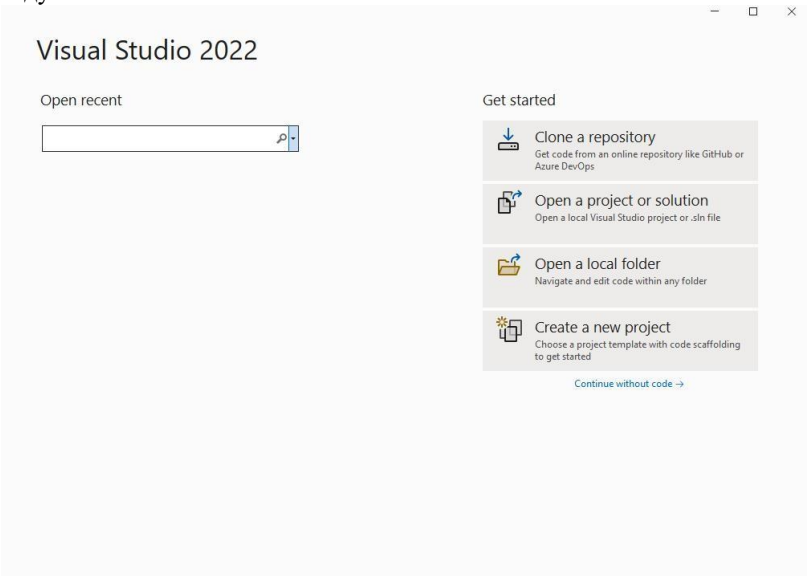
ТЕОРЕТИЧНА ЧАСТИНА ЛАБОРАТОРНОГО ЗАНЯТТЯ

ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ MS VISUAL STUDIO. ПЕРШЕ ЗНАЙОМСТВО

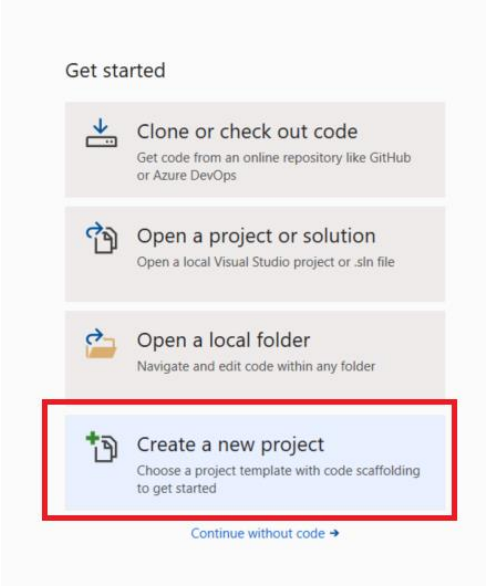
https://docs.microsoft.com/uk-ua/visualstudio/install/install-visual-studio?view=vs-2022

Завантажити Visual Studio 2022 в редакції Community можна з офіційного сайта – https://visualstudio.microsoft.com/

Після запуску Visual Studio ви перш за все побачите початкове вікно. Вікно запуску допомагає швидше дістатися до коду. Тут є дії, що дозволяють клонувати або витягти код, відкрити існуючий проєкт або рішення, створити новий проєкт або просто відкрити папку з файлами коду.

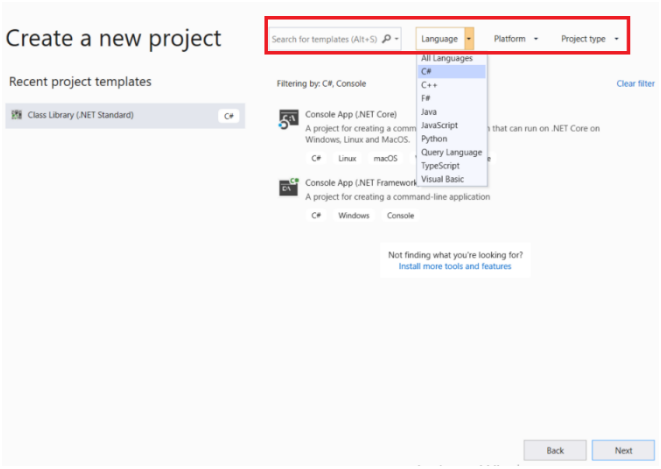


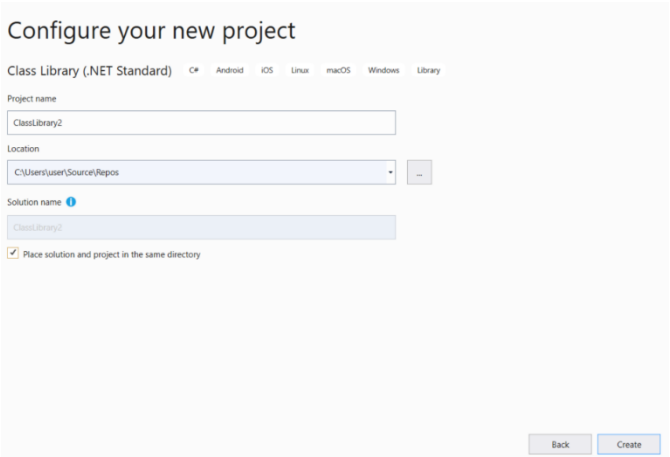
Для створення нового проєкту на початковому екрані виберіть Створити проєкт (Create anew project).



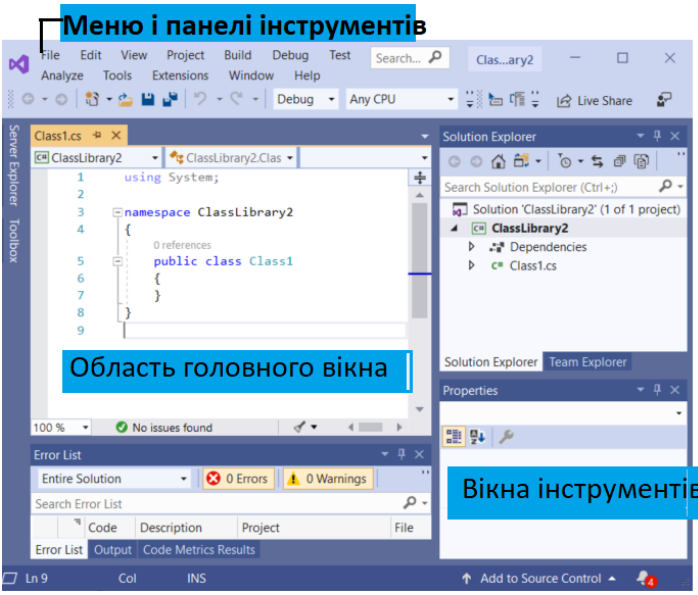
Відкривається діалогове вікно з заголовком Створення проєкту. У ньому можна виконати пошук, фільтрацію і вибір шаблону проєкту. Тут також відображається список недавно використаних шаблонів проєкту.

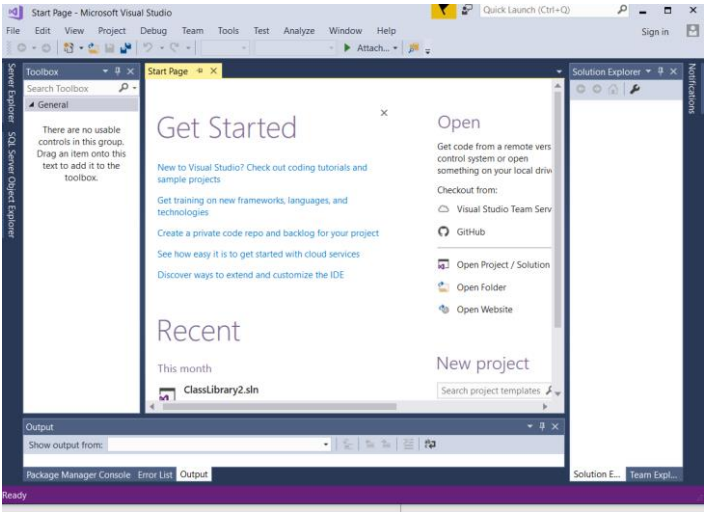
Введіть в поле пошуку вгорі рядок консоль, щоб залишити в списку лише ті типи проєктів, в імені яких є слово "консоль". Додатково уточніть результати пошуку, вибравши C # (або інший потрібну мову) із засобу вибору Мова.





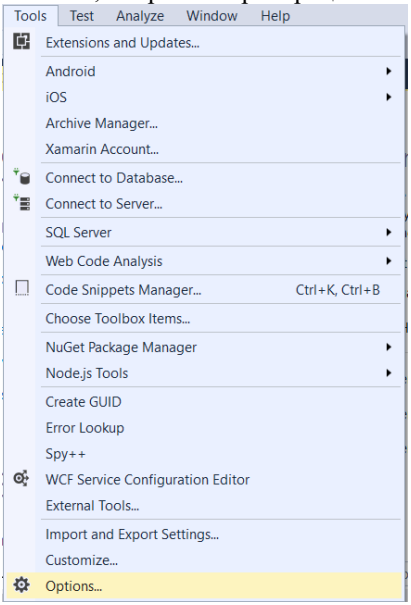
Після відкриття Visual Studio можна вказати вікна інструментів, меню і панелі інструментів і простір головного вікна. Вікна інструментів прикріплені по лівій і правій сторонах вікна програми, а у верхній його частині розташовуються панель швидкого запуску, рядок меню, а також стандартна панель інструментів. У центрі вікна програми розташовується Початкова сторінка. Коли завантажені рішення або проєкт, на її місці будуть відображатися редактори і конструктори. При створенні програми ви будете проводити більшу частину часу в цій центральній області.

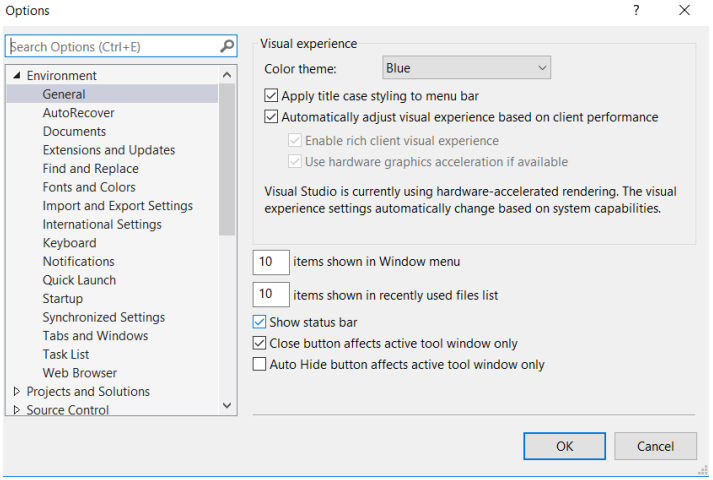




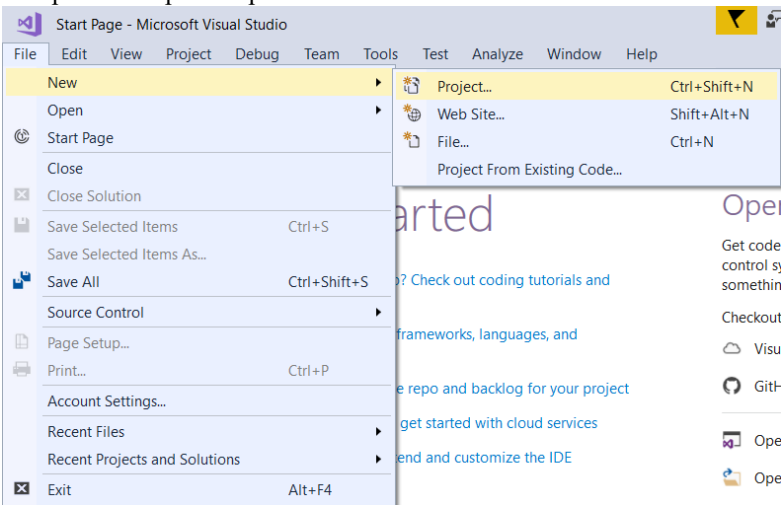
Можна зробити додаткові налаштування в Visual Studio, наприклад, змінити в редакторі накреслення і розмір шрифту тексту або за допомогою діалогового вікна Параметри колірну тему інтегрованого

середовища розробки. Залежно від застосованого поєднання параметрів, деякі елементи в цьому діалоговому вікні можуть не відображатися. Можна налаштувати, щоб всі можливі параметри відображалися, вибравши прапорець "Показати всі параметри".

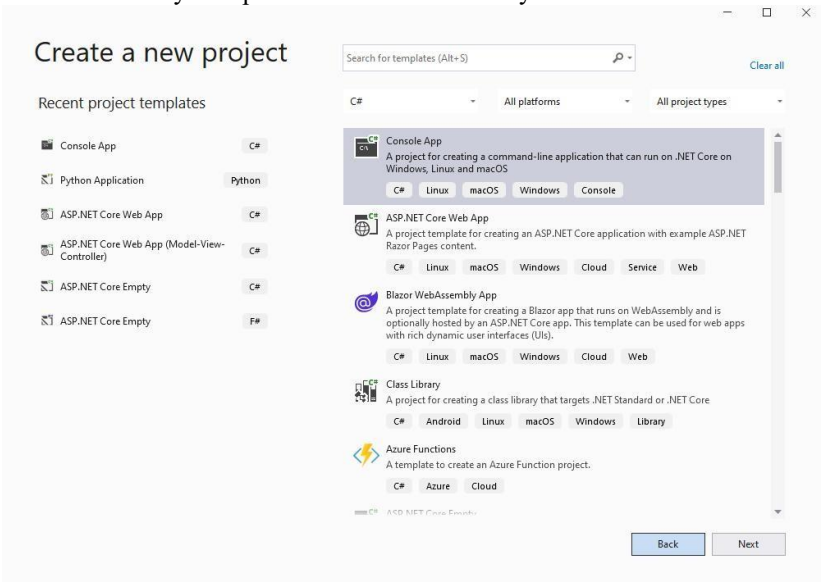


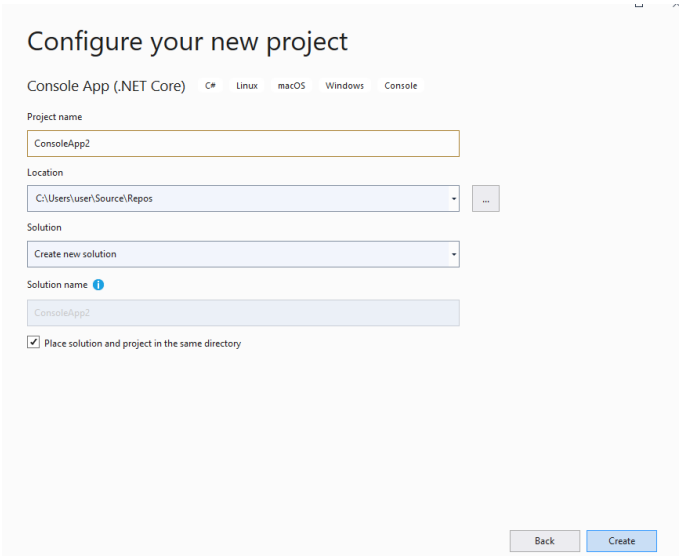


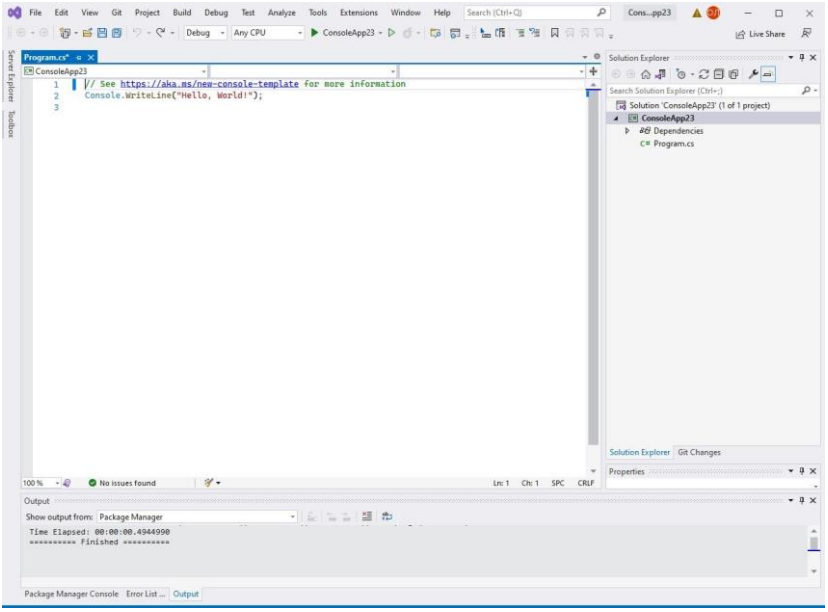
При створенні застосунку в Ms Visual Studio спочатку створюється проєкт і рішення.



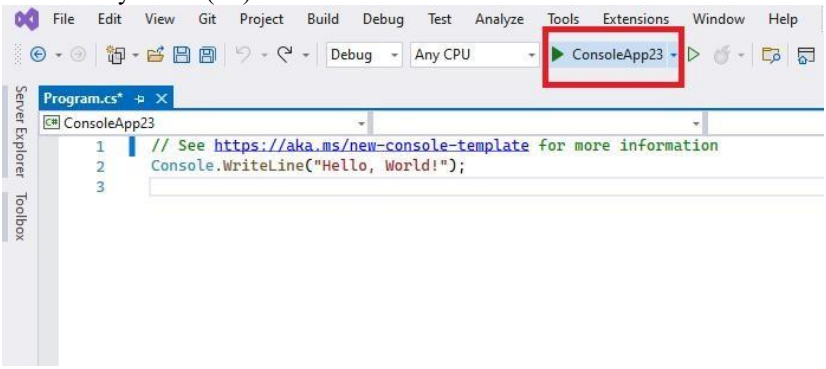
Спочатку створимо консольний застосунок:



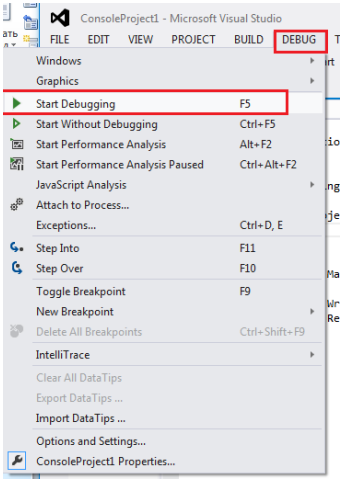

Configure your new project
Console App (.NET Core)
C#
Linux
macOS
Windows
Console
Project name
ConsoleApp2
Location
C:\Users\user\Source\Repos
Solution
Create new solution
Solution name
ConsoleApp2
Place solution and project in the same directory
Back
Create


// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
Solution Explorer
ConsoleApp23
Dependencies
Program.cs
Output
Time Elapsed: 00:00:00.4944990
========== Finished ==========

Запустимо (F5):



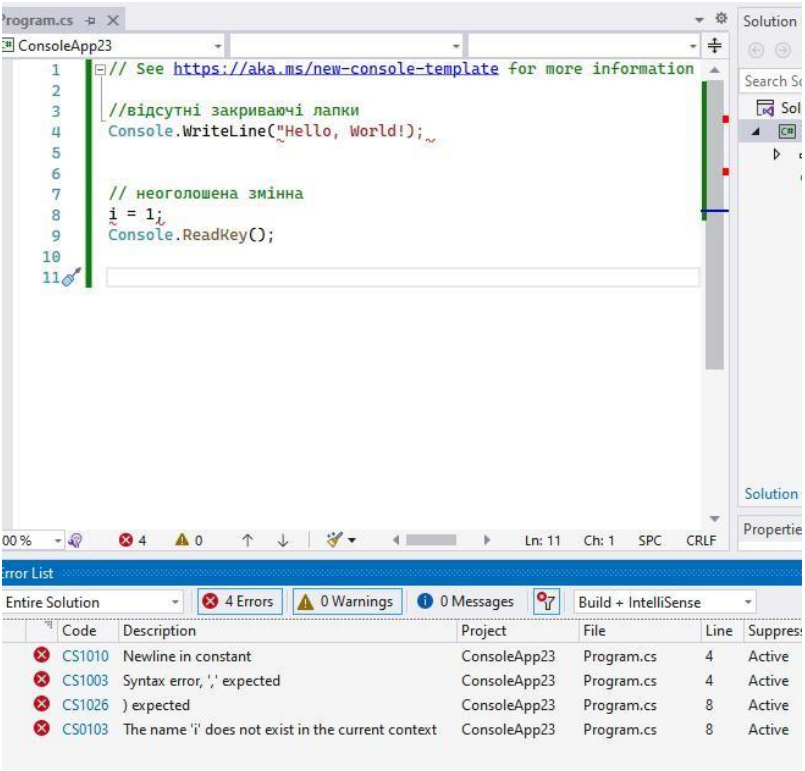
або так



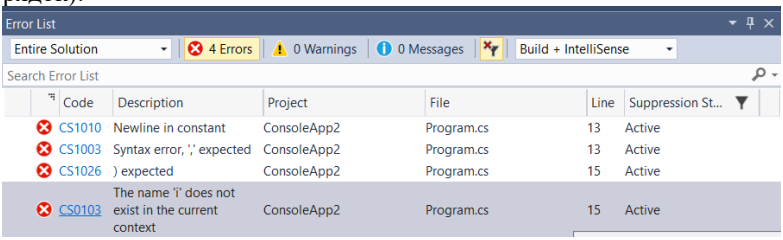
В результаті отримаємо:

```
Hello world!!!
```

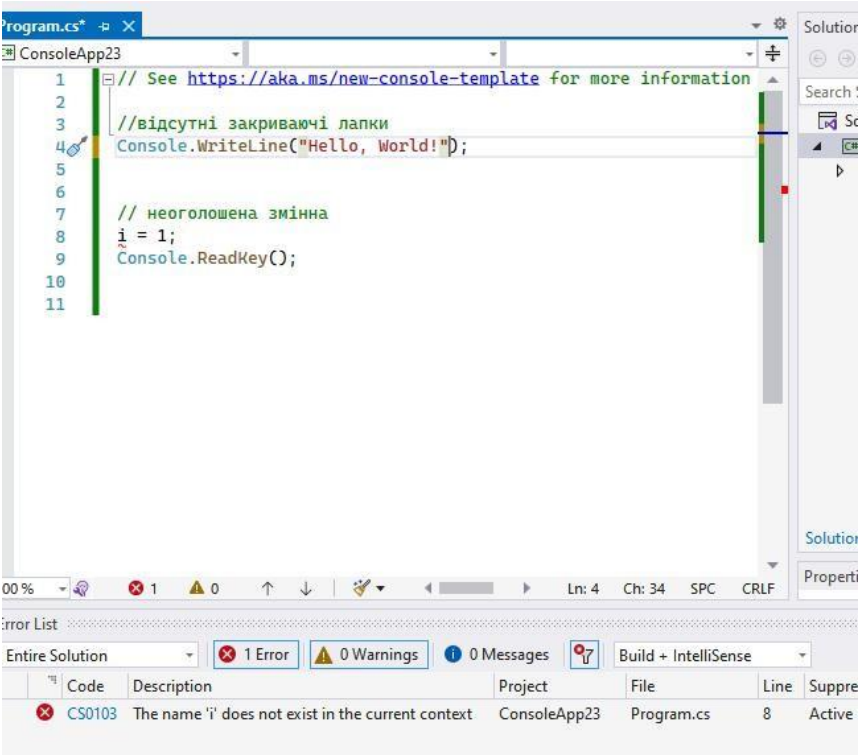
Нехай, ми допустилися помилок:



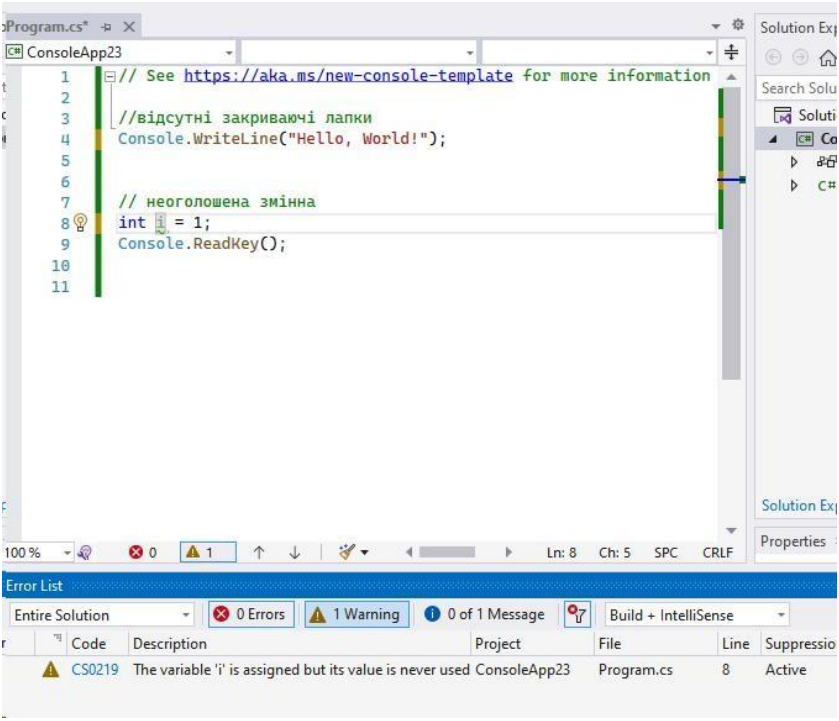
Тоді при виконання побудови, або при запуску отримаємо помилки (подвійним натисканням миші переходимо в потрібний рядок):



Виправимо першу помилку та знову перевіримо. При цьому отримаємо повідомлення про другу помилку:



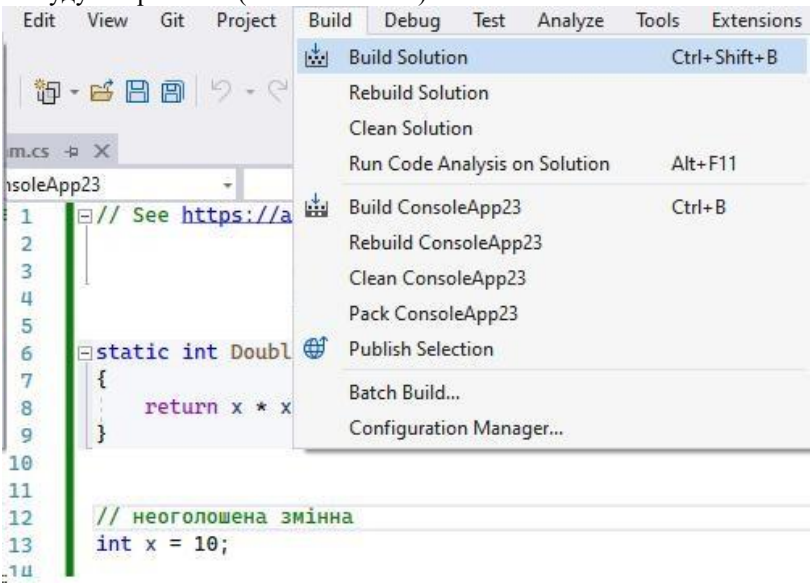
Виправимо описаним нижче чином. Отримали повідомлення про невикористану змінну:



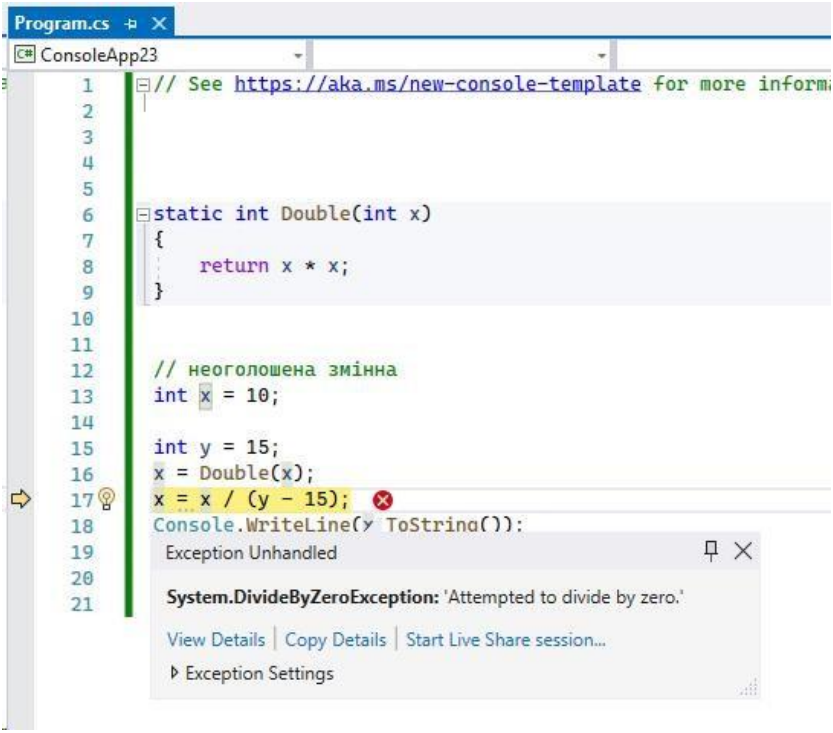
Якщо помилки компіляції допомагає знайти компілятор, то з логічними помилками дещо складніше. Додамо в код наступні зміни:

```
// See https://aka.ms/new-console-template

static int Double(int x)
{
    return x * x;
}

// неоголошена змінна
int x = 10;

int y = 15;
x = Double(x);
x = x / (y - 15);
Console.WriteLine(x.ToString());
```

Побудуємо рішення (помилок немає):

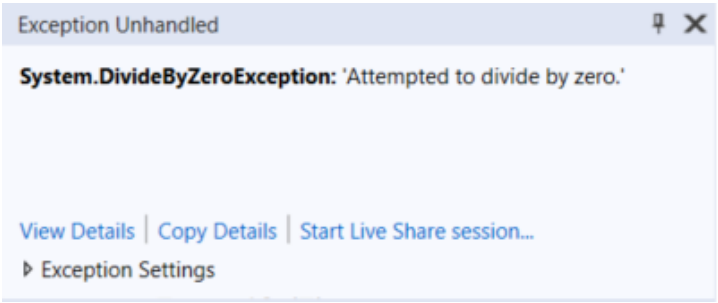


При запуску на виконання маємо помилку:

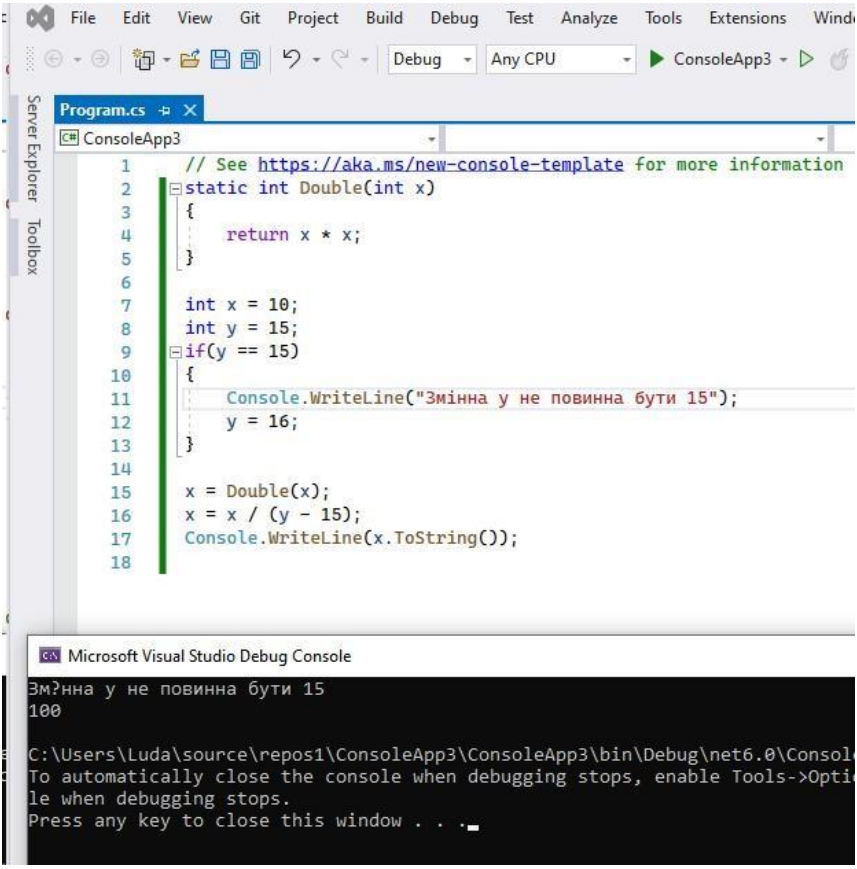


Сталася помилка, і середовище розробки взяло керування на себе, щоб ви змогли переглянути інформацію про помилку і спробували визначити її джерело. У редакторі коду жовтим кольором виділено рядок коду, в якому сталася помилка.

Назву помилки можна побачити в заголовку спливаючого вікна. Нижче під заголовком знаходиться опис помилки.



В даному випадку проблема проста (ділення на нуль) і наше завдання додати перед виконанням коду перевірку (перевірити, щоб змінна y не дорівнювала 15).

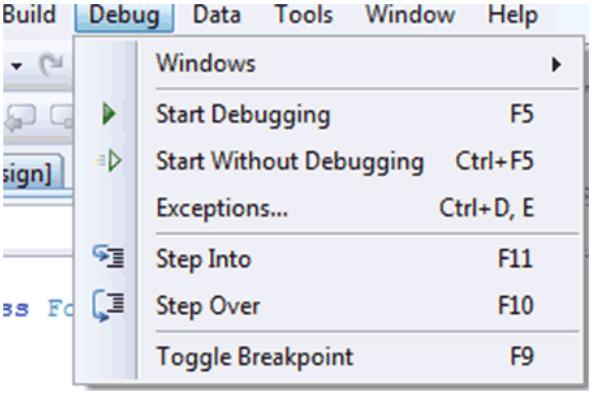


Помилку далеко не завжди так легко визначити. Бувають випадки, коли потрібно пройти програму покроково, рядок за рядком, у пошуках вразливого місця, яке призвело до проблеми.

Спробуємо пройти програму покроково. Проходити всю програму крок за кроком проблематично, тому що дуже багато відбуватися за лаштунками, в тому числі і обробка подій. Тому

найчастіше заздалегідь визначається блок коду, який потрібно проаналізувати.

Для того, щоб налагодження програми було доступне, програма повинна бути запущена в режимі налагодження з середовища розробки. Натиснення клавіші F5 (Start Debugging) запускає програму в налагоджувальному режимі. Щоб запустити програму без можливості налагодження, потрібно натиснути Ctrl + F5 (Start Without Debugging). Відповідні пункти меню можна знайти в меню Debug:



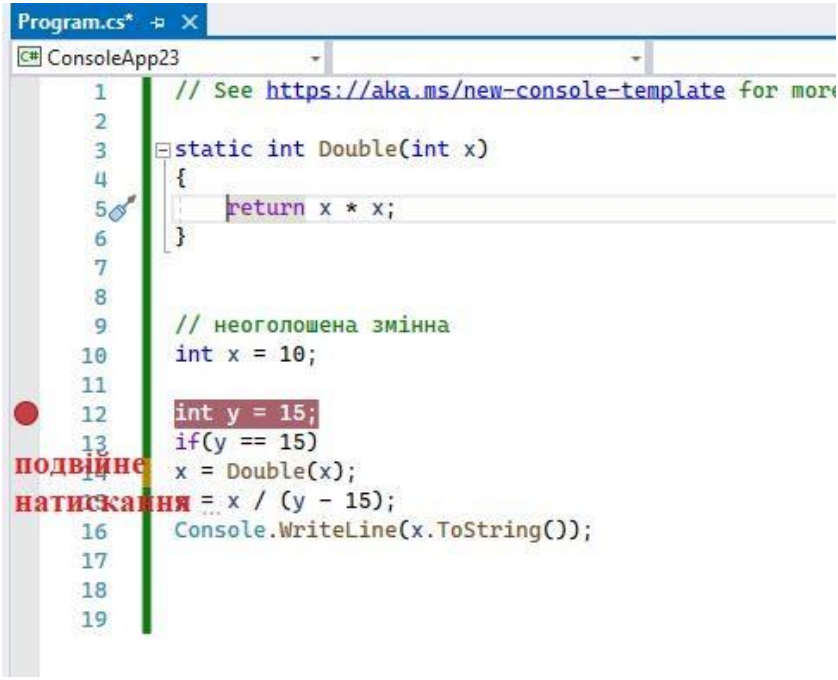
Debug | Start Debugging запустити в режимі налагодження

Debug | Start Without Debugging запустити в режимі виконання і без можливості налагодження.

Поставимо точки переривання в потрібному нам місці.

Точка переривання - це точка в коді програми, при досягненні якої виконання програми буде перервано і управління буде передано середовищу розробки.

Ви можете створювати точки зупинки (переривання) в довільному місці програми де є код і програма може перервати виконання.



Для створення точки перейдіть на потрібний рядок і:

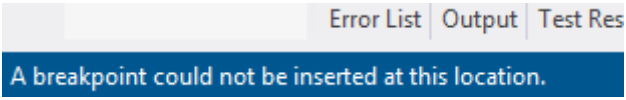
- Натисніть **F9**;

- Виберіть меню **Debug | Toggle Breakpoint**;

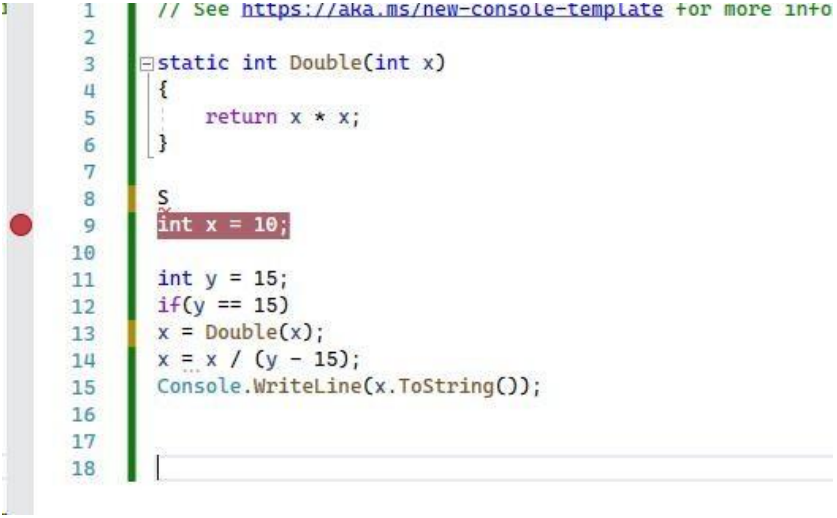
- Двічі клацніть на смужці сірого кольору, ліворуч від рядка тексту у вікні редактора коду.

Навпроти рядка на смужці сірого кольору зліва від тексту з'явитися червона крапка, що символізує, що тут стоїть точка зупинки. Якщо ще раз спробувати поставити точку зупинки на цьому ж рядку, то її буде знято.

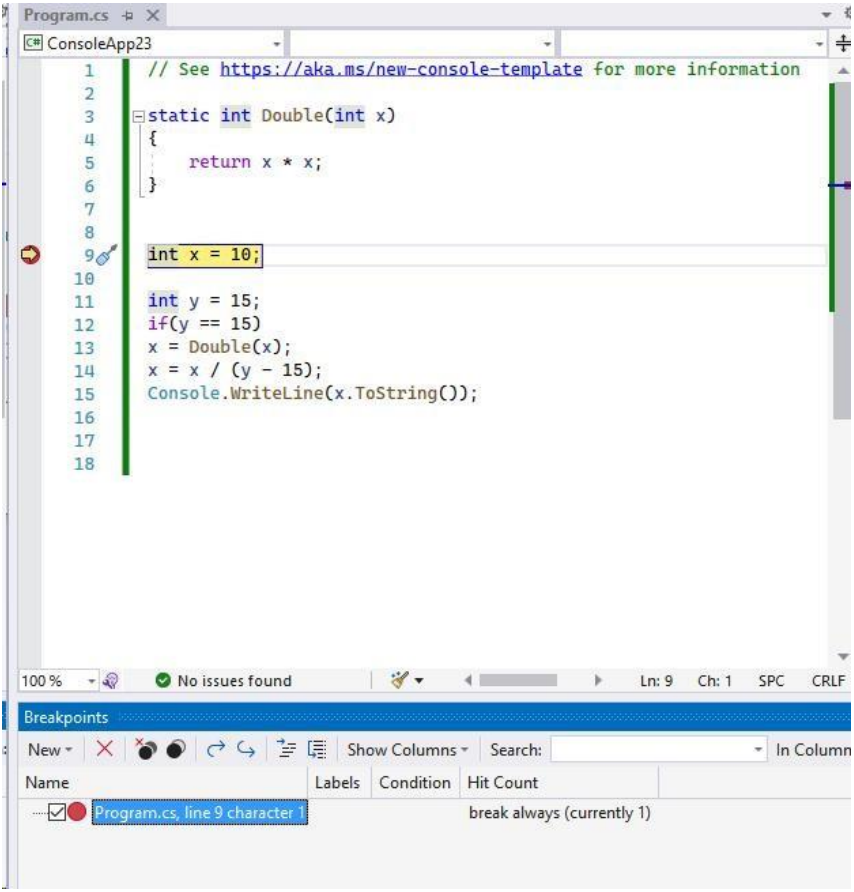
Якщо точку на поточному рядку встановити не можна, то в рядку стану середовища розробки з'явитися відповідне повідомлення:



Так як ми налагоджуємо метод з самого початку, то поставте курсор на рядок з іменем методу і натисніть F9. Тепер можна запускати програму. Точку зупинки можна ставити в будь-який момент, навіть під час виконання програми.

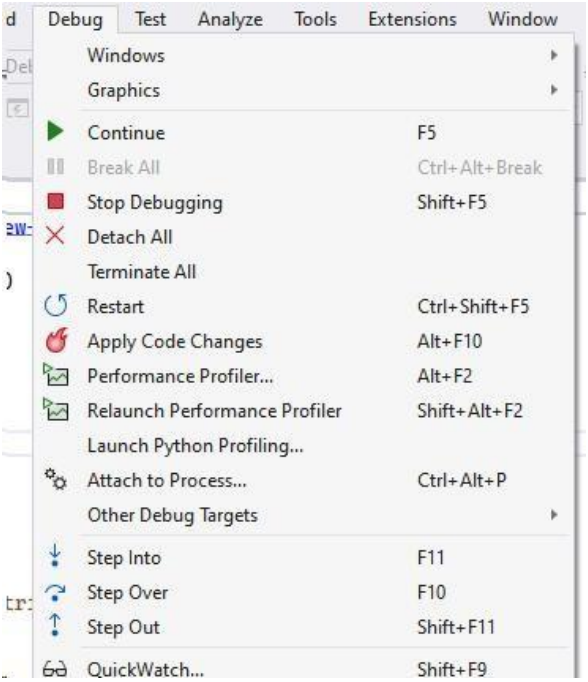


Запустіть програму. Так як ми поставили точку зупинки, середовище розробки перехопить на себе виконання і виділить оператор, який можна виконати наступним кроком жовтим кольором (назвемо цю точку курсором покрокового виконання програми):



Курсор покрокового виконання зупинився на точці і тепер хочемо почати тестування коду. На панелі з'явилася нова панель налагодження:

Ці команди також можна знайти в меню:



Continue (F5) - продовжити виконання програми.

Stop debugging (Shift+F5) - зупинити налагодження. При цьому зупиниться і виконання програми. Виконання програми перерветься на тій точці, на якій зараз і знаходитися, тобто вона не буде завершена коректно;

Restart (Ctrl+Shift+F5) - перезапустити програму. Виконання програми буде перервано;

Step Into (F11) - виконати черговий оператор з входженням в код методу. Якщо це метод, то перейти в початок цього методу, щоб почати налагодження. Наприклад, якщо ви знаходитесь на рядку: **x = Метод(x)** то, курсор покрокового виконання перейде на початок методу **Метод** і ви зможете налагодити цей метод;

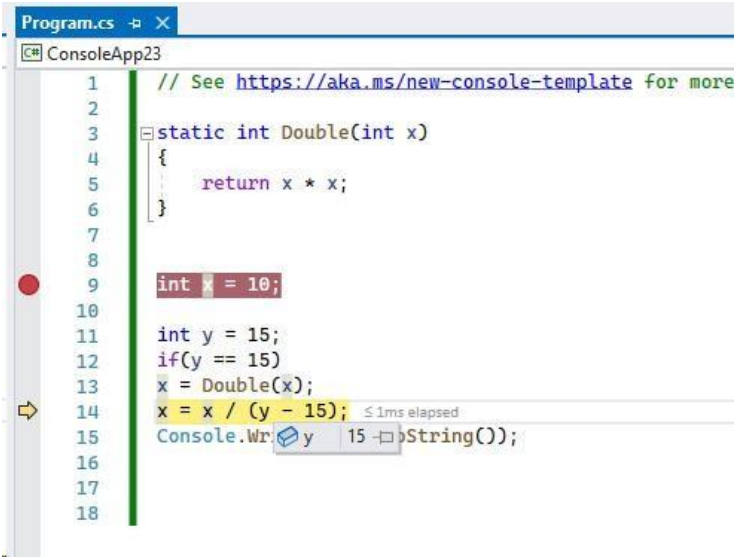
Step Over (F10) - виконати черговий оператор. Якщо це метод, то він буде повністю виконаний, тобто курсор виконання не входитиме всередину методу;

Step out (Shift+F11) - вийти з методу. Якщо ви налагоджуєте метод і натиснете цю кнопку, то метод виконається до кінця і курсор покрокового виконання вийде з методу і зупиниться на наступному рядку після виклику даного методу. Наприклад, якщо ви налагоджуєте деякий метод, і натиснете цю кнопку, то метод виконається до кінця, а виконання зупиниться на наступному за методом рядку.

Спробуйте покроково виконати код методу, натискаючи клавішу **F10**. Потім запустіть знову додаток і спробуйте покроково виконати його, натискаючи клавішу **F11**. Коли відбудеться помилка, спробуйте перервати роботу налагодження і додатку, натиснувши **Shift+F5**.

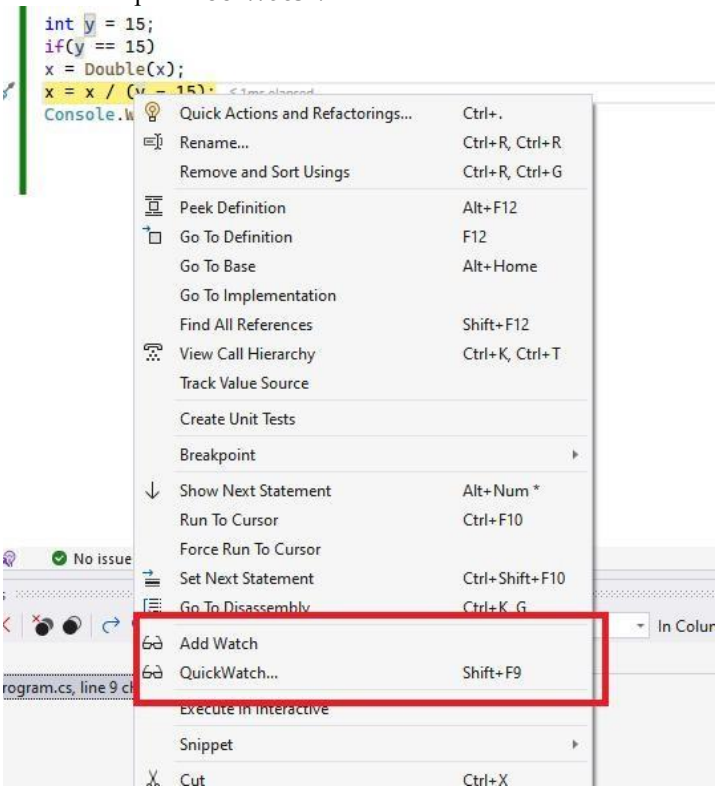
Виконайте покроково код так, щоб курсор виконання зупинився на наступному рядку: x = x / (y - 15);

Потрібно дізнатися, чому відбувається помилка. Так як помилка в діленні, то потрібно переглянути, на що відбувається ділення. У даному випадку це дужка. Наведіть мишкою на дужку, що відкривається або закривається і ви побачите спливаючу підказку, в якій знаходиться результат обчислення значення в дужках:



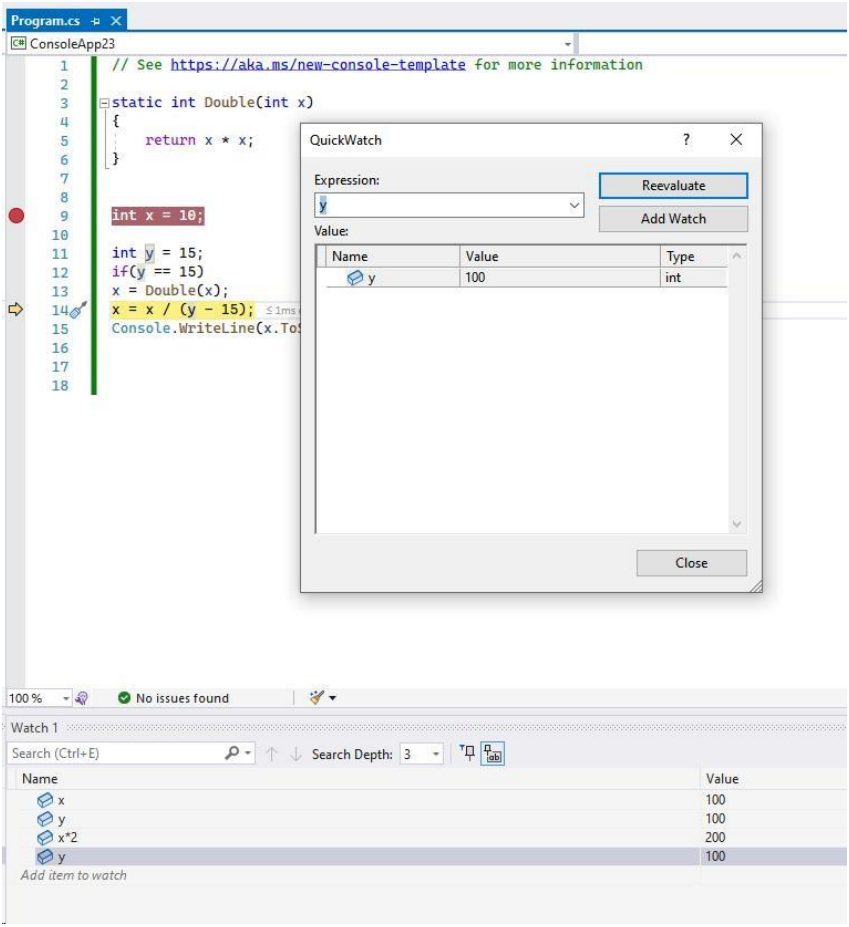
Результат знаходиться після вертикальної риски і він дорівнює нулю. Ви можете наводити мишкою на будь-яку змінну, і налагоджувач покаже вам її значення у вигляді спливаючої підказки.

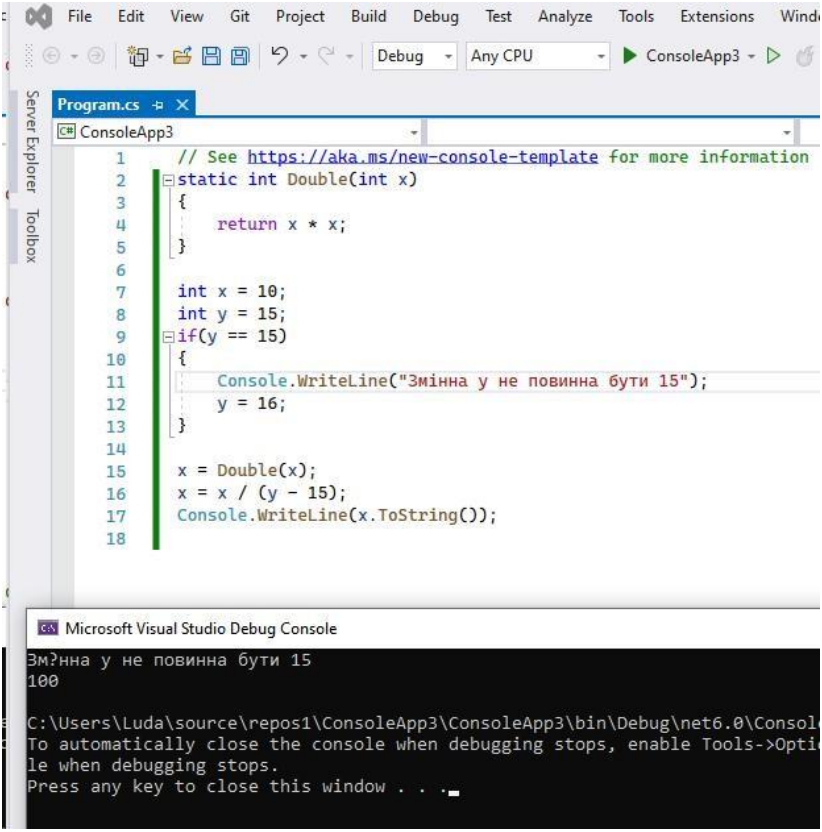
Якщо потрібно мати список змінних, значення яких будемо відслідковувати, тоді доцільно скористатися списком **Watch**. Для цього стоячи на змінній, в режимі відлагодження, натисніть праву кнопку миші і в меню оберіть **Add Watch**.



Крім того, спробуйте **Add Parallel Watch**

та **QuickWatch**:


Program.cs
ConsoleApp23
// See https://aka.ms/new-console-template for more information
static int Double(int x)
{
return x * x;
}
int x = 10;
int y = 15;
if(y == 15)
x = Double(x);
x = x / (y - 15);
Console.WriteLine(x.To
QuickWatch
Expression:
y
Reevaluate
Add Watch
Value:
Name
Value
Type
y
100
int
Close
100 %
No issues found
Watch 1
Search (Ctrl+E)
Search Depth: 3
Name
Value
x
100
y
100
x*2
200
y
100
Add item to watch


File Edit View Git Project Build Debug Test Analyze Tools Extensions
Debug
Any CPU
ConsoleApp3
Server Explorer
Toolbox
Program.cs
ConsoleApp3
// See https://aka.ms/new-console-template for more information
static int Double(int x)
{
return x * x;
}
int x = 10;
int y = 15;
if(y == 15)
{
Console.WriteLine("Змінна у не повинна бути 15");
y = 16;
}
x = Double(x);
x = x / (y - 15);
Console.WriteLine(x.ToString());
Microsoft Visual Studio Debug Console
Зм?нна у не повинна бути 15
100
C:\Users\Luda\source\repos1\ConsoleApp3\ConsoleApp3\bin\Debug\net6.0\Consol
To automatically close the console when debugging stops, enable Tools->Opti
le when debugging stops.
Press any key to close this window . . .

ТИЖДЕНЬ 4

Тема 4. МОВА C#: КЛАСИ, ПЕРЕТВОРЕННЯ ТИПІВ, ОПЕРАТОРИ ТА ВИРАЗИ, ОПЕРАТОРИ ОБРОБКИ ВИКЛЧЕНЬ, НЕЯВНІ, АНОНІМНІ ТА ДИНАМІЧНІ ТИПИ

Огляд, мета і призначення теми: навчитися використовувати принципи об'єктно-орієнтованого програмування на прикладі мови програмування C#. Класи, перетворення типів, оператори та вирази, оператори обробки виключень, неявні, анонімні та динамічні типи даних.

Вивчивши матеріал даної теми, студент зможе: вміти пояснити відмінності між класами та об’єктами. Знати та розуміти основні парадигми ООП. Розуміти та вміти використовувати перетворення типів, оператори та вирази, оператори обробки виключень, неявні, анонімні та динамічні типи даних в мові C#. Ознайомитися з вимогами та рекомендаціями з написання «чистого» коду, знати та в міти застосовувати різні види рефакторингу.

Лабораторний практикум 4

Вимоги та рекомендації з написання коду.
Рефакторинг.
Поліморфізм та спадкування в C# (60 хв.).
Прийом домашнього завдання (30 хв.)

Завдання для опрацювання матеріалу

Завдання 1. Використовуючи Visual Studio, створіть проєкт за шаблоном MAUI App (.NET MAUI). Створіть клас «Трикутник» із полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметра. Створити клас-нащадок «Рівносторонній трикутник», який має поле площі. Визначити метод обчислення площі.

Завдання 2. Використовуючи Visual Studio, створіть проєкт за шаблоном MAUI App (.NET MAUI). Напишіть програму для роботи з геометричними фігурами (трикутник, круг, прямокутник, квадрат, ромб). У програмі створити абстрактний базовий клас «Фігура» з віртуальними методами обчислення площі й периметра та його нащадків – «Трикутник», «Круг», «Прямокутник», «Квадрат», «Ромб».

Завдання 3. Використовуючи Visual Studio, створіть проєкт за шаблоном MAUI App (.NET MAUI). Створіть абстрактний клас «Трикутник» із віртуальними методами обчислення площі й периметра.

Поля даних повинні мати дві сторони й кут між ними. Визначити класи-нащадки «Прямокутний трикутник» і «Рівнобедрений трикутник» зі своїми функціями обчислення площі й периметра.

Завдання 4. Маємо "Географічний об'єкт", який складається з:

- Координата X;
- Координата У;
- Назва;
- Опис;
- Метод отримання інформації.

Маємо об'єкт "Річка", що містить додаткову інформацію:

- Швидкість течії (см/с);
- Загальна довжина.

Маємо ще один об'єкт "Гора", що містить додаткову інформацію

- "Найвища точка".

Відповідно до опису об'єктів, необхідно зробити наступне:

1. Створити новий solution.
2. Створити новий проект Console Application.
3. Додати новий абстрактний клас "Географічний об'єкт" ("Метод отримання інформації" - віртуальний).
4. Додати класи "Річка" та "Гора", що наслідуються від абстрактного класу з п.3.
5. Додати новий проект Console Application.
6. Додати новий інтерфейс "Географічний об'єкт".
7. Додати класи "Річка" та "Гора", що реалізують інтерфейс з п.6.

Завдання 5. Виконати наступні кроки:

1. Створюємо проект за шаблоном MAUI App (.NET MAUI).
2. Додаємо клас "Товар" (Ціна, Країна походження, Назва, Дата пакування, Опис).
3. Додаємо клас-нащадок "Продукти" (Термін придатності, К-ть, Одиниця виміру).
4. Додаємо клас-нащадок "Книги" (Кількість сторінок, Видавництво, Перелік авторів).
5. Додати на форму компоненти:
   5.1. таблицю (grid) (з переліком товарів);
   5.2. дві кнопки (button) (які відповідають за додавання та видалення різних товарів);

6. Заповнити деякою кількістю товарів grid з п.5 та забезпечити функціональність кнопок.

Перелічені компоненти та класи є мінімально-необхідним наповненням проекту

Завдання на опанування поліморфізму

Завдання 4. Створити консольний додаток. Запустіть його. Який результат?

```
4 references
public class Class1
{
    2 references
    public void Method1()
    {
        Console.WriteLine("Class1 Method1");
    }

    2 references
    public void Method2()
    {
        Console.WriteLine("Class1 Method2");
    }
}

3 references
public class Class2 : Class1
{
    1 reference
    public void Method1()
    {
        Console.WriteLine("Class2 Method1");
    }

    1 reference
    public void Method2()
    {
        Console.WriteLine("Class2 Method2");
    }
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Class1 x = new Class1();
        Class2 y = new Class2();
        Class1 z = new Class2();

        x.Method1(); x.Method2();
        Console.WriteLine("");
        y.Method1(); y.Method2();
        Console.WriteLine("");
        z.Method1(); z.Method2();
        Console.ReadKey();
    }
}
```

Завдання 5. Змініть консольний додаток. Запустіть його. Який результат?

```
public class Class1
{
    1 reference
    public void Method1()
    {
        Console.WriteLine("Class1 Method1");
    }

    1 reference
    public void Method2()
    {
        Console.WriteLine("Class1 Method2");
    }
}

4 references
public class Class2 : Class1
{
    2 references
    public new void Method1()
    {
        Console.WriteLine("Class2 Method1");
    }

    2 references
    public override void Method2()
    {
        Console.WriteLine("Class2 Method2");
    }
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Class1 x = new Class1();
        Class2 y = new Class2();
        Class2 z = new Class2();

        x.Method1(); x.Method2();
        Console.WriteLine("");
        y.Method1(); y.Method2();
        Console.WriteLine("");
        z.Method1(); z.Method2();
        Console.ReadKey();
    }
}
```

Завдання 6. Внесіть зміни в додаток. Запустіть його. Який результат?

```
4 references
public class Class1
{
    2 references
    public virtual void Method1()
    { Console.WriteLine("Class1 Method1"); }
    4 references
    public virtual void Method2()
    {  Console.WriteLine("Class1 Method2"); }
    2 references
    public virtual void Method3()
    { Console.WriteLine("Class1 Method3"); }
}
3 references
public class Class2 : Class1
{
    1 reference
    public new void Method1()
    { Console.WriteLine("Class2 Method1"); }
    4 references
    public override void Method2()
    { Console.WriteLine("Class2 Method2"); }
    1 reference
    public void Method3()
    { Console.WriteLine("Class2 Method3"); }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Class1 x = new Class1();
        Class2 y = new Class2();
        Class1 z = new Class2();
        x.Method1(); x.Method2(); x.Method3();
        Console.WriteLine("");
        y.Method1(); y.Method2(); y.Method3();
        Console.WriteLine("");
        z.Method1(); z.Method2(); z.Method3();
        Console.ReadKey();
    }
}
```

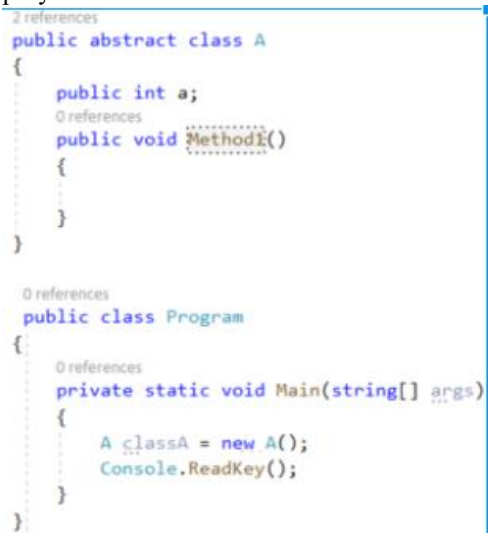
Завдання 7. Які висновки можна зробити? Виберіть усі правильні відповіді.

У C # ми можемо записати в змінну класу-батька об'єкт спадкоємця, але не навпаки;

- ☐ Модифікатор override використовується, щоб вказати на те, що повинен викликатися метод саме дочірнього класу;
- ☐ Щоб використовувати модифікатор override, метод батьківського класу повинен бути позначений ключовим словом virtual;
- ☐ Коли викликається метод якогось об'єкта по посиланню, то C # в першу чергу дивиться на тип посилання. Якщо в цьому класі виявлений модифікатор virtual, він починає шукати серед дочірніх класів тип об'єкта, і, якщо зустрічає new, запускає останній override метод, який зустрів (або метод типу посилання).
- ☐ Щоб використовувати модифікатори virtual і new, метод батьківського класу повинен бути позначений ключовим словом override;
- ☐ У C # ми можемо записати в змінну класу-нащадку об'єкт батьківського класа, але не навпаки;
- ☐ У C # ми можемо записати в змінну класу-батька об'єкт спадкоємця, та в змінну класу-нащадку об'єкт батьківського класу.

Завдання на опанування роботи з абстрактними класами

Завдання 8. Створити консольний додаток. Запустіть його. Який результат?



Завдання 9. Створити консольний додаток. Запустіть його. Який результат?

```
1 reference
public abstract class A
{
    public int a;
    1 reference
    public void Method1()
    {
        Console.WriteLine("abstract class A Method1");
    }
    0 references
    abstract public void Method2();

}
2 references
public class B : A
{

}
 0 references
 public class Program
{
    0 references
    private static void Main(string[] args)
    {
        B b = new B();
        b.Method1();
        Console.ReadKey();
    }
}
```

Завдання 10. Створити консольний додаток. Запустіть його. Який результат?

```
1 reference
public abstract class A
{
    public int a;
    1 reference
    public void Method1()
    {
        Console.WriteLine("abstract class A Method1");
    }
    1 reference
    abstract public void Method2();

}
2 references
public class B : A
{
    1 reference
    public override void Method2()
    {
        Console.WriteLine("abstract class B Method2");
    }
}
0 references
public class Program
{
    0 references
    private static void Main()
    {
        B b = new B();
        b.Method1();
        Console.ReadKey();
    }
}
```

Завдання 11. Створити консольний додаток. Запустіть його. Який результат?

```
namespace ConsoleApp2
{
    1 reference
    public  class A
    {
        public int a;
        1 reference
        public void Method1()
        {
            Console.WriteLine("abstract class A Method1");
        }
        1 reference
        abstract public void Method2();

    }
    2 references
    public class B : A
    {
        1 reference
        public override void Method2()
        {
            Console.WriteLine("abstract class B Method2");
        }
    }
    0 references
    public class Program
    {
        0 references
        private static void Main()
        {
            B b = new B();
            b.Method1();
            Console.ReadKey();
        }
    }
```

Завдання 12. Створити консольний додаток. Запустіть його. Який результат?

```
1 reference
public abstract class A
{
    public int a;
    1 reference
    public void Method1()
    {
        Console.WriteLine("abstract class A Method1");
    }
    2 references
    abstract public void Method2();

}
2 references
public class B : A
{
    2 references
    public override void Method2()
    {
        base.Method2();
        Console.WriteLine("abstract class B Method2");
    }
}
0 references
public class Program
{
    0 references
    private static void Main()
    {
        B b = new B();
        b.Method1();
        Console.ReadKey();
    }
}
```

Завдання 13. Створити консольний додаток. Запустіть його. Який результат?

```
4 references
public class A
{
    2 references
    public void Method1()
    {
        Console.WriteLine("class A Method1");
    }
}
2 references
public abstract class B : A
{
    2 references
    public new abstract void Method1();
}

2 references
public class C : B
{
    2 references
    public override void Method1()
    {
        Console.WriteLine("class C Method1");
    }
}
0 references
public class Program
{
    0 references
    private static void Main()
    {
        A a = new A();
        B b = new C();
        A c = new C();
        a.Method1();
        b.Method1();
        c.Method1();
        Console.ReadKey();
    }
}
```

Завдання 14. Створіть консольний додаток. Запустіть його. Який результат?

```
7 references
public class A
{
    3 references
    public void Method1()
    {
        Console.WriteLine("class A Method1");
    }
}
1 reference
public static abstract class B : A
{
    1 reference
    public new abstract void Method1();
}

0 references
public class C : B
{
    1 reference
    public override void Method1()
    {
        Console.WriteLine("class C Method1");
    }
}
0 references
public class Program
{
    0 references
    private static void Main()
    {
        A a = new A();
        A b = new A();
        A c = new A();
        a.Method1();
        b.Method1();
        c.Method1();
        Console.ReadKey();
    }
}
```

Завдання 15. Які висновки можна зробити? Виберіть усі правильні відповіді.

- ☐ Не можна створити екземпляр абстрактного класу за допомогою ключового слова new;
- ☐ Можна успадкувати звичайний клас від абстрактного;
- ☐ Можна створити екземпляр абстрактного класу за допомогою ключового слова new;
- ☐ Можна створити екземпляр абстрактного класу, за допомогою ключового слова new.
- ☐ Якщо ми хочемо оголосити метод в абстрактному класі, але не реалізовувати його, тоді до методу потрібно додати ключове слово abstract;
- ☐ Якщо ми оголошуємо абстрактний метод в абстрактному класі, то цей метод повинен реалізовуватися у неабстрактних спадкоємців цього класу;
- ☐ Абстрактні методи можуть бути оголошені тільки в абстрактних класах;
- ☐ Абстрактний клас може мати модифікатор static.
- ☐ Абстрактний клас не може мати модифікатор static.
- ☐ Не можна успадкувати звичайний клас від абстрактного; Можна створити екземпляр звичайного класу, успадкованого від абстрактного, за допомогою ключового слова new;
- ☐ Не можна створити екземпляр звичайного класу, успадкованого від абстрактного за допомогою ключового слова new.

Завдання на освоєння інкапсуляції (модифікатори доступу)

Завдання 16. Створити консольний додаток. Запустіть його. Який результат?

```
1 reference
class Modifiers
{
    1 reference
    static void A()
    {
        Console.WriteLine("Modifiers A");
    }
    1 reference
    public static void B()
    {
        Console.WriteLine("Modifiers B");
        A();
    }
}
0 references
class Program
{
    0 references
    static void Main()
    {
        Modifiers.B();
        Console.ReadKey();
    }
}
```

Завдання 17. Створити консольний додаток. Запустіть його. Який результат?

```
1 reference
class Modifiers
{
    2 references
    static void A()
    {
        Console.WriteLine("Modifiers A");
    }
    0 references
    public static void B()
    {
        Console.WriteLine("Modifiers B");
        A();
    }
}
0 references
class Program
{
    0 references
    static void Main()
    {
        Modifiers.A();
        Console.ReadKey();
    }
}
```

Завдання 18. Створити консольний додаток. Запустіть його. Який результат?

```
1 reference
class Modifiers
{
    1 reference
    static void A()
    {
        Console.WriteLine("Modifiers A");
    }
    1 reference
    public static void B()
    {
        Console.WriteLine("Modifiers B");
    }
    1 reference
    protected static void C()
    {
        Console.WriteLine("Modifiers C");
    }
}

1 reference
class ModifiersChild : Modifiers
{
    1 reference
    public static void D()
    {
        A();
        B();
        C();
    }
}
0 references
class Program
{
    0 references
    static void Main()
    {
        ModifiersChild.D();
        Console.ReadKey();
    }
}
```

Завдання 19. Створити консольний додаток. Запустіть його. Який результат?

```
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6   using TestModifierLibrary;
7
8   public namespace ConsoleApp2
9   {
        0 references
10      private class Program
11      {
            0 references
12          static void Main(string[] args)
13          {
14              Console.ReadKey();
15          }
16      }
17  }
```

Завдання 20. Створити консольний додаток. Запустіть його. Який результат?

```
3 references
class A
{
    protected int a;
    0 references
    void MethodA(A a1, B b1)
    {
        a1.a = 1;
        b1.a = 2;
    }
}

2 references
class B:A
{
    0 references
    void MethodB(A a1, B b1)
    {
        a1.a = 1;
        this.a = 10;
        b1.a = 2;
    }
}
0 references
public class Program
{
    0 references
    static void Main(string[] args)
    {
        Console.ReadKey();
    }
}
```

Завдання 21. Створити консольний додаток. Запустіть його. Який результат?

```
1 reference
class A
{

}

0 references
public class B:A
{

}
0 references
public class Program
{
    0 references
    static void Main(string[] args)
    {
        Console.ReadKey();
    }
}
```

Завдання 22. Які висновки можна зробити? Виберіть усі правильні відповіді.

- ☐ Не вкладені класи, що визначені безпосередньо в просторі імен, можуть мати модифікатор доступу - private.
- ☐ Модифікатор доступу за замовчуванням для членів класу – private.
- ☐ Модифікатор доступу internal значить, що доступ дозволений тільки з тієї ж збірки.
- ☐ Простори імен можуть мати модифікатори доступу public, protected, private.
- ☐ У просторів імен немає і не може бути модифікаторів доступу (можна вважати, що вони всі public).
- ☐ Модифікатор protected internal означає, що доступ є як з тієї ж збірки, так і з дочірніх класів.
- ☐ Батьківський клас не може бути менш доступний, ніж дочірній.

- ☐ Модифікатор доступу за замовчуванням для членів класу – public.
- ☐ Значення, що повертається методу не може бути менш доступне, ніж сам метод.
- ☐ Поле може бути доступніше, ніж його клас.
- ☐ Поле не може бути доступніше, ніж його клас.

Запитання до теми:

1. Що таке клас?
2. Що таке об'єкт?
3. Як пов'язані між собою класи та об'єкти в програмі?
4. Що таке інкапсуляція?
5. За рахунок чого реалізується захист від несанкціонованого доступу до даних?
6. Що таке властивість?
7. Які методи є у властивості?
8. Що роблять ці методи?
9. Що таке точка входу?
10. Що таке масив?
11. Як масив представляється в C #?
12. Які види масивів визначаються в C #?
13. Які призначення і логіка роботи циклу `foreach`?
14. Яке значення індекса першого елемента в масиві?
15. Що таке делегат?

Завдання для самостійної роботи.

Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторного заняття. Виконання домашнього завдання №4. Створити усі необхідні для реалізації лабораторної роботи №1 клас (для комірки - залежить від Вашого проєктування), створити основну форму та додати на неї DataGridView.

ТЕОРЕТИЧНА ЧАСТИНА ЛАБОРАТОРНОГО ЗАНЯТТЯ
ВИМОГИ І РЕКОМЕНДАЦІЇ З НАПИСАННЯ КОДУ

Написати програму – це більше, ніж правильно її оформити та змусити коректно виконуватись. Програми згодом неминуче модифікуються, повторно використовуються для побудови інших програм тощо. Тому велике значення має простота та зрозумілість їхньої внутрішньої структури. Інколи добре написану чужу програму набагато простіше зрозуміти, ніж власну, але написану погано. Культура написання коду сприяє зменшенню помилок у програмах і полегшує їхню модифікацію та повторне використання.

Під **стилем** програмування зазвичай розуміють набір прийомів і методів, що застосовуються з метою одержання правильних і ефективних програм, зручних для сприйняття, повторного застосування й модифікації.

Вибір імені

Імена мають програми, інтерфейси, класи, поля, …. Коли їх у програмі з десяток чи два, то вибір не є дуже принциповим питанням. Однак коли їхня кількість у системі сягає десятків, сотень, а то й тисяч, то, щоб не втратити контроль над системою, при виборі імен необхідно дотримуватись певної строгої дисципліни й порядку.

Доцільно використовувати осмислені імена для глобальних змінних і, за можливості, короткі – для локальних. Глобальні змінні, функції, класи та структури за означенням можуть з'являтись у довільному місці програми, тому найважливішою для них є інформативність, а не довжина. Краще використовувати глобальні імена так, щоб вони явно вказували на смисл понять, що ними подаються, тобто мали відповідну **мнемоніку**. Корисно опис кожної змінної, методу, класу, інтерфейсу супроводжувати коротким коментарем.

Однак коментарі не мають містити очевидної інформації на зразок того, що оператор `i++` збільшує змінну `i` тощо.

Для локальних змінних, навпаки, краще підходять короткі імена. Для використання всередині функції цілком підійдуть імена `i, j, n` тощо. При цьому для лічильників циклу зазвичай

використовуються імена `i, j`, для рядків – `s, t`. Як правило, використовують англомовні назви змінних, назв класів, інтерфейсів, методів, властивостей, полів тощо, які відповідають їх ролі в програмі.

Існує багато корпоративних домовленостей і традицій іменування, які фіксуються у спеціальних внутрішніх стандартах виробників. Наприклад, часто для методів використовуються імена, побудовані з дієслів та іменників, а для полів, класів, властивостей – іменники.

Форматування тексту

Оператори й вирази. Ці основні конструкції програм слід писати так, щоб їхній зміст був максимально зрозумілим. Найпростіший спосіб досягти цього – форматування коду за допомогою відступів, табуляцій, порожніх рядків. Розглянемо два тексти однієї й тієї самої програми. Очевидно, що другий текст більш читабельний саме завдяки використанню його якісного форматування.

Лістинг 1.

```
public class TestCollections{  public static void TestList()
{var testList = new List<int>{3, 4, 2, 1};
for(var i = testList.Count -1; i >= 0; i--) {if(testList[i]>2) testList.RemoveAt(i); }
testList.Sort();foreach (int el in testList) {Console.WriteLine(el); }}}
```

Лістинг 2.

```
public class TestCollections
{
    public static void TestList()
    {
        var testList = new List<int>{3, 4, 2, 1};
        for(var i = testList.Count -1; i >= 0; i--)
```

```
        {
                if(testList[i]>2)
                {
                        testList.RemoveAt(i);
                }
        }
            /*сортування в лексикографічному порядку*/
                testList.Sort();
        foreach (int el in testList)
        {
            Console.WriteLine(el);
        }
    }
}
```

Зрозумілість і стислість коду – не одне й те саме. Головним критерієм вибору синтаксису для коду має бути простота його сприйняття.

Відступи, табуляції, переходи на новий рядок допомагають краще структурувати код. Порожні рядки допомагаю розбивати код програми на логічні сегменти.

Декількома рядками можуть відділятися: секції у вихідному файлі, класи та інтерфейси.

Одним порожнім рядком відокремлюються один від одного: методи, локальні змінні від перших операторів, логічні секції всередині методу для більш зручного читання.

Один прогалик використовується в оголошенні методів після коми, але не перед дужками: `TestMethod (a, b, c);`

Приклад неправильного використання:

`TestMethod (a,b ,c);` або `TestMethod (a, b, c );`

Так само одиночний прогалик може бути використаний для виділення операторів: `a = b;` неправильне використання: `a=b;`

Також прогалики використовуються і при форматуванні циклів:`for (int i = 0; i <10; ++i);` неправильне використання: `for (int i=0; i<10; ++i),` або `for (int i = 0;i <10;+ + i).`

При оголошенні і ініціалізації змінних бажано використовувати «табличне» форматування:

```
string name     =     "Mr. Ed";
int myValue     =     5;
Test aTest      =     Test.TestYou;
```

Чи варто розташовувати відкриваючу фігурну дужку в тому самому рядку, що й `if` або `while`, чи в наступному? Важлива не стільки конкретика вибраного стилю, скільки логічність і послідовність його застосування.

Основні домовленості та рекомендації з написання коду

Деякі домовленості та рекомендації з написання коду на C#:

- Не використовуйте публічних або захищених полів, замість цього використовуйте властивості.
- Використовуйте автоматичні властивості.
- Завжди вказуйте модифікатор доступу `private`, навіть якщо дозволено його опускати.
- Завжди ініціалізуйте змінні, навіть коли існує автоматична ініціалізація.
- Використовуйте порожній рядок між логічними секціями у вихідному файлі, класі, методі.
- Використовуйте проміжну змінну для передачі bool-значення результату функції в умовний вираз `if`.

```
bool boolVariable = GetBoolValue ();
if (boolVariable)
{
}
```

- Уникайте файлів з більш ніж 500 рядками коду.

- Уникайте методів з більш ніж 200 рядками коду.
- Уникайте методів з більш ніж 5 аргументами, використовуйте структури для передачі великого числа параметрів.
- Один рядок коду не повинна мати довжину більше 120 символів.
- Для коментарів у один рядок використовується «C++» подібний стиль: «//» Цей стиль найбільш зручний при документуванні параметрів. Переважно використовувати даний стиль замість `/ * коментар * /` там, де це можливо.

```
int i; // змінна для циклу.
```

- Для стандартних блокових коментарів використовуються наступні стилі:

```
/ *
* Line 1
* Line 2
* Line 3
* /
або такий стиль:
/ * Blabla * /
```

- Загальноприйнятим стандартом в оголошеннях є один рядок на екземпляр. Завдяки цьому з'являється зручність читання і написання коментаря:

```
int level; // indentation level
int size; // size of table
```

Бажано не ставити оголошення в один рядок.

```
int a, b; // Неправильно!
```

- Розташовуйте відкриваючі та закриваючі фігурні дужки на новому рядку.
- Використовуйте фігурні дужки для виразів if, навіть коли у вираз входить лише один рядок коду.
- Не використовуйте пробіл замість символу табуляції.

- Намагайтеся застосовувати максимум інкапсуляції для примірників об'єктів які ви створюєте. Чи не ставте `public` там де це не потрібно. Використовуйте максимальний рівень захисту. Тобто намагайтеся відкривати доступ від мінімального (`private`) до максимального (`public`).

- Використовуйте властивості замість прямого відкриття `public`, для змінних класу.

- Не використовуйте так званих «магічних чисел». Замість цього оголошуйте константи і статичні змінні:

```
public class MyMath

{

public const double PI = 3.14159;
 }
```

Стилі використання регістрів

Нижче описані різні способи написання ідентифікаторів та рекомендації з їх застосування.

Стиль Pascal casing

Перша літера ідентифікатора і перша буква кожного наступного приєднаного слова є прописними. Стиль Pascal можна використовувати для ідентифікаторів, що складаються з трьох і більше букв.

Pascal casing рекомендується використовувати для опису:

• всіх визначень типів, у тому числі призначених для користувача класів, перерахувань, подій, делегатів і структур;

• значення перерахувань;

• `readonly` полів і констант;

• інтерфейсів;

• методів;

• просторів імен (`namespace`);

• властивостей;

• публічних полів;

Стиль Camel casing

Перша літера ідентифікатора рядкова, а перша літера кожного наступного приєднаного слова - прописана.

Стиль **Camel casing** рекомендується використовувати для опису імен:

• локальних змінних;

• аргументів методів;

• захищених (`protected`) полів.

Стиль Upper Case

Всі букви в назві є прописними. Цей стиль рекомендується використовувати лише при іменуванні «коротких» констант, наприклад `PI` або `E`. В інших випадках бажано використовувати Pascal Casing.

В таблиці 4 представлено зведену таблицю використання регістрів, префіксів та суфіксів при найменуванні ідентифікаторів в C#.

Таблиця 4. Зведена таблиця використання регістрів та префіксів

Ідентифікатор	**Регістр**	**Приклад**
Клас / структура	Pascal Casing	public class TestClass { ... }
Інтерфейс	Pascal Casing Починається префіксом «I»	public interface IEnumerable { ... }

Значення перелічного типу	Pascal Casing	enum SampleEnum
		{
		FirstValue,
		SecondValue
		}

Перелічний тип	Pascal Casing	
Делегат обробник події	Pascal Casing Закінчується суфіксом «EventHandler»	public delegate void AnswerCreatedEventHandler(object sender, AnswerCreatedEventArgs e); public class AnswerCreatedEventArgs: Event

Клас-нащадок від EventArgs	Pascal Casing Закінчується суфіксом «EventArgs»	Args
		{
		public int CreatedId;
		public int ParentId;
		public string CreatorName;
		}

Класи виключень	Pascal Casing Закінчується суфіксом «Exception»	public class SampleException:
		System.Exception
		{
		public SampleException()
		{
		}
		}

Namespace, властивості, методи, public поля,	Pascal Casing	namespace System.Security { … }
Protected/private поля, параметри	Camel Casing Починається з суфікса «_»	private int _samplePrivateField;

Користувацький атрибут	Pascal Casing Закінчується суфіксом «Attribute»	[System.AttributeUsage(System.AttributeTargets.All, Inherited = false, AllowMultiple = true)]
		sealed class SampleAttribute:
		System.Attribute
		{
		public SampleAttribute()
		{
		}
		}

«Коротка» (1-3 літери) константа	Стиль Upper Case	public const PI = 3.14159;

ТИЖДЕНЬ 5

Тема 5. МОВА C#: КЛАСИ, ПЕРЕТВОРЕННЯ ТИПІВ, ОПЕРАТОРИ ТА ВИРАЗИ, ОПЕРАТОРИ ОБРОБКИ ВИКЛЮЧЕНЬ, НЕЯВНІ, АНОНІМНІ ТА ДИНАМІЧНІ ТИПИ ДАНИХ

Огляд, мета і призначення теми: навчитися використовувати принципи об'єктно-орієнтованого програмування на прикладі мови програмування C#. Класи, перетворення типів, оператори та вирази, оператори обробки виключень, неявні, анонімні та динамічні типи даних.

Вивчивши матеріал даної теми, студент зможе: вміти пояснити відмінності між класами та об’єктами. Знати та розуміти основні парадигми ООП. Розуміти та вміти використовувати перетворення типів, оператори та вирази, оператори обробки виключень, неявні, анонімні та динамічні типи даних в мові C#.Знати основи роботи з .Net MAUI.

Лабораторний практикум 5

Поліморфізм та спадкування в C#, абстрактні класи, інкапсуляція, робота з файлами (60 хв.), робота з MAUI. Інтеграція з сервісами Google. Прийом домашнього завдання (30 хв.)

Завдання для опрацювання матеріалу

Завдання на освоєння принципів роботи з типами var, dynamic, object

Завдання 1. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
static void Main(string[] args)
{
    dynamic a;
    a = 10;
    Console.WriteLine("a=" + a);

    Console.ReadKey(true);
}
```

Завдання 2. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
static void Main(string[] args)
{
    object a;
    a = 10;
    Console.WriteLine("a=" + a);

    Console.ReadKey(true);
}
```

Завдання 3. Створити консольний додаток. Запустіть його. Який результат?

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        var a;
        a = 10;
        Console.WriteLine("a=" + a);

        Console.ReadKey(true);
    }
}
```

Завдання 4. Створити консольний додаток. Запустіть його. Який результат?

```
object a = 5;
Console.WriteLine("a=" + a);
object b = "bbb";
Console.WriteLine("b=" + b);
object c = a;
Console.WriteLine("c=" + c);
object d = "aaa";
Console.WriteLine("d=" + d);
d = 10;
Console.WriteLine("d=" + d);
```

Завдання 5. Створити консольний додаток. Запустіть його. Який результат?

```
dynamic a = 5;
Console.WriteLine("a=" + a);
dynamic b = "bbb";
Console.WriteLine("b=" + b);
dynamic c = a;
Console.WriteLine("c=" + c);
dynamic d = "aaa";
Console.WriteLine("d=" + d);
d = 10;
Console.WriteLine("d=" + d);
```

Завдання 6. Створити консольний додаток. Запустіть його. Який результат?

```
var a = 5;
Console.WriteLine("a=" + a);
var b = "bbb";
Console.WriteLine("b=" + b);
var c = a;
Console.WriteLine("c=" + c);
var d = "aaa";
Console.WriteLine("d=" + d);
d = 10;
Console.WriteLine("d=" + d);
```

Завдання 7. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
static void Main(string[] args)
{
    dynamic a;
    a = 10;
    Console.WriteLine("a=" + a);

    a = "aaa";
    Console.WriteLine("a=" + a);

    a = 3.1;
    Console.WriteLine("a=" + a);
    Console.WriteLine("a.SomeMethod = " + a.SomeMethod());

    Console.ReadKey(true);
}
```

Завдання 8. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
static void Main(string[] args)
{
    var a;
    a = 10;
    Console.WriteLine("a=" + a);

    a = "aaa";
    Console.WriteLine("a=" + a);

    a = 3.1;
    Console.WriteLine("a=" + a);
    Console.WriteLine("a.SomeMethod = " + a.SomeMethod());

    Console.ReadKey(true);
}
```

Завдання 9. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
static void Main(string[] args)
{
    object a;
    a = 10;
    Console.WriteLine("a=" + a);

    a = "aaa";
    Console.WriteLine("a=" + a);

    a = 3.1;
    Console.WriteLine("a=" + a);
    Console.WriteLine("a.SomeMethod = " + a.SomeMethod());

    Console.ReadKey(true);
}
```

Завдання 10. Створити консольний додаток. Запустіть його. Який результат?

```
static void Main(string[] args)
{
    var a = new { Name = "Ivan", Age = 17 };

    Console.WriteLine("a.Name=" + a.Name);

    Console.ReadKey(true);
}
```

Завдання 11. Створити консольний додаток. Запустіть його. Який результат?

```
static void Main(string[] args)
{
    dynamic a = new { Name = "Ivan", Age = 17 };

    Console.WriteLine("a.Name=" + a.Name);

    Console.ReadKey(true);
}
```

Завдання 12. Створити консольний додаток. Запустіть його. Який результат?

```
static void Main(string[] args)
{
    object a = new { Name = "Ivan", Age = 17 };

    Console.WriteLine("a.Name=" + a.Name);

    Console.ReadKey(true);
}
```

Завдання 13. Створити консольний додаток. Запустіть його. Який результат?

```
static void Main(string[] args)
{
    var a = 2;
    a += 4;
    Console.WriteLine("a=" + a);

    Console.ReadKey(true);
}
```

Завдання 14. Створити консольний додаток. Запустіть його. Який результат?

```
static void Main(string[] args)
{
    dynamic a = 2;
    a += 4;
    Console.WriteLine("a=" + a);

    Console.ReadKey(true);
}
```

Завдання 15. Створити консольний додаток. Запустіть його. Який результат?

```
static void Main(string[] args)
{
    object a = 2;
    a += 4;
    Console.WriteLine("a=" + a);

    Console.ReadKey(true);
}
```

Завдання 16. Які висновки можна зробити з запитань 1-3?
Завдання 17. Які висновки можна зробити з запитань 4-6?
Завдання 18. Які висновки можна зробити з запитань 7-9?
Завдання 19. Які висновки можна зробити з запитань 10-15?

Завдання на порівняння структур і класів в мові C#

Завдання 20. Чи можна створити конструктор без параметрів? Створити консольний додаток. Запустіть його. Який результат?

```
struct Student
{
    int age;
    string name;

    0 references
    public Student()
    {
        age = 16;
        name = "Ivan";
    }
}
```

Завдання 21. Чи можна створити конструктор без параметрів? Створити консольний додаток. Запустіть його. Який результат?

```
class Student
{
    int age;
    string name;

    0 references
    public Student()
    {
        age = 16;
        name = "Ivan";
    }
}
```

Завдання 22. Чи можна ініціалізувати значення полів в місці їх оголошення? Створити консольний додаток. Запустіть його. Який результат?

```
struct Student
{
    int age = 16;
    string name;
}
```

Завдання 23. Чи можна ініціалізувати значення полів в місці їх оголошення? Створити консольний додаток. Запустіть його. Який результат?

```
class Student
{
    int age = 16;
    string name;
}
```

Завдання 24. Чи можна створити конструктор з параметрами? Створити консольний додаток. Запустіть його. Який результат?

```
struct Student
{
    int age;
    string name;
    0 references
    public Student(string nn)
    {
        name = nn;
        age = 16;
    }
}
```

Завдання 25. Чи можна створити конструктор з параметрами? Створити консольний додаток. Запустіть його. Який результат?

```
class Student
{
    int age;
    string name;
    0 references
    public Student(string nn)
    {
        name = nn;
        age = 16;
    }
}
```

Завдання 26. Чи можна в конструкторі ініціалізувати не усі поля? Створити консольний додаток. Запустіть його. Який результат?

```
struct Student
{
    int age;
    string name;
    0 references
    public Student(int n)
    {
        age = n;
    }
}
```

Завдання 27. Чи можна в конструкторі ініціалізувати не усі поля? Створити консольний додаток. Запустіть його. Який результат?

```
class Student
{
    int age;
    string name;
    0 references
    public Student(int n)
    {
        age = n;
    }
}
```

Завдання 28. Створити консольний додаток. Запустіть його. Який результат?

```
struct Student
{
    public int age;
    public string name;
    1 reference
    public Student(int a, string n)
    {
        age = a;
        name = n;
    }
}
    0 references
    internal class Program
    {
        0 references
        static void Main()
        {
        Student st1 = new Student();
        Student st2 = new Student(16, "Ivan");
        Student st3;
        st3.name = "Kolya";

        Console.WriteLine("st1: name {0}, age: {1}", st1.name,st1.age);
        Console.WriteLine("st2: name {0}, age: {1}", st2.name, st2.age);
        Console.WriteLine("st3: name {0}, age: {1}", st3.name, st3.age);
        Console.ReadKey(true);
        }
    }
```

Завдання 29. Створити консольний додаток. Запустіть його. Який результат?

```
class Student
{
    public int age;
    public string name;
    2 references
    public Student(int a, string n)
    {
        age = a;
        name = n;
    }
}
    0 references
    internal class Program
    {
        0 references
        static void Main()
        {
        Student st1 = new Student();
        Student st2 = new Student(16, "Ivan");
        Student st3;
        st3.name = "Kolya";

        Console.WriteLine("st1: name {0}, age: {1}", st1.name,st1.age);
        Console.WriteLine("st2: name {0}, age: {1}", st2.name, st2.age);
        Console.WriteLine("st3: name {0}, age: {1}", st3.name, st3.age);
        Console.ReadKey(true);
        }
    }
```

Завдання 30. Створити консольний додаток. Запустіть його. Який результат?

```
struct Student
{
    public int age;
    public string name;
    0 references
    public Student(int a, string n)
    {
        age = a;
        name = n;
    }
}
0 references
internal class Program
{
    0 references
    static void Main()
    {
        Student st = null;
        Console.ReadKey(true);
    }
}
```

Завдання 31. Створити консольний додаток. Запустіть його. Який результат?

```
class Student
{
    public int age;
    public string name;
    0 references
    public Student(int a, string n)
    {
        age = a;
        name = n;
    }
}
0 references
internal class Program
{
    0 references
    static void Main()
    {
        Student st = null;
        Console.ReadKey(true);
    }
}
```

Завдання 32. Створити консольний додаток. Запустіть його. Який результат?

```
struct Student
{
    public int age;
}

3 references
class Student1
{
    public int age;
}

0 references
internal class Program
{
    0 references
    static void Main()
    {
        Student st1 = new Student() { age = 17 };
        Student st2 = st1;
        st2.age = 20;
        Console.WriteLine("Struct: st1.age={0}, st2.age={1}", st1.age, st2.age);

        Student1 st3 = new Student1() { age = 17 };
        Student1 st4 = st3;
        st4.age = 20;
        Console.WriteLine("Class: st3.age={0}, st4.age={1}", st3.age, st4.age);

        Console.ReadKey(true);
    }
}
```

Завдання 33. Які висновки можна зробити з запитань 20 та 21?
Завдання 34. Які висновки можна зробити з запитань 22 та 23?
Завдання 35. Які висновки можна зробити з запитань 24 та 25?
Завдання 36. Які висновки можна зробити з запитань 26 та 27?
Завдання 37. Які висновки можна зробити з запитань 28 та 29?
Завдання 38. Які висновки можна зробити з запитань 30 та 31?
Завдання 39. Які висновки можна зробити з запитання 13?

Завдання на опанування властивостями в мові C#

Завдання 40. Створити консольний додаток. Запустіть його. Який результат?

```
2 references
class Square /*квадрат*/
{
    private double side;  /*довжина сторони*/
    2 references
    public double Area  /*площа*/
    {
        get { return side * side; }
        set { side = Math.Sqrt(value); }
    }
}

0 references
internal class Program
{
    0 references
    static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Square t = new Square();
        //Призначення властивості площі ініціює виклик методу set
        t.Area = 25;
        //Звернення до властивості Area ініціює виклик методу get
        Console.WriteLine("Площа квадрата: " + t.Area);
        Console.ReadKey();
    }
}
```

Завдання 41. Використання Automatic Properties. Створити консольний додаток. Запустіть його. Який результат?

```
2 references
class Student
{
    //Автоматично реалізовані властивості (Auto-implemented properties)
    //на випадок примітивних get і set

    // ініціалізація автоматично реалізованих
    //властивостей можлива, починаючи з версії C# 6
    1 reference
    public string Name { get; set; } = "Іван";
    2 references
    public int Age { get; set; } = 21;
}

0 references
internal class Program
{
    0 references
    static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Student st = new Student();
        st.Age = 18;
        Console.WriteLine("Ім'я студента: " + st.Name);
        Console.WriteLine("Вік студента: " + st.Age);
        Console.ReadKey();
    }
}
```

Завдання 42. Властивість можна зробити лише для читання, лише надавши доступ get. Створити консольний додаток. Запустіть його. Який результат?

```
2 references
class Student
{
    public string name = "Іван";
    private int age = 18;
    1 reference
    public string Name
    {
        get { return name; }
    }
    2 references
    public int Age
    {
        get { return age; }
    }
}

0 references
internal class Program
{
    0 references
    static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Student st = new Student();
        st.Age = 18;
        Console.WriteLine("Ім'я студента: " + st.Name);
        Console.WriteLine("Вік студента: " + st.Age);
        Console.ReadKey();
    }
}
```

Завдання 43. Властивість можна зробити лише для запису, лише надавши доступ set. Створити консольний додаток. Запустіть його. Який результат?

```
2 references
class Student
{
    private string name = "Іван";
    private int age;
    2 references
    public string Name
    {
        set { name = value; }
    }
    2 references
    public int Age
    {
        set { age = value; }
    }
}

0 references
internal class Program
{
    0 references
    static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Student st = new Student();
        st.Age = 18;
        st.Name = "Іван";
        Console.WriteLine("Ім'я студента: " + st.Name);
        Console.WriteLine("Вік студента: " + st.Age);
        Console.ReadKey();
    }
}
```

Завдання 44. Чи можемо ми визначити властивості як дві різні множини? Створити консольний додаток. Запустіть його. Який результат?

```
2 references
class Student
{
    private string name;
    2 references
    public string Name
    {
        set { name = value; }
    }

    2 references
    public string Name
    {
        get { return name; }
    }

}

0 references
internal class Program
{
    0 references
    static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Student st = new Student();
        st.Name = "Іван";
        Console.WriteLine("Ім'я студента:" + st.Name);
        Console.ReadKey();
    }
}
```

Завдання 45. Чи можемо ми визначити властивості з такими ж іменами, як уже визначена змінна? Створити консольний додаток. Запустіть його. Який результат?

```
2 references
class Student
{
    private string name;
    4 references
    public string name
    {
        set { name = value; }
        get { return name; }
    }


}

0 references
internal class Program
{
    0 references
    static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Student st = new Student();
        st.name = "Іван";
        Console.WriteLine("Ім'я студента: " + st.name);
        Console.ReadKey();
    }
}
```

Завдання 46. Чи може властивість бути позначено як статичну? Створити консольний додаток. Запустіть його. Який результат?

```
2 references
class Student
{
    private static string name;
    2 references
    public static string Name
    {
        set { name = value; }
        get { return name; }
    }
}

0 references
internal class Program
{
    0 references
    static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Student.Name = "Іван";
        Console.WriteLine("Ім'я студента: " + Student.Name);
        Console.ReadKey();
    }
}
```

Завдання 47. Які висновки можна зробити? Виберіть усі правильні відповіді.

- ☐ Get використовується для зчитування та запису значення властивості.
- ☐ Змінна, що використовується для властивості, повинна бути того ж типу даних, що і тип даних властивості.
- ☐ Властивість може мати тип результату void.
- ☐ Властивість не може мати тип результату void.
- ☐ Get використовується лише для зчитування значення властивості. Властивість, яка має лише get, не може бути встановлена жодним значенням від користувача.

Завдання на опанування перерахувань (enum) в мові C#

Завдання 48. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Wage wage = Wage.Small;
        Console.WriteLine("Wage is {0}", wage);
        Console.WriteLine("Wage is {0}", (int)wage);
        Console.ReadKey();
    }
}

2 references
enum Wage
{
    Small,
    Medium,
    Pretty_med,
    Large
}
```

Завдання 49. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Wage wage = Wage.Small;
        ClothesSize size = ClothesSize.L;
        Console.WriteLine("Wage is {0}", wage);
        Console.WriteLine("Wage is {0}", (int)wage);
        Console.WriteLine("Size is {0}", size);
        Console.WriteLine("Size is {0}", (int)size);
        Console.ReadKey();
    }
}

3 references
enum Wage
{
    Small,
    Medium,
    Pretty_med,
    Large
}

2 references
enum ClothesSize : Wage
{
    XS,
    S,
    M,
    L,
    XL
}
```

Завдання 50. Створити консольний додаток. Запустіть його. Який результат?

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Wage wage = Wage.Small;
        Console.WriteLine("Wage is {0}", wage);
        Console.WriteLine("Wage is {0}", (int)wage);
        Console.ReadKey();
    }
}

3 references
enum Wage
{
    Small,
    Medium,
    Pretty_med,
    Large
}

0 references
class ClothesSize : Wage
{

}
```

Завдання 51. Створити консольний додаток. Запустіть його. Який результат?

```
8 references
enum Wage
{
    Small,
    Medium,
    Pretty_med,
    Large
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        int wage = 1;
        Console.WriteLine(Wage.Medium.CompareTo(Wage.Small));
        Console.WriteLine(Wage.Medium.CompareTo(Wage.Medium));
        Console.WriteLine(Wage.Medium.CompareTo(Wage.Large));

        if (wage == Wage.Medium)
        {
            Console.WriteLine("true");
        }

        if (wage == (int)Wage.Medium)
        {
            Console.WriteLine("true");
        }

        Console.ReadKey();
    }
}
```

Завдання 52. Створити консольний додаток. Запустіть його. Який результат?

```
4 references
enum Wage
{
    Small = 90,
    Medium,
    Pretty_med = 100,
    Large
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        string[] names;
        names = Wage.GetNames(typeof(Wage));
        foreach (var name in names)
        {
            Console.WriteLine(name);
        }

        Console.WriteLine((int)Wage.Medium);
        Console.WriteLine((int)Wage.Large);
        Console.ReadLine();
    }
}
```

Завдання 53. Створити консольний додаток. Запустіть його. Який результат?

```
2 references
enum Wage
{
    Small = 90,
    Medium,
    Pretty_med = Small,
    Large
}

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine((int)Wage.Medium);
        Console.WriteLine((int)Wage.Large);
        Console.ReadLine();
    }
}
```

Завдання 54. Створити консольний додаток. Запустіть його. Який результат?

```
7 references
internal enum Wage
{
    Small,
    Medium,
    Large
}
0 references
internal class Program
{
    0 references
    private static void Main(string[] args)
    {
        Console.WriteLine(CheckWage(Wage.Small));
        Console.WriteLine(CheckWage(Wage.Large));
        Console.WriteLine(CheckWage(Wage.Medium));
        Console.ReadKey();
    }

    3 references
    public static string CheckWage(Wage wage)
    {
        switch (wage)
        {
            case Wage.Small:
                return "Small";
            case Wage.Medium:
                return "Medium";
            case Wage.Large:
                return "Large";
            default:
                return "no wage";
        }
    }
}
```

Завдання 55. Створити консольний додаток. Запустіть його. Який результат?

```
7 references
internal enum Wage
{
    Small,
    Medium,
    Large
}
0 references
internal class Program
{
    0 references
    private static void Main(string[] args)
    {
        Console.WriteLine(CheckWage((Wage)0));
        Console.WriteLine(CheckWage((Wage)2));
        Console.WriteLine(CheckWage((Wage)1));
        Console.ReadKey();
    }

    3 references
    public static string CheckWage(Wage wage)
    {
        switch (wage)
        {
            case Wage.Small:
                return "Small";
            case Wage.Medium:
                return "Medium";
            case Wage.Large:
                return "Large";
            default:
                return "no wage";
        }
    }
}
```

Завдання 56. Які висновки можна зробити? Виберіть усі правильні відповіді.

- ☐ Значення перелічного типу можна змінити після ініціалізації.
- ☐ Перелічний тип не може бути похідним від будь-якого іншого типу, окрім типу байт, sbyte, короткий, ushort, int, uint, long або ulong.
- ☐ Більше одного члена enum не може ініціалізувати одне і те ж постійне значення.
- ☐ Методи перетворення не можуть використовуватися для перетворення enum з одного типу даних в інший.
- ☐ Численні заздалегідь визначені методи перетворення можуть використовуватися для перетворення enum з одного типу даних в інший.
- ☐ Класи можуть успадковуватися від enum.
- ☐ Більше одного члена enum може ініціалізувати одне і те ж постійне значення.
- ☐ Перелічний тип діє як константа, тому його значення не можна змінити після ініціалізації.
- ☐ Перелічний тип може бути похідним від довільного типу.

Запитання до теми:

1. Що таке виключення?
2. Що таке конструкція `try - catch`?
3. Що таке конструкція `try - catch - finally`?
4. Як створити користувацьке виключення?
5. Що таке анонімні типи?
6. Що таке динамічний тип?
7. Що робить оператор `dynamic`?

ТЕОРЕТИЧНА ЧАСТИНА ЛАБОРАТОРНОГО ЗАНЯТТЯ №5

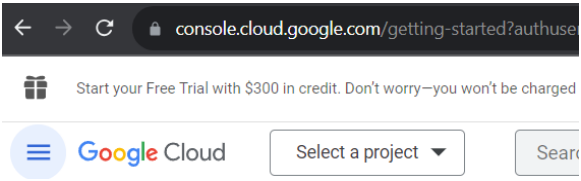
ІНТЕГРАЦІЯ З СЕРВІСАМИ GOOGLE

Огляд, мета і призначення теми: навчитися використовувати сервіси Google у програмних рішеннях. Реалізувати авторизацію користувача та вивантаження/завантаження файлів у хмарне сховище Google Drive.

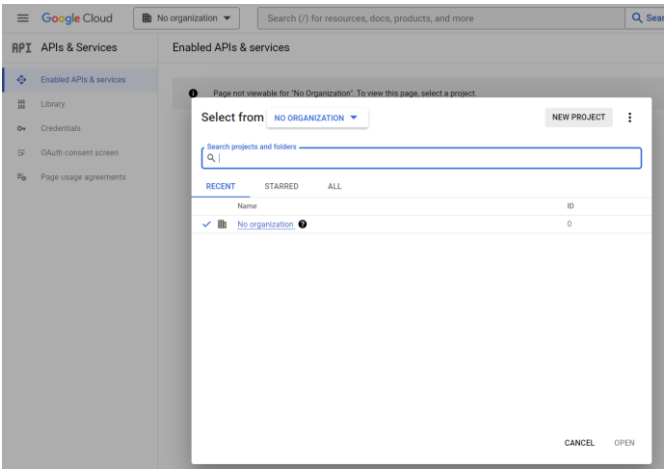
Google надає екосистему сервісів, що містять відкриті прикладні програмні інтерфейси (API) для інтеграції програмних продуктів. В рамках виконання лабораторних робіт до курсу, студенти мають можливість познайомитися з технологіями взаємодії з веб API сервісів G
o
o У даному додатку наведені підходи до отримання облікових g даних Google та взаємодії з сервісами Google від імені l автентифікованого користувача.
e

Увага! Кроки наведені в цьому розділі підходять лише для д есктопних (Windows, MacOS) реліз-версій MAUI застосунку. Окрім цього, механізм отримання облікових даних є спрощеним і не р екомендованим до використання у опублікованих застосунках. Р екомендовані підходи до отримання облікових даних будуть розглядатися у курсі «Інструментальні середовища та технології м рограмування» і будуть включати в себе реалізацію бекенд-сервера для застосунку.
в
и **Реєстрація застосунку Google.** Google, як і більшість сервісів, к що публікують відкриті API для інтеграції, вимагає від третьої сторони (розробників) реєстрації застосунків для подальшого використання р ервісів Google від імені користувача. На порталі Google Cloud Console https://console.cloud.google.com/ розробникам пропонується зареєструвати свій проєкт для запиту необхідних прав доступу та к лючів. В рамках даної демонстрації, пропонується пройти процес р реєстрації проєкту в Google Cloud Console.
н Перейдіть на сторінку порталу у браузері, пройдіть процедуру р реєстрації (з особистого @gmail.com акаунту, це безкоштовно) та в відкрийте панель вибору проєкту («Select a project»):

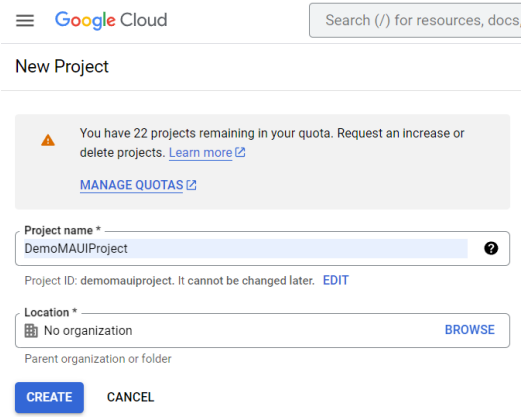
в
і
д
к
р
и
т
и



У панелі вибору проєкту оберіть створення нового проєкту («New project»):

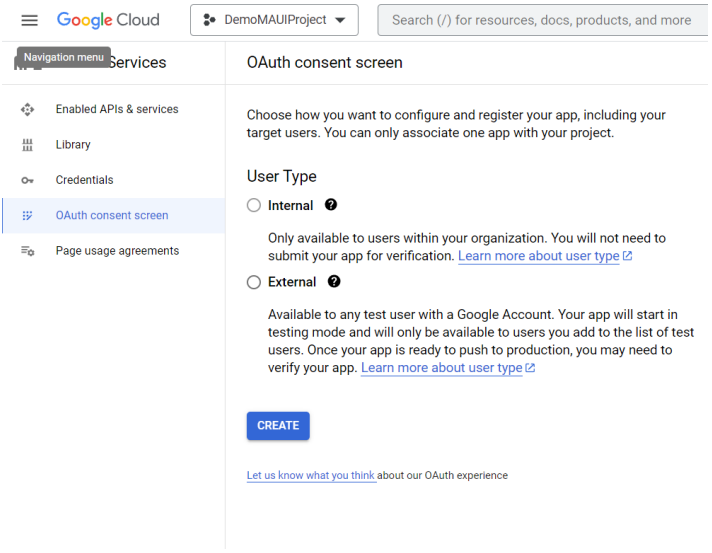


Введіть назву проєкту. Пункт Location залиште в значенні «No organization»:

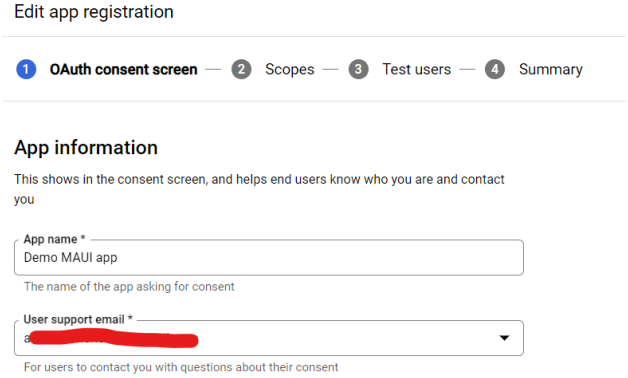


Після створення проєкту необхідно налаштувати вікно передачі облікових даних через OAuth. Спрощено, дане налаштування вказуватиме порядок передачі облікових даних користувачів застосунку та рівні доступу до особистих даних користувачів, на які вони матимуть погоджуватися перед використанням.

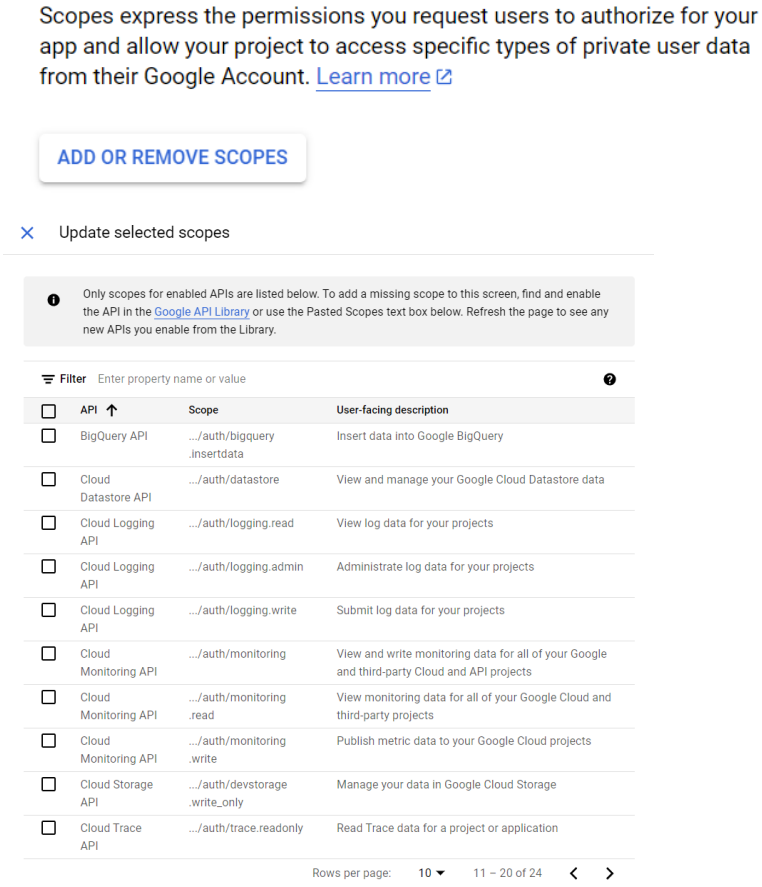
На панелі швидкого доступу оберіть пункт «OAuth consent screen». Тип користувачів оберіть External (оскільки ваш персональний Google акаунт не є частиною організації, і зовнішні користувачі мають мати доступ до нього) та натисніть Create:



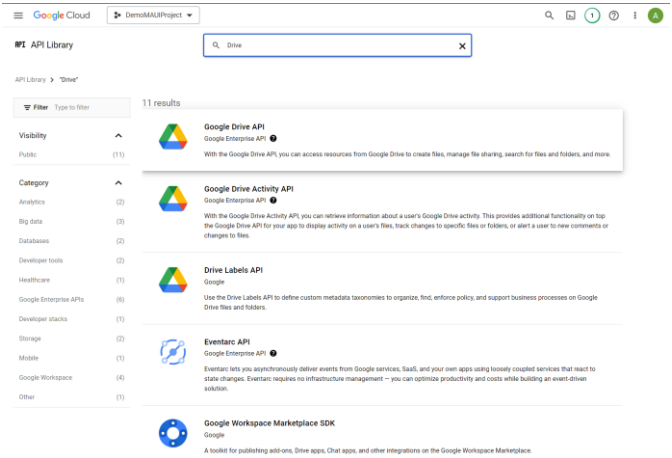
Заповніть ім’я вашого застосунку та свою електронну адресу. Необов’язкові поля можна залишити порожніми:



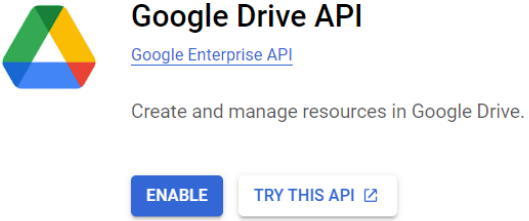
Далі, необхідно зазначити список прав доступу для застосунку. Натисніть «Add or Remove scopes».



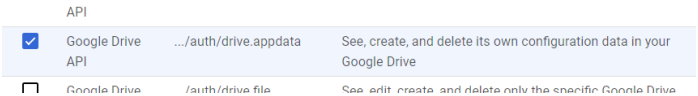
Оскільки новостворений проєкт не має за замовчуванням доступу до більшості сервісів Google, цей доступ необхідно запросити окремо. Для цього, на панелі швидкого доступу оберіть пункт «Library». Використовуючи пошук, знайдіть потрібний сервіс Google Drive:



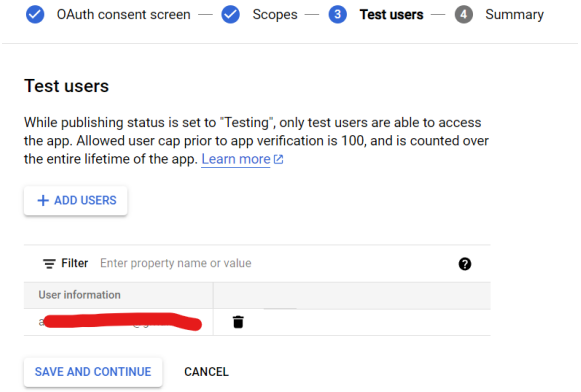
Оберіть знайдений сервіс та натисніть Enable:



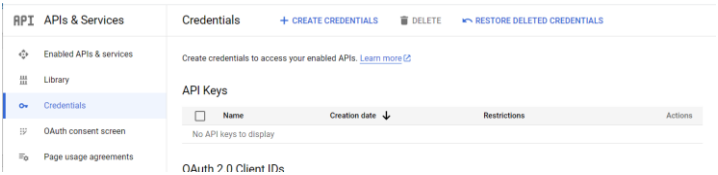
Поверніться до пункту вибору прав доступу. В рамках даної демонстрації, буде використовуватися доступ до виділеної папки додатку на користувацькому Google Drive (для кожного застосунку, що використовує Google, при встановленні створюється така папка). Для доступу до цієї, застосунку потрібні права доступу https://www.googleapis.com/auth/drive.appdata. Знайдіть у пошуку та додайте:



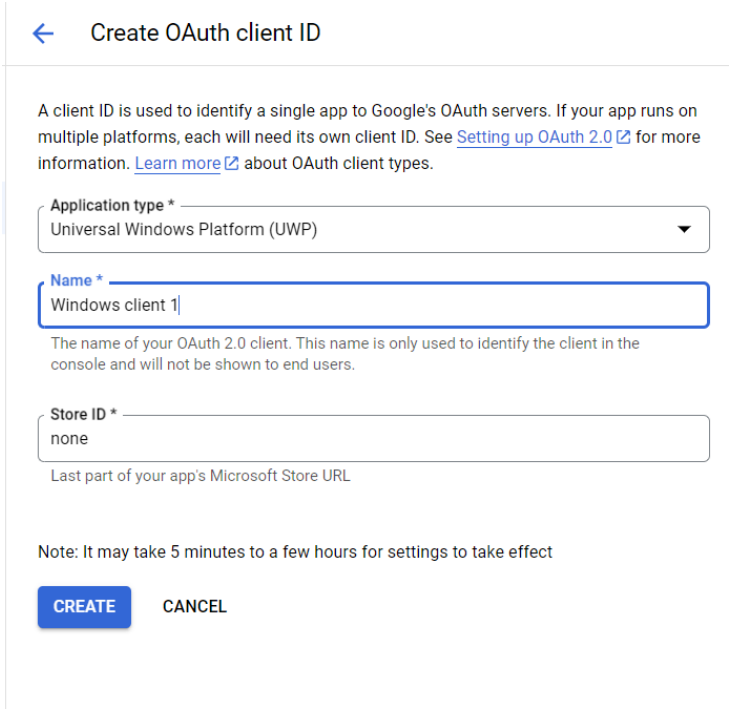
Далі, додайте тестових користувачів – електронні адреси користувачів Google які матимуть можливість встановлювати (надавати доступ) застосунку до публікації/верифікації.



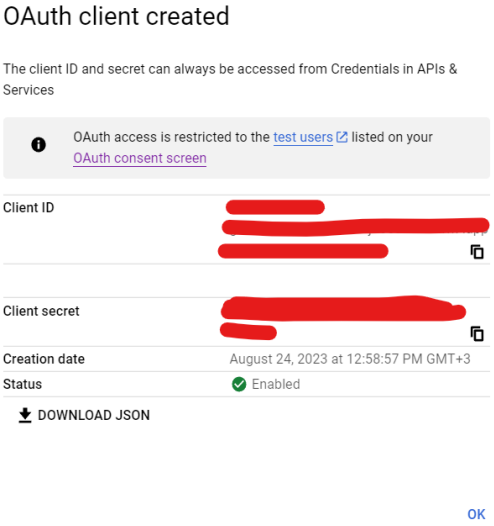
Після налаштування вікна OAuth, необхідно створити облікові дані, за допомогою яких застосунок зможе надсилати запити на отримання доступу від імені користувачів. На панелі швидкого доступу, оберіть пункт «Credentials». Натисніть «Create credentials»:



Оберіть тип вашого застосунку. Оскільки цільовою платформою застосунку в рамках цього розділу є Windows, оберіть «Universal Windows Platform». Далі необхідно вказати назву для облікових даних, а також Store Id (до публікації – може бути будь-яким).



Після створення, скачайте файл з обліковими даними натиснувши «Download JSON». **Увага!** Не публікуйте цей файл.



Взаємодія з Google у .NET. Для первинного ознайомлення з будь-якою технологією чи бібліотекою, зазвичай створюють так званий «Hello World» проєкт. В даній частині, буде наведено кроки для створення консольного застосунку, який створює файл у сховищі застосунку Google Drive, виводить список доступних файлів в сховищі та виводить контент файлу, що зберігається у хмарі.

Першим кроком після створення проєкту є встановлення залежностей. Google надає SDK, який доступний у вигляді пакетів у пакетному менеджері NuGet. Для даного прикладу, необхідно завантажити пакети:

Для реалізації механізму входу користувача в Google аккаунт та надання застосунку

М

```
// використовуючи файл, завантажений з попереднього розділу
```

```
// використовуючи дані з завантаженого файлу з попереднього розділу явно (не рекомендовано)
```

Т

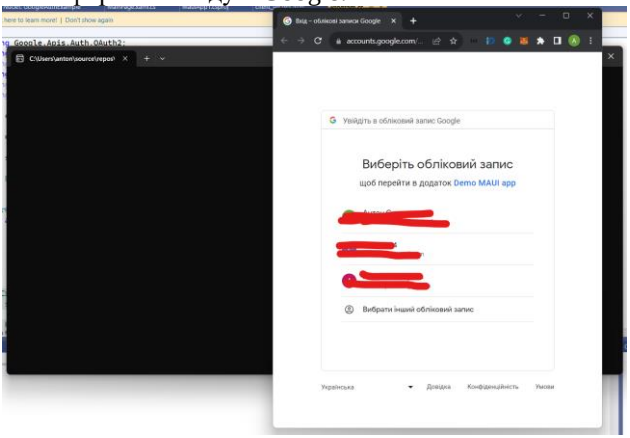
Процес авторизації потребує вказання списку прав доступу. Забігаючи наперед, кожен клас-сервіс з Google SDK містить колекцію назв прав доступу, що відносяться до цього сервісу. Для означення прав доступу https://www.googleapis.com/auth/drive.appdata необхідно використати наступний код:

П

Таким чином, виклик методу виглядатиме наступним чином:

застосунку для ініціалізації сесії входу. Можна використати декілька

При виклику даного методу, буде відкрито вкладку браузера за замовчуванням з формою входу в Google:



Для доступу до файлів користувача

Отримання списку файлів у папці на Google Drive. Для отримання списку файлів використовується об’єкт класу запиту на

Метод повертає об’єкт класу запиту, який можна модифікувати, виконувати та отримувати результати виконання. В рамках демонстраційної програми

Д

Д

Таким же чином реалізуються операції запису та читання файлів. Повний код демонстраційної програми наведено нижче:

У прикладі одразу використовуються асинхронні версії методів бібліотеки. При перенесенні у MAUI застосунок, це дозволить не блокувати основний потік виконання і не «блокувати» інтерфейс.

Повну документацію щодо API Google Drive можна знайти за посиланням:
https://developers.google.com/drive/api/guides/about-sdk

JSON - ЗАГАЛЬНА ІНФОРМАЦІЯ

JSON (JavaScript Object Notation) є одним із найбільш популярних форматів для зберігання та передачі даних. Основна функціональність роботи з JSON зосереджена у просторі імен System.Text.Json. Ключовим типом є клас JsonSerializer, який дозволяє серіалізувати об'єкт в json і десеріалізувати код json в об'єкт C#.
Деякі атрибути, які допомагають працювати з перетвореннями:

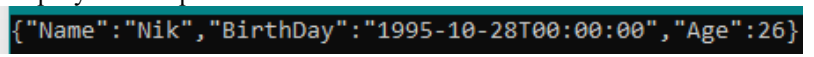
- JsonPropertyName - вказати назву властивості для формату json, не змінюючи саму назву
- JsonIgnore - не серіалізувати властивість
- JsonPropertyOrder - визначає порядок властивостей, присутній у json під час серіалізації. Менші значення серіалізуються першими. Якщо атрибут не вказано, значенням за замовчуванням є 0.

Приклад 1

Виконаємо наступний код

```
class Person
{
    1 reference
    public string Name { get; }
    3 references
    public DateTime BirthDay { get; set; }
    0 references
    public int Age
    {
        get
        {
            var diffYear = DateTime.Today.Year - BirthDay.Year;
            return BirthDay.AddYears(diffYear) > DateTime.Today
                ? diffYear - 1
                : diffYear;
        }
    }
    1 reference
    public Person(string name, DateTime birthDay)
    {
        Name = name;
        BirthDay = birthDay;
    }
}
var person = new Person("Nik", new DateTime(1995, 10, 28));
Console.WriteLine(System.Text.Json.JsonSerializer.Serialize(person));
```

Як результат отримаємо

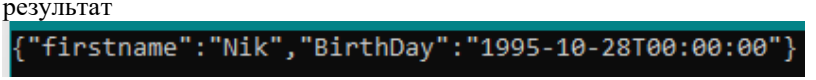


Приклад 2

Додамо використання атрибутів

```
class Person
{
    [JsonPropertyName("firstname")]
    1 reference
    public string Name { get; }
    3 references
    public DateTime BirthDay { get; set; }
    [JsonIgnore]
    0 references
    public int Age
    {
        get
        {
            var diffYear = DateTime.Today.Year - BirthDay.Year;
            return BirthDay.AddYears(diffYear) > DateTime.Today
                ? diffYear - 1
                : diffYear;
        }
    }
    1 reference
    public Person(string name, DateTime birthDay)
    {
        Name = name;
        BirthDay = birthDay;
    }
}
```

Виконавши код з попереднього прикладу отримаємо наступний результат



JsonSerializerOptions

За замовчуванням JsonSerializer серіалізує об'єкти у мініміфікованому коді. За допомогою додаткового параметра JsonSerializerOptions можна налаштувати механізм серіалізації/десеріалізації, використовуючи властивості JsonSerializerOptions. Деякі з його властивостей:

- AllowTrailingCommas: встановлює, чи потрібно додавати після останнього елемента в JSON кому. Якщо true => кома додається
- DefaultIgnoreCondition: встановлює, чи серіалізуватиметься/десеріалізуватиметься в JSON властивості зі значеннями за умовчанням
- IgnoreReadOnlyProperties: аналогічно встановлює, чи серіалізуватимуться властивості, призначені тільки для читання
- IncludeFields:отримує або встановлює значення, яке вказує, чи обробляються поля під час серіалізації та десеріалізації
- WriteIndented: встановлює, чи додаватимуться в JSON пробіли (умовно кажучи, для краси). Якщо true => встановлюються додаткові прогалини

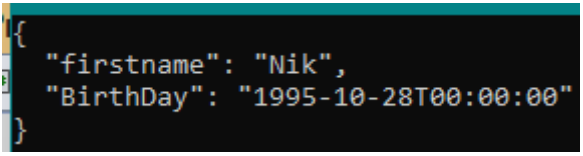
Приклад 3

Передамо наступні параметри і змінимо клас Person

```
var options = new JsonSerializerOptions
{
    WriteIndented = true
};
var person = new Person("Nik", new DateTime(1995, 10, 28));
Console.WriteLine(System.Text.Json.JsonSerializer.Serialize(person, options));
```

```
2 references
class Person
{
    [JsonPropertyName("firstname")]
    1 reference
    public string Name { get; }
    3 references
    public DateTime BirthDay { get; set; }
    public int Age;
    1 reference
    public Person(string name, DateTime birthDay)
    {
        Name = name;
        BirthDay = birthDay;
        var diffYear = DateTime.Today.Year - BirthDay.Year;
        Age = BirthDay.AddYears(diffYear) > DateTime.Today
                    ? diffYear - 1
                    : diffYear;
    }
}
```

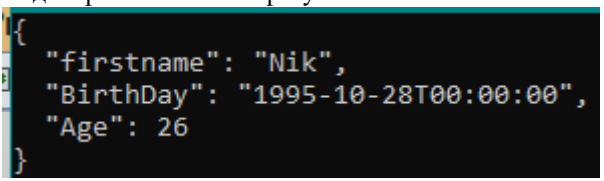
Отримаємо наступний результат



Щоб серіалізувати параметр Age, необхідно змінити JsonSerializerOptions:

```
var options = new JsonSerializerOptions
{
    WriteIndented = true,
    IncludeFields = true
};
```

Тоді отримаємо інший результат:



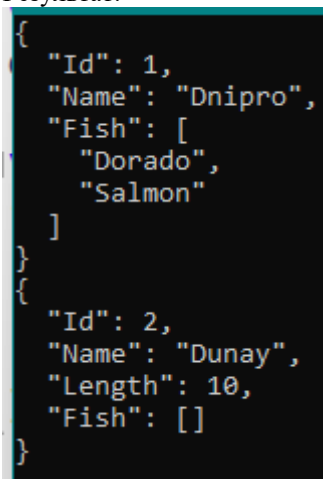
Приклад 4

```
public class River
{
    2 references
    public River(int id, string name)
    {
        Id = id;
        Name = name;
        Fish = new HashSet<string>();
        _fishCount = new Dictionary<string, int>();
    }

    [JsonPropertyOrder(10)]
    2 references
    public HashSet<string> Fish { get; private set; }

    1 reference
    public int Id { get; private set; }
    1 reference
    public string Name { get; private set; }

    1 reference
    public decimal Length { get; set; }

    private Dictionary<string, int> _fishCount;

    3 references
    public void AddFish(string fishName)
    {
        if (Fish.Add(fishName))
        {
            _fishCount[fishName] = 1;
            return;
        }
        _fishCount[fishName] = _fishCount[fishName] + 1;
    }

    0 references
    public void WriteCountByFish(string fishName)
    {
        var fishCount = _fishCount.TryGetValue(fishName, out int count) ? count : 0;
        Console.WriteLine($"{fishName} = {fishCount}");
    }
}
```

Виконаємо наступний код

```
var dnipro = new River(1, "Dnipro");
dnipro.AddFish("Dorado");
dnipro.AddFish("Salmon");
dnipro.AddFish("Dorado");

var dunay = new River(2, "Dunay");
dunay.Length = 10m;

var options = new JsonSerializerOptions
{
    WriteIndented = true,
    IncludeFields = true,
    DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingDefault
};

Console.WriteLine(System.Text.Json.JsonSerializer.Serialize<River>(dnipro, options));
Console.WriteLine(System.Text.Json.JsonSerializer.Serialize<River>(dunay, options));
```

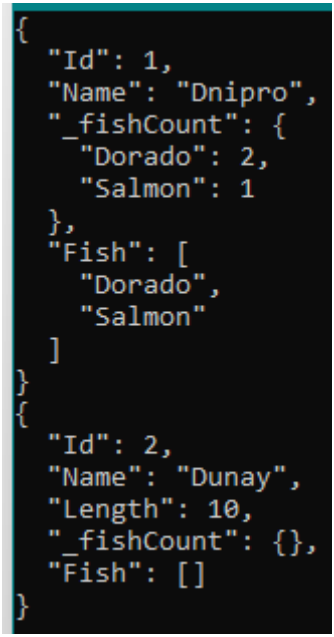
Результат:



Якщо ж для _fishCount вказати модифікатор доступу public

```
public Dictionary<string, int> _fishCount;
```

То буде наступний результат:



Запис у файл та зчитування інформації із файла

```
public class River
{
    2 references
    public River(int id, string name)
    {
        Id = id;
        Name = name;
        Fish = new HashSet<string>();
        _fishCount = new Dictionary<string, int>();
    }

    [JsonPropertyOrder(10)]
    2 references
    public HashSet<string> Fish { get; private set; }

    1 reference
    public int Id { get; private set; }
    1 reference
    public string Name { get; private set; }

    1 reference
    public decimal Length { get; set; }

    private Dictionary<string, int> _fishCount;

    3 references
    public void AddFish(string fishName)
    {
        if (Fish.Add(fishName))
        {
            _fishCount[fishName] = 1;
            return;
        }
        _fishCount[fishName] = _fishCount[fishName] + 1;
    }

    0 references
    public void WriteCountByFish(string fishName)
    {
        var fishCount = _fishCount.TryGetValue(fishName, out int count) ? count : 0;
        Console.WriteLine($"{fishName} = {fishCount}");
    }
}
```

Приклад “Запис у файл”

Виконати наступний код

```
var rivers = new List<River>();
var dnipro = new River(1, "Dnipro");
dnipro.AddFish("Dorado");
dnipro.AddFish("Salmon");
dnipro.AddFish("Dorado");

var dunay = new River(2, "Dunay");
dunay.Length = 10m;
rivers.Add(dnipro);
rivers.Add(dunay);

var options = new JsonSerializerOptions
{
    WriteIndented = true,
    DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingDefault
};

string path = @"C:\Test2\SerializeJsonExample.json";
using (FileStream fstream = new FileStream(path, FileMode.Create))
{
    await JsonSerializer.SerializeAsync(fstream, rivers, options);
    Console.WriteLine(@"Записано у файл");
}
```

Файл матиме наступний вигляд:

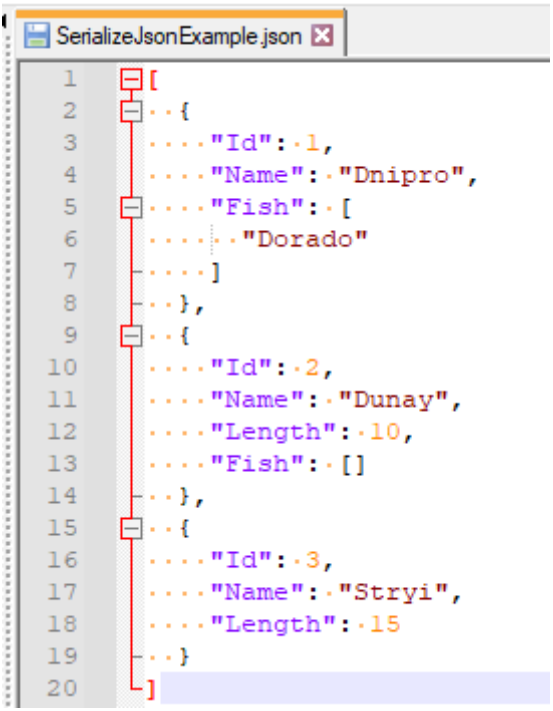
SerializeJsonExample.json

```
[
  {
    "Id": 1,
    "Name": "Dnipro",
    "Fish": [
      "Dorado",
      "Salmon"
    ]
  },
  {
    "Id": 2,
    "Name": "Dunay",
    "Length": 10,
    "Fish": []
  }
]
```

Приклад “Зчитати з файлу”

Запишемо у файл наступну інформацію

SerializeJsonExample.json



```
[
  {
    "Id": 1,
    "Name": "Dnipro",
    "Fish": [
      "Dorado"
    ]
  },
  {
    "Id": 2,
    "Name": "Dunay",
    "Length": 10,
    "Fish": []
  },
  {
    "Id": 3,
    "Name": "Stryi",
    "Length": 15
  }
]
```

Виконаємо код:

```
string path = @"C:\Test2\SerializeJsonExample.json";
using (FileStream fs = new FileStream(path, FileMode.Create))
{
    var rivers = await JsonSerializer.DeserializeAsync<List<River>>(fs);
    foreach (var river in rivers)
    {
        Console.WriteLine($"{river.Id} - {river.Name} - {river.Length} - {river.Fish.Count}");
    }
}
Console.ReadKey();
```

Отримаємо наступний результат:

```
1 - Dnipro - 0 - 0
2 - Dunay - 10 - 0
3 - Stryi - 15 - 0
```

ТИЖДЕНЬ 6

Тема 6. МОВА C#: КЛАСИ, ПЕРЕТВОРЕННЯ ТИПІВ, ОПЕРАТОРИ ТА ВИРАЗИ, ОПЕРАТОРИ ОБРОБКИ ВИКЛЮЧЕНЬ, НЕЯВНІ, АНОНІМНІ ТА ДИНАМІЧНІ ТИПИ ДАНИХ (продовження)

Огляд, мета і призначення теми: навчитися використовувати принципи об'єктно-орієнтованого програмування на прикладі мови програмування C#. Класи, перетворення типів, оператори та вирази, оператори обробки виключень, неявні, анонімні та динамічні типи даних.

Вивчивши матеріал даної теми, студент зможе: вміти пояснити відмінності між класами та об’єктами. Знати та розуміти основні парадигми ООП. Розуміти та вміти використовувати перетворення типів, оператори та вирази, оператори обробки виключень, неявні, анонімні та динамічні типи даних в мові C#.Знати основи синтаксичного аналізу та обчислення значень виразів. Вміти застосовувати ANTLR.

Лабораторний практикум 6

Основи синтаксичного аналізу та обчислення виразів. Використання ANTLR для виконання лабораторної роботи №1 (60 хв.), захист домашнього завдання № 4 (30 хв.) (використовуючи матеріали теоретичної частини цього лабораторного заняття (ANTLR) написати фрагмент лабораторної роботи, що відповідає за обчислення виразів у відповідності зі своїм варіантом).

Завдання для опрацювання матеріалу

Завдання 1.

Маємо файл з вмістом:

```
[
  {
    "PublishingHouseId": 2,
    "Title": "Підручник. Алгебра 8",
    "PublishingHouse": {
      "Id": 2,
      "Name": "ГІМНАЗІЯ",
      "Adress": "Адреса 2"
```

```
    }
  },
  {
    "PublishingHouseId": 1,
    "Title": "Щоденник нейрохірурга",
    "PublishingHouse": {
      "Id": 1,
      "Name": "Видавництво старого лева",
      "Adress": "Адреса 1"
    }
  },
  {
    "PublishingHouseId": 2,
    "Title": "Посібник. Алгебра 9",
    "PublishingHouse": {
      "Id": 2,
      "Name": "ГІМНАЗІЯ",
      "Adress": "Адреса 2"
    }
  }
]
```

1. Створити відповідні класи та десеріалізувати вміст файлу
2. Як необхідно змінити опис класів, щоб не серіалізувати в подальшому параметр PublishingHouseId ?
3. Як необхідно змінити опис класів, щоб серіалізований об'єкт містив параметр Title з назвою "Name" ?

Завдання 2. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    var numbers = new List<string>() { "4", "2", "5", "12", "1" };
    foreach(var number in numbers.OrderBy(x => x))
    {
        Console.WriteLine("{0}", number);
    }
    Console.ReadLine();
}
```

Завдання 3. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    List<int> list = new List<int>() { 1, 5, 14 , 9 };
    var nums = from elem in list
               where elem > 7
               select elem;
    list[0] += 10;
    foreach(int num in nums)
    {
        Console.WriteLine(num);
    }
    Console.ReadLine();
}
```

Завдання 4. Створіть консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    List<int> list = new List<int>() { 1, 5, 14 , 9 };
    var nums = (from elem in list
               where elem > 7
               select elem).ToArray();
    list[0] += 10;
    foreach(int num in nums)
    {
        Console.WriteLine(num);
    }
    Console.ReadLine();
}
```

Завдання 5. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
public class Student
{
    public string Name;
}
Ссылок: 0
static void Main(string[] args)
{
    var myStudents1 = new List<Student>
    {
        new Student() {Name ="Alice"},
        new Student() {Name ="Bob"},
        new Student() {Name ="Clarie"},
        new Student() {Name ="Alice"}
    };
    var myStudents2 = new List<Student>
    {
        new Student() {Name ="Alice"},
        new Student() {Name ="Bob"},
        new Student() {Name ="Tim"},
        new Student() {Name ="Alice"}
    };

    var result = myStudents1.Intersect(myStudents2);

    Console.WriteLine("{0}", result.Count());
    Console.ReadLine();
}
```

Завдання 6. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
public class Student
{
    public int Age;
    public string Name;
}
Ссылок: 0
static void Main(string[] args)
{
    var myStudents = new List<Student>
    {
        new Student() {Age =18 , Name ="Alice"},
        new Student() {Age =20 , Name ="Bob"},
        new Student() {Age =18 , Name ="Clarie"},
        new Student() {Age =18 , Name ="Alice"}
    };

    Console.WriteLine(myStudents.Distinct().Count());
    Console.ReadLine();
}
```

Завдання 7. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    var words = new List<string> { "bac", "dbce", "acbfg", "bcad", "acbd" };
    var words1 = words.Where(x=>x.Contains("cb"));
    Console.WriteLine(words1.Count());
    Console.ReadLine();
}
```

Завдання 8. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
public class Student
{
    public string FirstName;
    public string LastName;
}
Ссылок: 0
static void Main(string[] args)
{
    var myStudents = new List<Student>
    {
        new Student {FirstName="Bob" , LastName="Borisov"},
        new Student {FirstName="Adam" , LastName="Cepler"},
        new Student {FirstName="Clark" , LastName="Adler"}
    };

    var studentlist = myStudents.OrderBy(x => x.FirstName)
                                .OrderBy(x => x.LastName)
                                .ToList();

    foreach(var student in studentlist)
    {
        Console.WriteLine("{0} {1}", student.FirstName, student.LastName);
    }
    Console.ReadLine();
}
```

Завдання 9. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
public class Student
{
    public string FirstName;
    public string LastName;
    public List<int> BookId;
}
Ссылок: 0
static void Main(string[] args)
{
    var myStudents = new List<Student>
    {
        new Student {FirstName="Bob" , LastName="Borisov", BookId=new List<int>(){ 1 , 24 } },
        new Student {FirstName="Adam" , LastName="Cepler", BookId=new List<int>(){ 5 , 6 , 12 , 15 }},
        new Student {FirstName="Clark" , LastName="Adler", BookId=new List<int>(){ 7 , 82 , 34 }}
    };

    var studentlist = myStudents.Select(x => x.BookId.Count(y => y > 10)).ToList();
    foreach(var student in studentlist)
    {
        Console.WriteLine("{0}", student );
    }
    Console.ReadLine();
}
```

Завдання 10. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    string[] colors = { "green", "brown", "blue", "red" };
    string s = "g";
    var query = colors.Where(c => c.Contains(s));
    s = "n";
    query = query.Where(c => c.Contains(s));
    Console.WriteLine(query.Count());
    Console.ReadLine();
}
```

Завдання 11. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    string[] colors = { "green", "brown", "blue", "red" };
    var query = colors.Where(c => c.Contains("g"));
    query = query.Where(c => c.Contains("n"));
    Console.WriteLine(query.Count());
    Console.ReadLine();
}
```

Завдання 12. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    string[] colors = { "green", "purple", "red", "yellow" };
    var query = from c in colors
                where c.Length == colors.Average(x => x.Length)
                select c;
    foreach (var color in query)
        Console.WriteLine("{0}", color);
    Console.ReadLine();
}
```

Завдання 13. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    var nameList = new List<string>() { "Adler", "Tomas", "Bob", "Toni", "Xavier" };
    var a = nameList.First(e => e.Contains("To"));
    var b = nameList.Single(e => e.Contains("To"));
    Console.WriteLine(a == b);
    Console.ReadLine();
}
```

Завдання 14. Створити консольний додаток. Запустіть його. Який результат? Поясніть.

```
static void Main(string[] args)
{
    int[] numbers = { 40, 28, 35, 90, 60, 32, 68, 55 };
    var query = numbers.SkipWhile((x, y) => x > y * 10);
    foreach(var num in query)
    {
        Console.WriteLine("{0}", num);
    }
    Console.ReadLine();
}
```

і

Завдання 15. Список студентів потрібно впорядкувати за віком, а при збігу віку - по імені. Як це виконати в LINQ?

Завдання 16. Різниця методів Except , Intersect , Union , Concat

Запитання до теми:

1. Призначення рефакторингу.
2. Види рефакторингу.
3. LINQ (Language Integrated Query – мова інтегрованих запитів). Призначення та синтаксис.
4. LINQ to Objects
5. LINQ to DataSet та SQL
6. LINQ to Entities

Завдання для самостійної роботи.

Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторного заняття. Підготовка до контрольної роботи. Використати ANTLR для свого варіанту лабораторної роботи №1, 1*.

ТЕОРЕТИЧНА ЧАСТИНА ЛАБОРАТОРНОГО ЗАНЯТТЯ №6

СИНТАКСИЧНИЙ АНАЛІЗ І ОБЧИСЛЕННЯ ВИРАЗІВ

Як транслятори, розроблені для компіляції програм, написаних на мовах програмування високого рівня, перетворюють текст програми в команди, виконувані комп'ютером? Як компілятор відшукує помилки в тексті програми? Спробуємо знайти відповідь на ці запитання, відповівши на більш просте питання: як написати програму, що буде одержувати на вході рядок, що містить числовий вираз, наприклад, 7 - 2 * 3, а на виході видавати відповідний результат? Процедура, яка передує виконанні цієї дії, називається ***синтаксичним аналізом виразів*** (**expression parsing**) і є основою всіх компіляторів та інтерпретаторів мов, електронних таблиць і всіх інших програм, у яких потрібно перетворювати числові вирази у форму, зрозумілу комп'ютеру.

Синтаксичний аналізатор – це програма чи частина програми, що виконує *синтаксичний аналіз*, на першому кроці якого виконується *лексичний аналіз*, далі будується *дерево розбору*. Синтаксичний аналізатор входить до структури транслятора.

Розглянемо принципи побудови трансляторів на простому прикладі аналізу та обчислення значення арифметичного виразу. Обмежимося числовими виразами. Крім того, будемо вважати, що вони складатимуться з таких елементів: числа, оператори **+**, **-**, **/**, *****, **^**, **%**, **=**, дужки та змінні.

Перераховані елементи можна комбінувати у виразах відповідно до правил алгебри, наприклад: **(11 + a) / 2*6; 7 + 8; a + b – c; a = 9 – b**.

Нехай оператори мають наступний пріоритет у порядку спадання:

вищий унарний + та –

^

- **/ %**
- **+ -**
- **нижчий =**

Оператори рівного пріоритету виконуються зліва направо.

Для спрощення будемо вважати, що імена змінних не чутливі до регістру та є однобуквеними (тобто допускається 26 змінних, від **A** до **Z**). Наприклад, **b** й **B** позначають ту саму змінну. Кожне числове

значення має тип **double**. Буде здійснюватися лише мінімальний контроль за помилками.

Іншими словами, запропоновану мову арифметичних виразів можна задати наступною граматикою :

<складний-вираз> → ***<присвоювання>*** | ***<вираз>***

<присвоювання> → ***<змінна>*=<вираз>**

<вираз> → ***<доданок>*** ***<операція1>*** ***<доданок>***

<доданок> → ***<множник >*** ***<операція2>*** ***<множник>***

<множник> → ***<степінь>*** ***<операція3 >*** ***<множник>***

<степінь> → ***<операція1>*(<вираз>) | (<вираз>)| <змінна> | <число>**

<операція1> → **+ | -**

<операція2> → *** | / | %**

<операція3> → **^**

<число> → **(*<цифра>*)***

<цифра> → **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

<змінна> → ***a* | *b* | … | *z* | *A* | *B* | … | *Z***

Спробуйте обчислити наступний вираз: **7 - 2 * 3**

Зрозуміло, що значення дорівнює 1. Спочатку вам може прийти в голову наступний алгоритм:

```
a = одержати перший операнд
while(є операнд)
{
op = одержати оператор
b = одержати другий операнд
a = a op b
}
```

Однак при обчисленні виразу **7 - 2 * 3** в результаті виходить **15** замість **1**, оскільки описана процедура не враховує пріоритет операторів. Не можна просто вибирати операнди й оператори ліворуч праворуч, оскільки правила алгебри говорять, що множення застосовується раніше віднімання.

Лексичний аналіз

Для того щоб обчислювати вирази, необхідно вміти розбивати їх на лексеми. Які об’єкти вважати лексемами залежить від мови програмування. З кожною лексемою пов’язується структура, що складається з пари вигляду (тип лексеми, вказівник на комірку, де зберігається інформація про конкретну лексему). Число типів лексем

залежить від кожної конкретної мови, що аналізується (в даному випадку мова арифметичних виразів), але для кожної конкретної мови вважається скінченним.

Таким чином, *лексичний аналізатор* – це програма, входом якої є низка символів, що представляє вхідний текст програми, а виходом – послідовність лексем. Цей вихід утворює вхід *синтаксичного аналізатора*.

Конструкції, що розпізнаються лексичним аналізатором описуються *автоматною граматикою* та *регулярними виразами*. На першому етапі лексичного аналізу будується детермінований скінченний автомат за регулярними виразами, потім будується таблиця переходів для автомата, а далі вхідний потік символів інтерпретується аналізатором у вихідний потік лексем.

В основі лексичного аналізатора лежить діаграма переходів відповідного скінченного автомата.

Таким чином, для розроблюваного аналізатора використаємо лише три типи лексем: *змінна, число* та *операція* або *роздільник*.

У загальному випадку для лексичного аналізу необхідний метод, що повертає один за іншим всі елементи виразу (тексту). Метод, що розбиває вираз на лексеми, повинен виконувати наступні дії:

(1) ігнорувати роздільники (пробіли, табуляції та переходи на новий рядок);

(2) виділяти лексеми з тексту;

(3) визначати тип лексеми.

Кожен елемент виразу називається *лексемою* (**token**). Тому метод, що повертає чергову лексему, часто називається **GetToken()**.

```
class Parser
        {
  const int Alphabet = 26;            //лексеми
            private enum Types { NONE, DELIMITER,
VARIABLE, NUMBER };
   private string exp; // рядок виразу
   private int expIdx; /* поточний індекс у виразі*/
   private string token; // поточна лексема
   private Types tokType; // тип лексеми
            // Масив для змінних
   private double[] vars = new double[Alphabet];
            public Parser()
            {
 /* Ініціалізуємо змінні   нульовими значеннями.*/
```

```
                for (int i = 0; i <
        vars.Length; i++)
                    vars[i] = 0.0;
            }
          //Метод повертає значення true,
            // якщо c - роздільник
            bool IsDelim(char c)
            {
                if (("+-/*%^=()".IndexOf(c) != -1))
                    return true;
                return false;
            }
      // отримаємо наступну лексему
            void GetToken()
            {
                tokType = Types.NONE;
                token = "";
                if (expIdx == exp.Length) return; /*кінець
виразу*/
                // пропускаємо пробіл
                while (expIdx < exp.Length &&
Char.IsWhiteSpace(exp[expIdx])) ++expIdx;
                // Хвостовий пробіл
                if (expIdx == exp.Length) return;
                if (IsDelim(exp[expIdx]))
                {
                    token += exp[expIdx];
                    expIdx++;
                    tokType = Types.DELIMITER;
                }
                else if (Char.IsLetter(exp[expIdx]))
                {
                    // Це змінна?
                    while (!IsDelim(exp[expIdx]))
                    {
                        token += exp[expIdx];
                        expIdx++;
                        if (expIdx >= exp.Length) break;
                    }
                    tokType = Types.VARIABLE;
                }
                else if (Char.IsDigit(exp[expIdx]))
                {
                    // Це число?
```

```
                while (!IsDelim(exp[expIdx]))
                {
                    token += exp[expIdx];
                    expIdx++;
                    if (expIdx >= exp.Length) break;
                }
                tokType = Types.NUMBER;
            }
        }
    }
```

Давайте розглянемо наведені вище методи більш докладно. Після декількох ініціалізацій метод **getToken()** перевіряє, чи не досягнуто кінця рядка. Якщо у виразі ще є нерозібрана частина, метод **getToken()** спочатку пропускає пробіли, якщо вони є. Після цього **prog** вказує на число, змінну, оператор або — якщо вираз завершувався пробілами — на кінця рядка. Якщо черговий символ є оператором, він повертається у вигляді рядка, збереженого в атрибуті **token**, а **tokType**, що містить тип отриманої лексеми, присвоюється значення **DELIMITER**. Якщо ж наступний символ є буквою, він вважається іменем змінної та повертається в рядку **token**. При цьому **tokType** одержує значення **VARIABLE**. У випадку, коли черговий символ є цифрою, зчитується все число, причому воно міститься в змінній **token**, а його типом буде **NUMBER**. Нарешті, якщо наступний символ не є жодним з перерахованих вище, вважається, що досягнуто кінець виразу. У цьому випадку **token** містить порожній рядок, повернення якого означає кінець виразу.

Як уже було сказано раніше, щоб не ускладнювати код цього методу, були опущені деякі засоби контролю за помилками й зроблені деякі допущення. Наприклад, будь-який нерозпізнаний символ завершує вираз. Крім того, у даній версії програми імена змінних можуть мати будь-яку довжину, але значущою є тільки перша буква. Відповідно до вимог конкретного завдання ви можете ускладнити засоби контролю за помилками та додати інші подробиці.

Щоб краще зрозуміти принцип дії методу **getToken()**, нижче наведені лексеми, що повертаються ним, і типи лексем для наступного вхідного виразу:

A * B - (W + 10)

Лексема	Тип лексеми
A	**VARIABLE**
*****	**DELIMITER**

B	**VARIABLE**
-	**DELIMITER**
(	**DELIMITER**
W	**VARIABLE**
+	**DELIMITER**
10	**NUMBER**
)	**DELIMITER**

Після того як в результаті роботи лексичного аналізу лексеми розпізнано, інформація про них збирається та запам'ятовується в одній з кількох таблиць (*таблиці ідентифікаторів*). Зміст цієї інформації залежить від мови (в даному випадку мови арифметичних виразів).

Таблиця ідентифікаторів (*змінних*) повинна забезпечити можливість швидкого додавання нових ідентифікаторів та нових відомостей про них та можливість швидкого пошуку інформації, що відноситься до даного ідентифікатора.

В нашому випадку, для спрощення, ми вважаємо що застосовуються лише однобуквені змінні латинського алфавіту без урахування регістру. Тому, тут запропоновано ввести таку *таблицю ідентифікаторів*, де кожна змінна зберігається в одному осередку масиву з 26 елементів типу **double**. У більш складному випадку, зрозуміло, що для реалізації таблиці потрібно застосовувати інші методи (списки структур, об'єктів).

Тому у вихідний текст аналізатора додамо наступний фрагмент:

```
double[] vars = new double[26];
```

Ці змінні початково ініціалізуються **0.0** в конструкторі класу.

Крім цього, знадобиться метод для одержання значення заданої змінної. Оскільки імена змінних є буквами від **A** до **Z**, їх можна використати для індексації масиву **vars**, віднімаючи код ASCII букви **A** з імені змінної. Нижче описано метод **findVar()**, що повертає значення змінної:

```
// Повертаємо значення змінної
        double FindVar(string vname)
        {
          if (!Char.IsLetter(vname[0]))
          {
            SyntaxErr(Errors.SYNTAX);
            return 0.0;
          }
          return vars[Char.ToUpper(vname[0]) - 'A'];
        }
```

Цей метод написано так, що він приймає імена будь-якої довжини, але тільки перший символ є значимим. Дане обмеження можна змінити відповідно до ваших потреб.

```
// Одержання значення числа, або змінної
        void Atom(out double result)
        {
            switch (tokType)
            {
                case Types.NUMBER:
                    try
                    {
                        result = Double.Parse(token);
                    }
                    catch (FormatException)
                    {
                        result = 0.0;
                        SyntaxErr(Errors.SYNTAX);
                    }
                    GetToken();
                    return;
                case Types.VARIABLE:
                    result = FindVar(token);
                    GetToken();
                    return;
                default:
                    result = 0.0;
                    SyntaxErr(Errors.SYNTAX);
                    break;
            }
        }
```

Синтаксичний аналіз

Виходом лексичного аналізатора є низка лексем, що утворює вхід синтаксичного аналізатора, що досліджує лише перші компоненти лексем – типи. Інформація про кожну лексему (друга компонента) використовується на більш пізньому етапі процесу компіляції.

Синтаксичний аналіз – це процес, в якому досліджується низка лексем, та встановлюється, чи задовольняє вона синтаксичним правилам, явно сформульованим у визначенні синтаксису мови.

На виході синтаксичний аналізатор видає дерево розбору та таблиці (ідентифікаторів, типів), які є входом для компілятора.

Як правило, фази лексичного та синтаксичного аналізу взаємодіють між собою на одному перегляді. При цьому основною є фаза синтаксичного аналізу, який в свою чергу звертається до лексичного аналізу за черговим термінальним символом.

Більшість відомих методів аналізу належать до одного з двох класів, один із яких об'єднує *низхідні*, а інший *висхідні* алгоритми. Походження цих термінів пов'язане з тим, яким чином утворюються вузли синтаксичного дерева: від аксіом граматики до термінальних символів (*низхідні* аналізатори), чи від термінальних символів до аксіом граматики (*висхідні* аналізатори).

З низхідними аналізаторами пов'язані так звані LL-граматики, які можуть бути проаналізовані достатньо простим для реалізації методом рекурсивного спуску.

Висхідні аналізатори пов'язані LR-граматики. LR-граматики можуть аналізувати більш широкий клас граматик (за допомогою LR-граматик можна визначити більшість мов програмування), тому саме для таких методів існують програми автоматичної побудови аналізаторів.

Для початку розглянемо граматику G_1, яка не містить присвоювання:

<вираз>→ <доданок> <операція1> <доданок>

<доданок> → <множник> <операція2> <множник>

<множник> → <степінь><операція3 > <множник>

<степінь> → <операція1>(<вираз>) | (<вираз>)| <змінна> | <число>

<операція1> → + | -

<операція2> → * | / | %

<операція3> → ^

<число> → (<цифра>)**

<цифра> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<змінна> → a | b | … | z | A | B | … | Z

Побудуємо значення функції FIRST для правих частин правил заданої з аксіомою <*вираз*>:

Тоді:

$FIRST_1$(<*доданок*> <*операція1*> <*доданок*>) ={ +, -, (, *a*, …, *z*, *A*, …, *Z*, 0, …9}

$FIRST_1$(<*множник* > <*операція2*> <*множник*>) = { +, -, (, *a*, …, *z*, *A*, …, *Z*, 0, …9}

$FIRST_1$(<*степінь*><*операція3* > <*множник*>) = { +, -, (, *a*, …, *z*, *A*, …, *Z*, 0, …9}

$FIRST_1$(<*операція1*>(<*вираз*>)) = { +, -}

$FIRST_1$((<*вираз*>)) = { (}

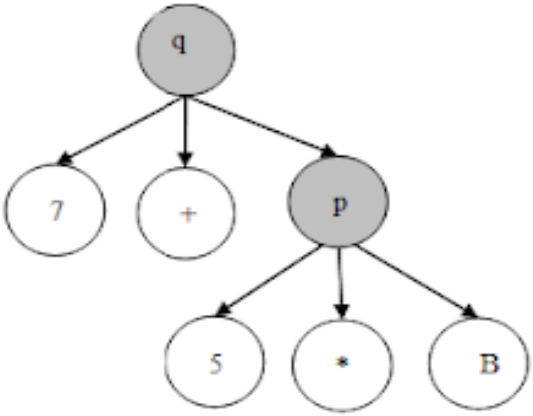
$FIRST_1$(<*змінна*>) = { *a*, …, *z*, *A*, …, *Z*}

$FIRST_1$(<*число*>) = {0, …, 9}.

Значення функції $FIRST_1$ для всіх альтернативних правих частин продукцій попарно не перетинаються, а отже кажуть, що граматика є LL(1)-граматикою. Таким чином, для синтаксичного аналізу цієї граматики може бути застосований метод рекурсивного спуску (без повернень).

Метод рекурсивного спуску без повернень можна використати тільки для LL(1)-граматик, правила яких задовольняють наступній умові: першого символу кожного правила повинно бути достатньо для визначення, застосовного правила.

Для прикладу, нехай вихід лексичного аналізатора – послідовність лексем 7 + 5 * B, тут 7 та 5 числові константи тип лексем **NUMBER**, B – **VARIABLE**, + та * – **DELIMITER**. Запропонований вираз складається з двох доданків: 7 та 5 * B. Другий доданок складається із двох множників: 5 та B. Ці множники являють собою одне число й одну змінну. Послідовність кроків, які необхідно виконати для обчислення цього виразу можна зобразити у вигляді дерева виводу. Побудова дерева виводу це послідовності команд відповідного автомату. Корінь дерева помічається початковим станом автомату, далі корені співпадають з відповідними станами на кожному кроці.



Щоб зрозуміти, як же в дійсності аналізатор обчислює вираз, давайте розглянемо наступний приклад: 7 + 5 * B.

При виклику методу **Evaluate()** (аналіз <*присвоювання*> | <*вираз*>) вхідної точки аналізатора — із вхідного рядка вибирається лексема. Якщо лексема є порожнім рядком, то виникає помилка «Порожній рядок» і робота завершується. Однак у цьому випадку лексемою є число 7. Оскільки це не порожній рядок, викликається метод **EvalExp2()** (аналіз <*доданок*> <*операція1*> <*доданок*>). У результаті, метод **EvalExp2()** викликає **EvalExp3()**(аналіз <*множник* > <*операція2*> <*множник*>), а та у свою чергу викликає **EvalExp4()** (аналіз <*степінь*><*операція3* > <*множник*>), а **EvalExp4()** викликає **EvalExp5()** (<*операція1*>(<*вираз*>)) . Потім метод **EvalExp5()** перевіряє, чи не є лексема унарним плюсом або мінусом. У цьому випадку це не так, тому викликається метод **EvalExp6()**(аналіз (<*вираз*>)| <*змінна*> | <*число*>).. У цей момент **EvalExp6()** може рекурсивно викликати або **EvalExp2()** (у випадку виразу, вкладеного в дужки), або **Atom()** щоб визначити значення числа. Оскільки лексема не є відкриваючою дужкою, виконується метод **Atom()** і **result** присвоюється значення 7. Потім відбувається вибірка наступної лексеми й повернення з ланцюжка викликів методів. Лексемою стає оператор +.

Оскільки поточною лексемою є символ +, він зберігається в змінній **op**. Потім аналізатор вибирає наступну лексему й спуск по ланцюжку починається знову. Як і раніше, викликається метод **Atom()**. Отримане значення 5 повертається змінною **result**, і зчитується лексема

*. Це викликає повернення по ланцюжку до **EvalExp2()**, де зчитується остання лексема B, яка є змінною. Метод FindVar() знаходить в таблиці її значення 0.0. У цей момент відбувається перша арифметична операція — множення 5 на 0. Отриманий результат повертається методу **EvalExp2()**, де відбувається додавання. У результаті додавання у відповіді виходить 7.

Доповнимо мову, що аналізуємо, присвоюванням. Для цього розглянемо граматику *G:*

<складний-вираз>→ <присвоювання> | <вираз>
<присвоювання>→ <змінна>=<вираз>
<вираз>→ <доданок> <операція1> <доданок>
<доданок> → <множник > <операція2> <множник>
<множник> → <степінь><операція3 > <множник>
<степінь> → <операція1>(<вираз>) | (<вираз>)| <змінна> | <число>
<операція1> → + | -
<операція2> → * | / | %
<операція3> → ^
<число> → (<цифра>)*
<цифра> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<змінна> → a | b | … | z | A | B | … | Z

Побудуємо значення функції FIRST для правих частин правил цієї граматики з аксіомою <*складний-вираз*>:

Тоді:

$FIRST_1$(<*присвоювання*>) = { *a*, …, *z*, *A*, …, *Z*}

$FIRST_1$(<*вираз*>) = $FIRST_1$(<*доданок*> <*операція1*> <*доданок*>) ={ +, -, (, *a*, …, *z*, *A*, …, *Z*, 0, …9}

$FIRST_1$(<*множник* > <*операція2*> <*множник*>) = {+, -, (, *a*, …, *z*, *A*, …, *Z*, 0, …9}

$FIRST_1$((<*вираз*>)) = { (}

$FIRST_1$(<*змінна*>) = { *a*, …, *z*, *A*, …, *Z*}

$FIRST_1$(<*число*>) = {0, …, 9}.

Очевидно, граматика не є LL(1)-граматикою. Це випливає з того, що значення $FIRST_1$(<*присвоювання*>)∩$FIRST_1$(<*вираз*>)={*a*, …, *z*, *A*, …, *Z*}.

Існують алгоритми, що перетворюють граматики, які не є LL(1) на LL(1). Проте, як правило такі алгоритми достатньо складні, і

зручніше користуватися для синтаксичного аналізу *методом рекурсивного спуску з поверненням*, який до них може бути застосований. При цьому лексичний аналізатор матиме додатковий метод, який в усіх неоднозначних ситуаціях перед початком розбору запам'ятовує поточний стан лексичного аналізатора та намагається продовжити розбір тексту за першою з можливих альтернатив, якщо альтернатива виявилась невірною, то повертаємося до запам'ятованого стану та переглядаємо наступну альтернативу. Якщо всі варіанти розбору завершилися невдачею, то повертаємо повідомлення про помилку.

З технічної точки зору, це все, що потрібно аналізатору для коректної обробки змінних. Однак поки немає способу присвоїти цим змінним значення. Часто це робиться за межами аналізатора, але в аналізаторі можна розглядати знак рівності як знак операції присвоювання й зробити обробку цього знака частиною аналізатора. Цього можна досягти декількома способами. Один з них — додати в аналізатор метод **EvalExp1()** (аналіз <*присвоювання*>→ <*змінна*>=<*вираз*>), показаний нижче:

```
void EvalExp1(out double result)
        {
            int varIdx;
            Types ttokType;
            string temptoken;
            if (tokType == Types.VARIABLE)
            {
                // зберегти стару лексему
                temptoken = String.Copy(token);
                ttokType = tokType;
                // обчислити індекс змінної
                varIdx = Char.ToUpper(token[0]) - 'A';
                GetToken();
                if (token != "=")
                {
                    PutBack();/* повернути поточну лексему та
відновити стару лексему - це не присвоювання*/
                    token = String.Copy(temptoken);
                    tokType = ttokType;
                }
                else
                {
```

```
// отримати наступну частину виразу
                    GetToken();
            EvalExp2(out result);
                    vars[varIdx] = result;
                    return;
                }
            }
            EvalExp2(out result);
        }
```

Як видно, цьому методу доводиться заглядати вперед, щоб визначити, чи виконується насправді присвоювання. Це пов'язане з тим, що ім'я змінної завжди перебуває перед оператором присвоювання, але саме по собі наявність імені змінної не гарантує, що за нею піде присвоювання. Інакше кажучи, аналізатор сприймає вираз A = 100 як присвоювання, причому він може визначити, що A / 10 ним не є. Для цього метод **EvalExp1()** зчитує з вхідного потоку наступну лексему. Якщо ця лексема не є знаком рівності, вона за допомогою методу PutBack**()** повертається для наступного використання:

```
//повертаємо лексему
void PutBack()
        {
            for (int i = 0; i < token.Length; i++) expIdx--;
        }
```

Evaluate() <*складний-вираз*>→ <*присвоювання*> | <*вираз*>
EvalExp1() <*присвоювання*>→ <*змінна*>=<*вираз*>
EvalExp2() <*вираз*>→ <*доданок*> <*операція1*> <*доданок*>
EvalExp3() <*доданок*> → <*множник* > <*операція2*> <*множник*>
EvalExp4() <*множник*> → <*степінь*><*операція3* > <*множник*>
EvalExp5() <*степінь*> → <*операція1*>(<*вираз*>) | (<*вираз*>)| <*змінна*> | <*число*>
<*операція1*> → + | -
<*операція2*> → * | / | %
<*операція3*> → ^
<*число*> → (<*цифра*>)*
<*цифра*> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<*змінна*> → *a* | *b* | … | *z* | *A* | *B* | … | *Z*

```
namespace MyExcel1
{
    //клас, що виключає помилки для анализатора
    class ParserException : ApplicationException
```

```
        {
            public ParserException(string str) : base(str) {}
            public override string ToString()
            { return Message; }
        }
        class Parser
        {
            enum Types { NONE, DELIMITER, VARIABLE, NUMBER
}; //лексеми.
            enum Errors { SYNTAX, UNBALPARENS, NOEXP,
DIVBYZERO }; // помилки.
            string exp; // рядок виразу
            int expIdx; // поточний індекс у виразі
            string token; // поточна лексема
            Types tokType; // тип лексеми
            // Масив для змінних
            double[] vars = new double[26];
            public Parser()
            {
                // Ініціалізуємо змінні нульовими значеннями.
                for (int i = 0; i < vars.Length; i++)
                    vars[i] = 0.0;
            }

      /* Точка входу аналізатора. */
            public double Evaluate(string expstr)
            {
                double result;
                exp = expstr;
                expIdx = 0;
                try
                {
                    GetToken();
                    if (token == "")
                    {
                        SyntaxErr(Errors.NOEXP); // Вираз відсутній
                        return 0.0;
                    }
                    EvalExp1(out result);
                    if (token != "") /* остання лексема повинна бути
нульова */
                        SyntaxErr(Errors.SYNTAX);
                    return result;
                }
```

```
            catch (ParserException exc)
            {

                Console.WriteLine(exc);
                return 0.0;
            }
        }

        // Обробка присвоювання
        void EvalExp1(out double result)
        {
            int varIdx;
            Types ttokType;
            string temptoken;
            if (tokType == Types.VARIABLE)
            {
                // зберегти стару лексему
                temptoken = String.Copy(token);
                ttokType = tokType;
                // обчислити індекс змінної
                varIdx = Char.ToUpper(token[0]) - 'A';
                GetToken();
                if (token != "=")
                {
                  PutBack();// повернути поточну лексему  в поток
    //відновити стару лексему - це не присвоювання
                    token = String.Copy(temptoken);
                    tokType = ttokType;
                }
                else
                {
                    GetToken();//  отримати  наступну  частину
виразу
                    EvalExp2(out result);
                    vars[varIdx] = result;
                    return;
                }
            }

            EvalExp2(out result);
        }

    // Додавання або віднімання двох доданків
        void EvalExp2(out double result)
```

```
    {
        string op;
        double partialResult;

        EvalExp3(out result);
        while ((op = token) == "+" || op == "-")
        {
            GetToken();
            EvalExp3(out partialResult);
            switch (op)
            {
                case "-":
                    result = result - partialResult;
                    break;
                case "+":
                    result = result + partialResult;
                    break;
            }
        }
    }
// Виконуємо множення чи ділення доух множників
    void EvalExp3(out double result)
    {
        string op;
        double partialResult = 0.0;
        EvalExp4(out result);
        while ((op = token) == "*" || op == "/" || op == "%")
        {
            GetToken();
            EvalExp4(out partialResult);
            switch (op)
            {
                case "*":
                    result = result * partialResult;
                    break;
                case "/":
                    if (partialResult == 0.0)
                        SyntaxErr(Errors.DIVBYZERO);
                    result = result / partialResult;
                    break;
                case "%":
                    if (partialResult == 0.0)
                        SyntaxErr(Errors.DIVBYZERO);
                    result = (int)result % (int)partialResult;
```

```
                    break;
                }
            }
        }
    // Піднесення до степені
        void EvalExp4(out double result)
        {
            double partialResult, ex;
            int t;
            EvalExp5(out result);
            if (token == "^")
            {
                GetToken();
                EvalExp4(out partialResult);
                ex = result;
                if (partialResult == 0.0)
                {
                    result = 1.0;
                    return;
                }
                for (t = (int)partialResult - 1; t > 0; t--)
                    result = result * (double)ex;
            }
        }

        // Множення унарних операторів + й -.
        void EvalExp5(out double result)
        {
            string op;

            op = "";
            if ((tokType == Types.DELIMITER) && token == "+" ||
token == "-")
            {
                op = token;
                GetToken();
            }
            EvalExp6(out result);
            if (op == "-") result = -result;
        }
    // Обчислення виразів в дужках
        void EvalExp6(out double result)
        {
            if ((token == "("))
```

```
            {
                GetToken();
                EvalExp2(out result);
                if (token != ")")
                    SyntaxErr(Errors.UNBALPARENS);
                GetToken();
            }
            else Atom(out result);
        }
        // Одержання значення числа, або змінної
        void Atom(out double result)
        {
            switch (tokType)
            {
                case Types.NUMBER:
                    try
                    {
                        result = Double.Parse(token);
                    }
                    catch (FormatException)
                    {
                        result = 0.0;
                        SyntaxErr(Errors.SYNTAX);
                    }
                    GetToken();
                    return;
                case Types.VARIABLE:
                    result = FindVar(token);
                    GetToken();
                    return;
                default:
                    result = 0.0;
                    SyntaxErr(Errors.SYNTAX);
                    break;
            }
        }
// Повертаємо значення змінної
        double FindVar(string vname)
        {
            if (!Char.IsLetter(vname[0]))
            {
                SyntaxErr(Errors.SYNTAX);
                return 0.0;
            }
```

```
            return vars[Char.ToUpper(vname[0]) - 'A'];
        }
        //повертаємо лексему у вхідний потік.
        void PutBack()
        {
            for (int i = 0; i < token.Length; i++) expIdx--;
        }
        // Обробляємо синтаксичну помилку
        void SyntaxErr(Errors error)
        {
            string[] err ={
                    "Синтаксическая оошибка",
                    "Дисбаланс скобок",
                    "Выражение отсутствет",
                    "Деление на нуль"};
            throw new ParserException(err[(int)error]);
        }
    // отримуємо наступну лексему
        void GetToken()
        {
            tokType = Types.NONE;
            token = "";
            if (expIdx == exp.Length) return; //кінець виразу
            // пропускаємо пробіл
            while (expIdx < exp.Length &&
Char.IsWhiteSpace(exp[expIdx])) ++expIdx;
            // Хвостовий пробіл
            if (expIdx == exp.Length) return;
            if (IsDelim(exp[expIdx]))
            {
                token += exp[expIdx];
                expIdx++;
                tokType = Types.DELIMITER;
            }
            else if (Char.IsLetter(exp[expIdx]))
            {
                // Це змінна?
                while (!IsDelim(exp[expIdx]))
                {
                    token += exp[expIdx];
                    expIdx++;
                    if (expIdx >= exp.Length) break;
                }
                tokType = Types.VARIABLE;
```

```
            }
            else if (Char.IsDigit(exp[expIdx]))
            {
                // Це число?
                while (!IsDelim(exp[expIdx]))
                {
                    token += exp[expIdx];
                    expIdx++;
                    if (expIdx >= exp.Length) break;
                }
                tokType = Types.NUMBER;
            }
        }
        // Метод повертає значення true,
        // якщо c - роздільник
        bool IsDelim(char c)
        {
            if (("+-/*%^=()".IndexOf(c) != -1))
                return true;
            return false;
        }
    }
}
```

КАЛЬКУЛЯТОР (ДЛЯ ЛАБОРАТОРНОЇ РОБОТИ 1) З ВИКОРИСТАННЯМ ANTLR4[1]

http://www.antlr.org/
https://github.com/sharwell/antlr4cs
https://dzone.com/articles/getting-started-with-antlr-in-c

Step 2: Install ANTLR Language Support for Visual Studio (optional) (https://github.com/sharwell/antlr4cs)
Для Visual Studio 2019 пропустити (4.11.2019)
This step is optional, but highly recommended for users working with a version of Visual Studio that the extension supports. If you have one of the Express Editions of Visual Studio, or would like to skip this step, move on to the following step.

1. Open Visual Studio
2. Select **Tools → Extensions and Updates...**
3. In the left pane, select **Online**
4. In the top right, type **ANTLR** to search for extensions
5. If your version of Visual Studio is supported, the results pane will show the extension **ANTLR Language Support** by **Sam Harwell**. You can click the result and then select **Download** to install the extension.
6. Restart Visual Studio after installing the extension

Step 3: Update the NuGet Package Manager

---

[1] **ANTLR** (англ. *Another Tool For Language Recognition*) — генератор синтаксичних аналізаторів, дозволяє автоматично створювати програму-парсер (як і лексичний аналізатор) однією з декількох цільових мов програмування (Java, C++, C#, Python, Ruby) за описом LL(*)-граматики мовою. Дозволяє конструювати компілятори, інтерпретатори, транслятори з різних формальних мов. Також, надає зручні засоби для відновлення після помилок, і повідомлення про них. ANTLR — продовження PCCTS (Purdue Compiler Construction Tool Set), який було розроблено 1989 року.

Основоположником проєкту є професор Теренс Парр з Університету Сан-Франциско. ANTLR — проєкт з відкритим кодом, версія 3.0 поширюється за ліцензією BSD.

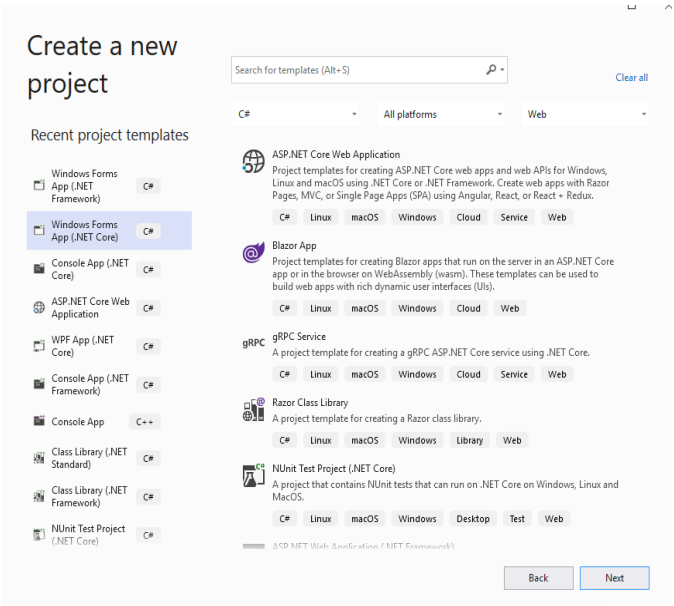
For nearly all Visual Studio 2010 and newer installations

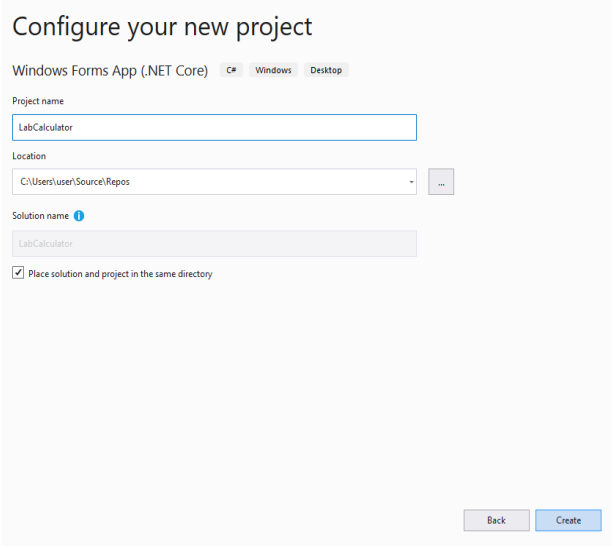
1. Open Visual Studio
2. Select **Tools → Extensions and Updates (Extensions → Manage Extensions** for VS2019**)...**
3. In the left pane, select **Updates**, then select **Product Updates**
4. Wait for ~10 seconds while the application checks for updates (it might not notify you that it's checking in the background)
5. If an update for **NuGet Package Manager** is listed in the results, click to update it
6. Repeat steps 3-5 for any other sections listed under **Updates** in the left pane
7. If you updated NuGet, restart Visual Studio before continuing to Step 4 below

For Visual Studio 2008, and Visual C# 2010 Express Edition

These versions of Visual Studio do not support the NuGet Package Manager extension, but the C# target for ANTLR 4 does support .NET 2.0 and higher so you should be able to use the command line NuGet utility instead of working directly within Visual Studio.

Step 4: Install ANTLR 4 support in a C# project

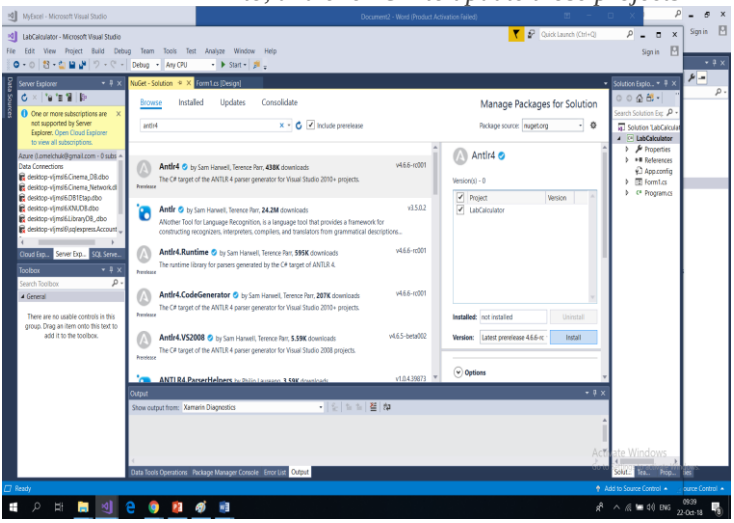


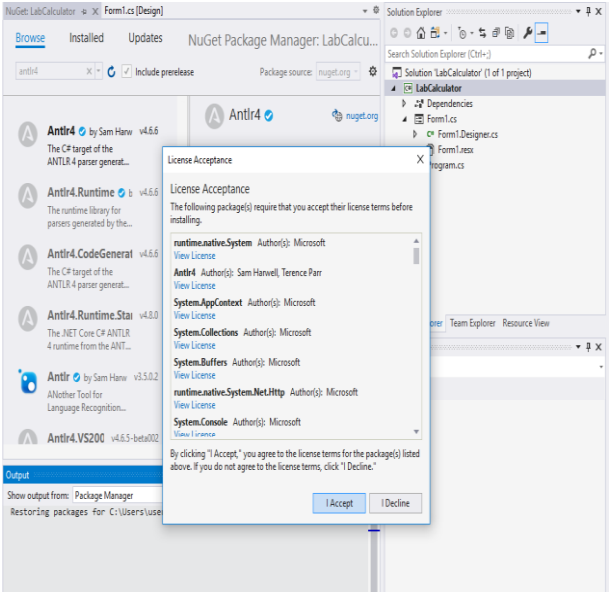


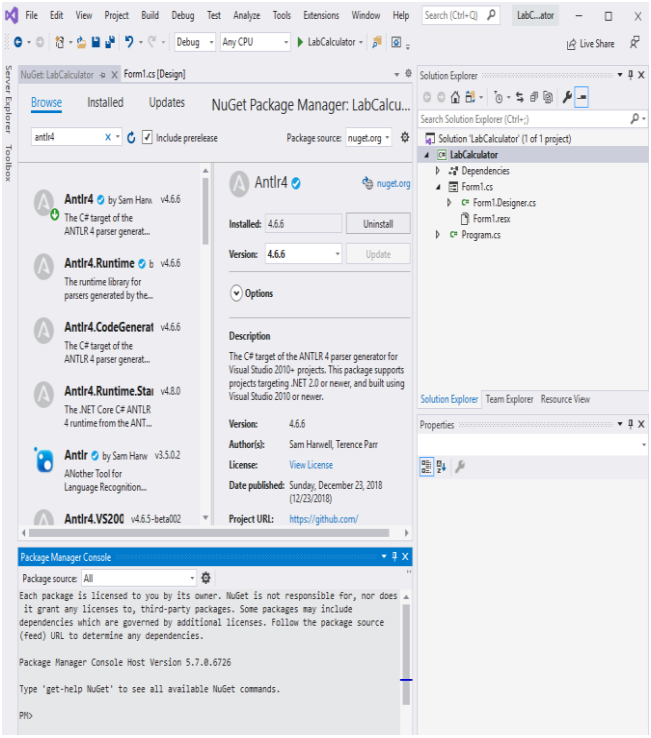
For nearly all Visual Studio 2010 and newer installations

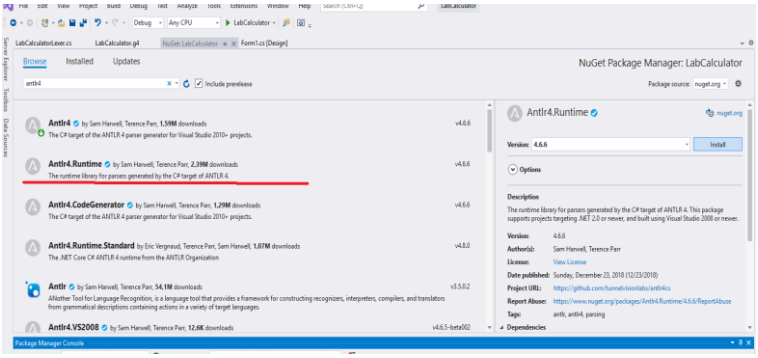
1. Create or open a C# project which will use ANTLR
2. Right click the top-level solution node in the Solution Explorer window and select **Manage NuGet Packages for Solution...**
3. In the left pane, select **Online (Browse)**, then select **nuget.org**
4. At the top of the middle pane, if a drop down says **Stable Only**, change the selection to **Include Prerelease**
5. In the top right, type **Antlr4** to search for the package
6. In the search results, locate and select the package called **ANTLR 4**. In the right pane, verify that the **Id** is listed as exactly **Antlr4**.
7. Click install on the search result

*Select the C# projects you want to add support for ANTLR 4 to, and click **OK** to update those projects*




NuGet: LabCalculator
Form1.cs [Design]
Browse
Installed
Updates
NuGet Package Manager: LabCalcu...
antlr4
Include prerelease
Package source:
nuget.org
Antlr4 by Sam Harw v4.6.6
The C# target of the ANTLR 4 parser generat...
Antlr4.Runtime
Antlr4.CodeGenerat
Antlr4.Runtime.Stai
Antlr by Sam Harw v3.5.0.2
ANother Tool for Language Recognition...
Antlr4.VS200 v4.6.5-beta002
Antlr4
nuget.org
License Acceptance
License Acceptance
The following package(s) require that you accept their license terms before installing.
runtime.native.System Author(s): Microsoft
View License
Antlr4 Author(s): Sam Harwell, Terence Parr
View License
System.AppContext Author(s): Microsoft
View License
System.Collections Author(s): Microsoft
View License
System.Buffers Author(s): Microsoft
View License
runtime.native.System.Net.Http Author(s): Microsoft
View License
System.Console Author(s): Microsoft
By clicking "I Accept," you agree to the license terms for the package(s) listed above. If you do not agree to the license terms, click "I Decline."
I Accept
I Decline
Output
Show output from: Package Manager
Restoring packages for C:\Users\use
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'LabCalculator' (1 of 1 project)
LabCalculator
Dependencies
Form1.cs
Form1.Designer.cs
Form1.resx
rogram.cs
Team Explorer
Resource View

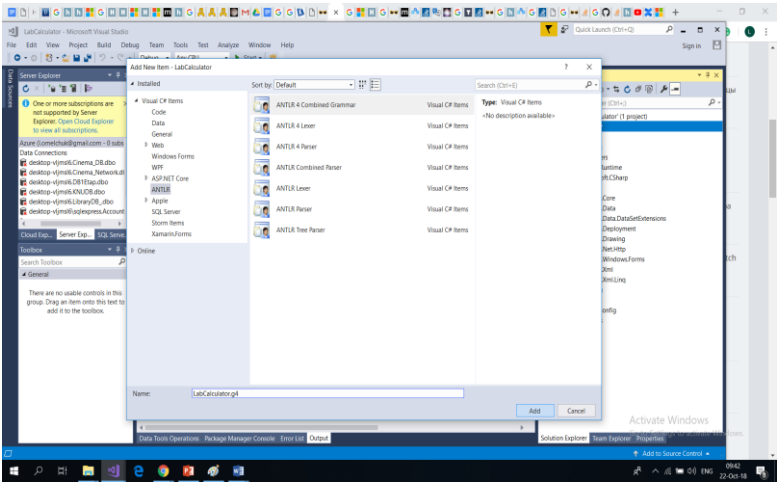


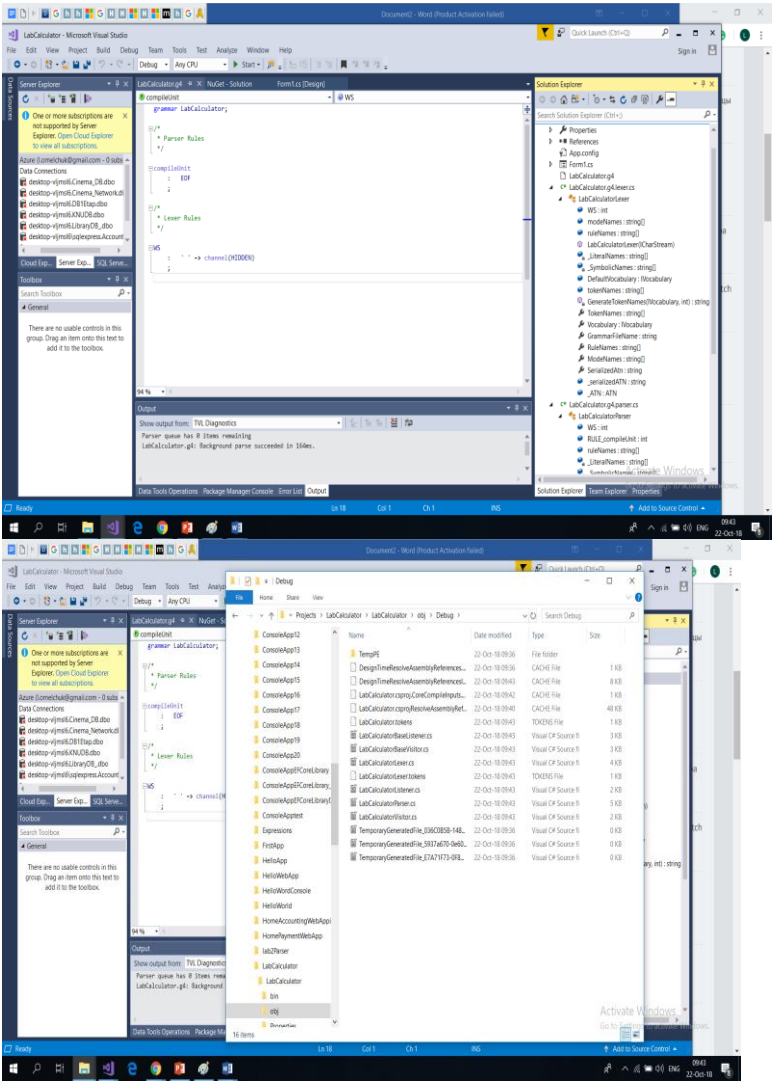


При встановленому ANTLR Language Support:

Using templates provided by the ANTLR Language Support extension

1. Right click the project (or subfolder) in **Solution Explorer** where the new grammar should be placed
2. Select **Add → New Item...**
3. In the left pane, expand **Visual C# Items** and select **ANTLR**
4. Select one of the ANTLR 4 templates and give it a name, and click **Add** to create the new grammar file and add it to the project





Без встановленого ANTLR Language Support (VS 2019 на 04.11.2019):

Without using the ANTLR Language Support extension

1. Right click the project (or subfolder) in **Solution Explorer** where the new grammar should be placed

2. Select **Add → New Item...**

3. In the left pane, expand **Visual C# Items** and select **General**

4. In the middle pane, select **Text File**

5. In the name box, type the complete name of the grammar file (including the .g4 extension), e.g. CustomLanguage.g4, and click **Add** to create the new file and add it to the project

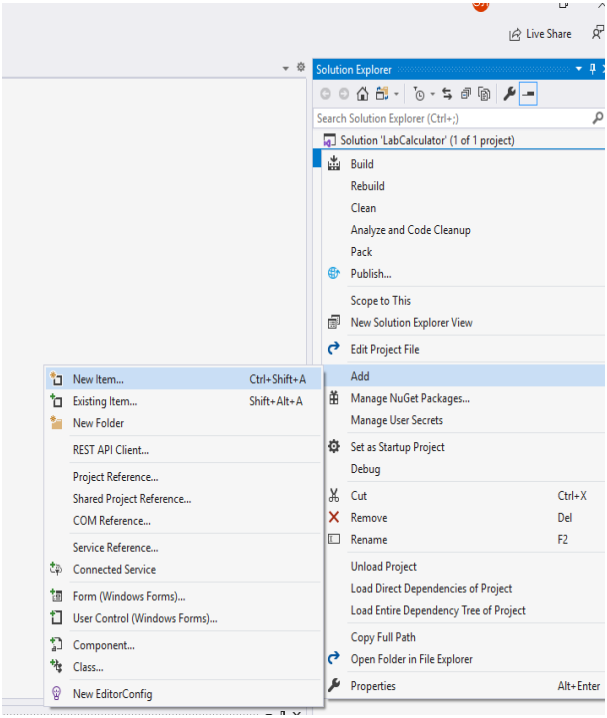
6. Select **File → Advanced Save Options...**

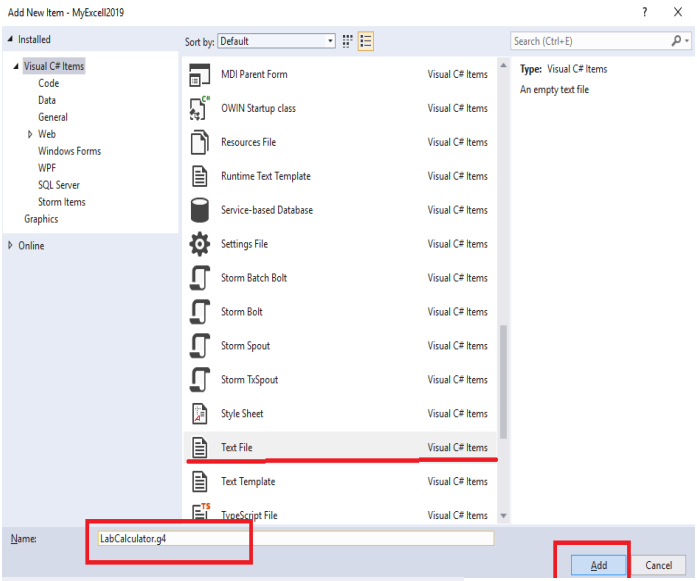
7. For **Encoding**, select **Unicode (UTF-8 without signature) - Codepage 65001**

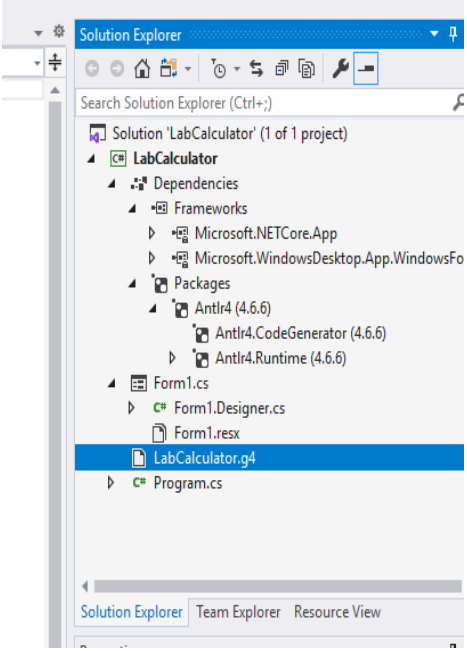
8. Click **OK**, and then save the file

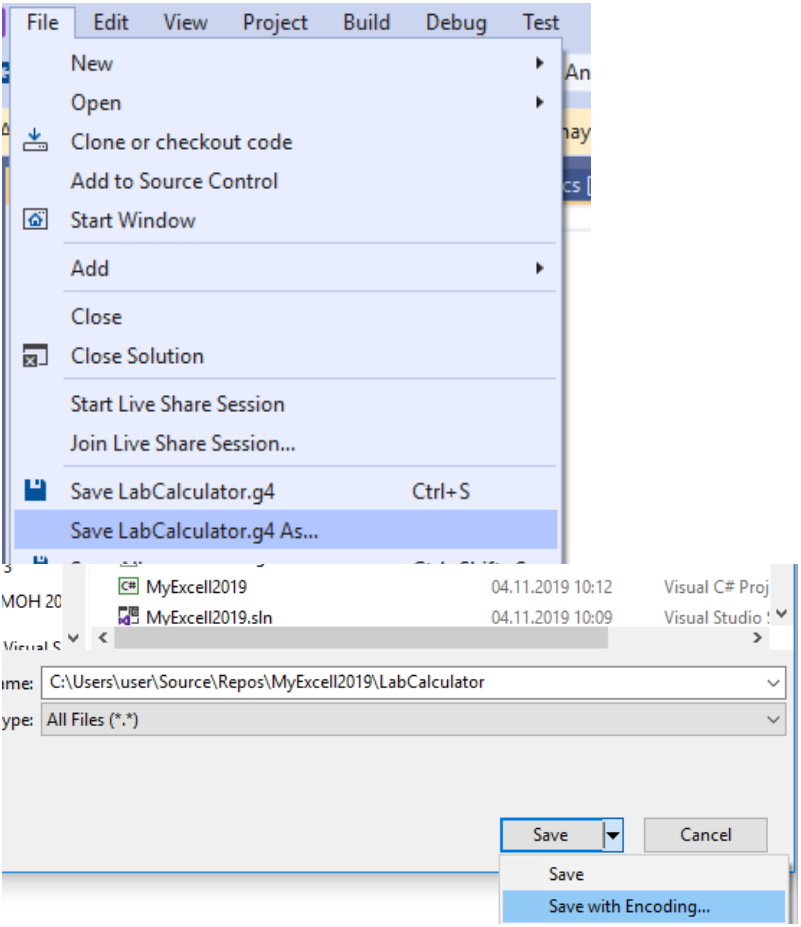
9. Add the grammar declaration at the top of the file, e.g. the following for a grammar named CustomLanguage.g4
```
grammar CustomLanguage;
```

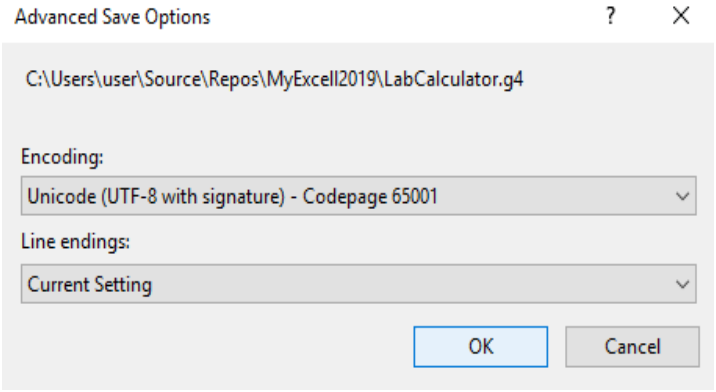
10. Follow step 2 of the *Add an existing grammar to the project* section below to configure the build properties of the newly added grammar file


Live Share
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'LabCalculator' (1 of 1 project)
Build
Rebuild
Clean
Analyze and Code Cleanup
Pack
Publish...
Scope to This
New Solution Explorer View
Edit Project File
Add
Manage NuGet Packages...
Manage User Secrets
Set as Startup Project
Debug
Cut Ctrl+X
Remove Del
Rename F2
Unload Project
Load Direct Dependencies of Project
Load Entire Dependency Tree of Project
Copy Full Path
Open Folder in File Explorer
Properties Alt+Enter
New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Folder
REST API Client...
Project Reference...
Shared Project Reference...
COM Reference...
Service Reference...
Connected Service
Form (Windows Forms)...
User Control (Windows Forms)...
Component...
Class...
New EditorConfig

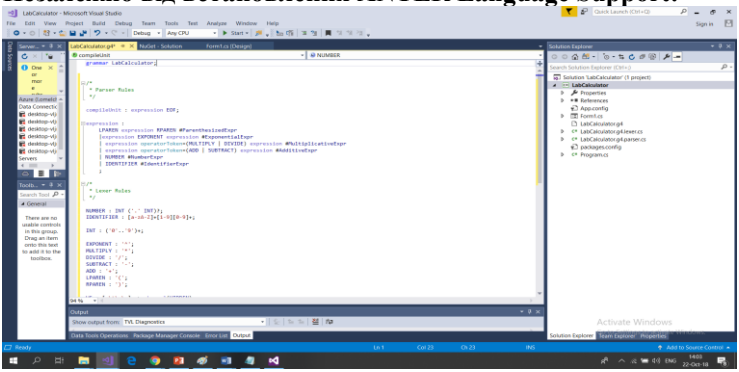

Add New Item - MyExcell2019
Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
SQL Server
Storm Items
Graphics
Online
Sort by: Default
Search (Ctrl+E)
MDI Parent Form
OWIN Startup class
Resources File
Runtime Text Template
Service-based Database
Settings File
Storm Batch Bolt
Storm Bolt
Storm Spout
Storm TxSpout
Style Sheet
Text File
Text Template
TypeScript File
Visual C# Items
Type: Visual C# Items
An empty text file
Name:
LabCalculator.g4
Add
Cancel


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'LabCalculator' (1 of 1 project)
LabCalculator
Dependencies
Frameworks
Microsoft.NETCore.App
Microsoft.WindowsDesktop.App.WindowsFo
Packages
Antlr4 (4.6.6)
Antlr4.CodeGenerator (4.6.6)
Antlr4.Runtime (4.6.6)
Form1.cs
Form1.Designer.cs
Form1.resx
LabCalculator.g4
Program.cs
Solution Explorer
Team Explorer
Resource View


File
Edit
View
Project
Build
Debug
Test
New
Open
Clone or checkout code
Add to Source Control
Start Window
Add
Close
Close Solution
Start Live Share Session
Join Live Share Session...
Save LabCalculator.g4 Ctrl+S
Save LabCalculator.g4 As...
MyExcel2019 04.11.2019 10:12 Visual C# Proj
MyExcel2019.sln 04.11.2019 10:09 Visual Studio
C:\Users\user\Source\Repos\MyExcel2019\LabCalculator
All Files (*.*)
Save
Cancel
Save
Save with Encoding...



Незалежно від встановлення ANTLR Language Support:



grammar LabCalculator;[2]

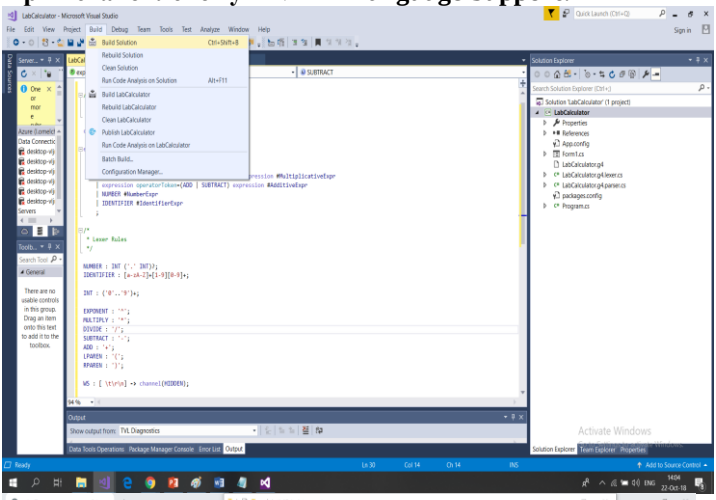
2

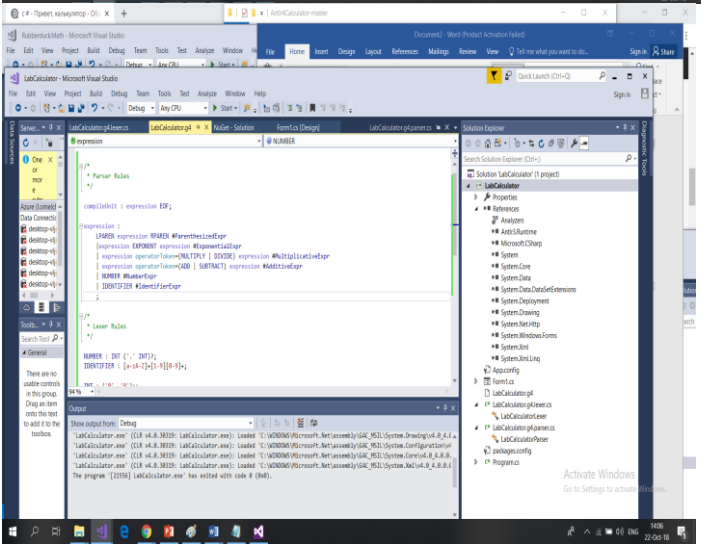
(...) підправило
(...)* повторення підправила 0 чи більше разів
(...)+ Повторення підправила 1 чи більше разів
(...)? підправило, може бути відсутнє
{...} семантичні дії
[...] параметри правила
| оператор альтернативи
.. оператор діапазону
~ заперечення
. довільний символ

```
/*
 * Parser Rules
 */

compileUnit : expression EOF;

expression :
           LPAREN expression RPAREN #ParenthesizedExpr
           |expression EXPONENT expression #ExponentialExpr
    | expression operatorToken=(MULTIPLY | DIVIDE) expression
#MultiplicativeExpr
           | expression operatorToken=(ADD | SUBTRACT) expression
#AdditiveExpr
           | NUMBER #NumberExpr
           | IDENTIFIER #IdentifierExpr
           ;

/*
 * Lexer Rules
 */

NUMBER : INT ('.' INT)?;
IDENTIFIER : [a-zA-Z]+[1-9][0-9]+;

INT : ('0'..'9')+;

EXPONENT : '^';
MULTIPLY : '*';
DIVIDE : '/';
SUBTRACT : '-';
ADD : '+';
LPAREN : '(';
RPAREN : ')';

WS : [ \t\r\n] -> channel(HIDDEN);
```

= присвоювання
: мітка початку
; мітка закінчення правила

При встановленому ANTLR Language Support:





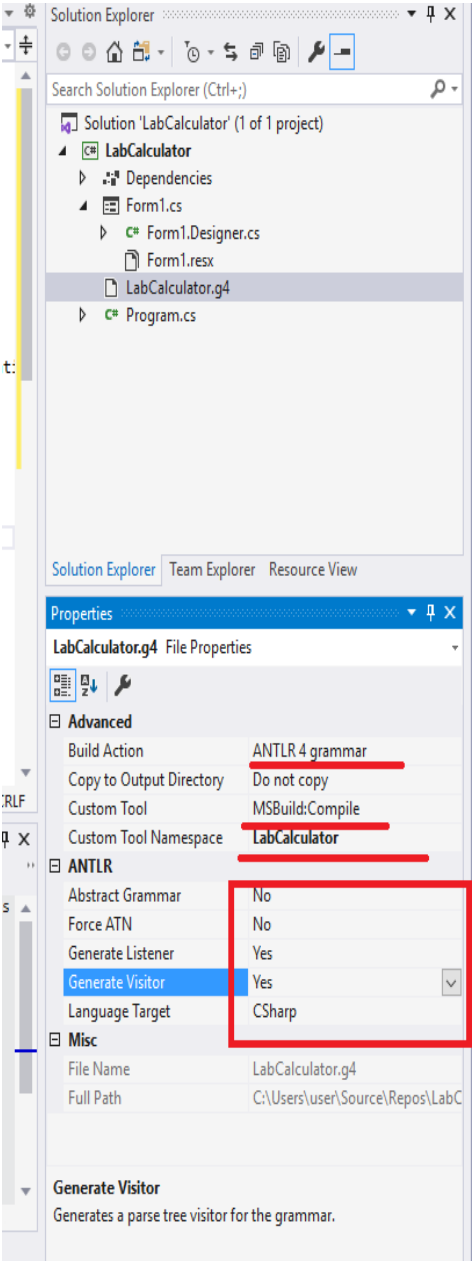
Без встановленого ANTLR Language Support (VS 2019 на 04.11.2019):

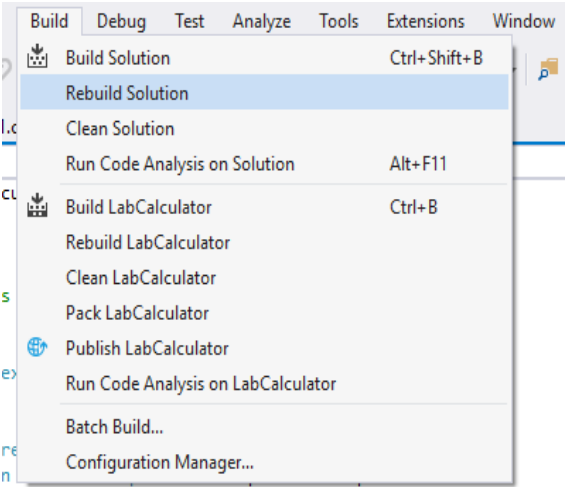
If you have the ANTLR Language Support extension installed, this step is automatically performed. Otherwise, you will need to right click the

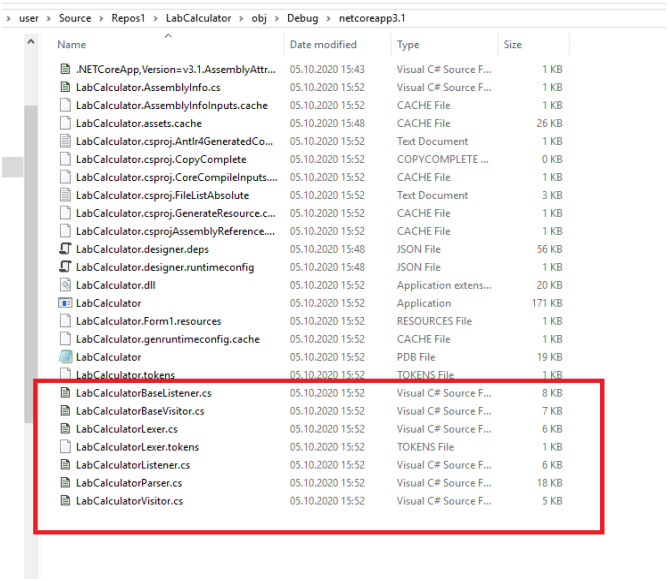
grammar file in **Solution Explorer** and select **Properties**. In the properties window, configure the following items.

- **Build Action**: Antlr4
- **Custom Tool**: MSBuild:Compile
- **Custom Tool Namespace**: The complete name of the namespace you want the generated classes to be located within. The ANTLR Language Support extension configures this field according to the root namespace configured for the C# project combined with the subfolder within the project where the grammar is located.

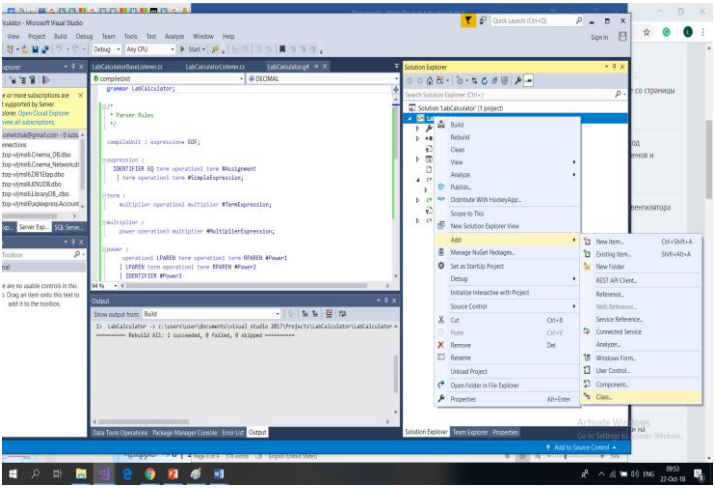
Закрийте та відкрийте заново Visual Studio з Вашим проєктом

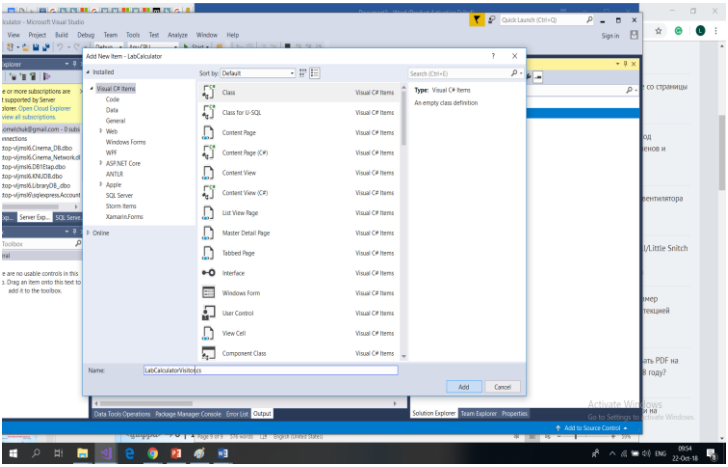

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'LabCalculator' (1 of 1 project)
LabCalculator
Dependencies
Form1.cs
Form1.Designer.cs
Form1.resx
LabCalculator.g4
Program.cs
Solution Explorer
Team Explorer
Resource View
Properties
LabCalculator.g4 File Properties
Advanced
Build Action
ANTLR 4 grammar
Copy to Output Directory
Do not copy
Custom Tool
MSBuild:Compile
Custom Tool Namespace
LabCalculator
ANTLR
Abstract Grammar
No
Force ATN
No
Generate Listener
Yes
Generate Visitor
Yes
Language Target
CSharp
Misc
File Name
LabCalculator.g4
Full Path
C:\Users\user\Source\Repos\LabC
Generate Visitor
Generates a parse tree visitor for the grammar.

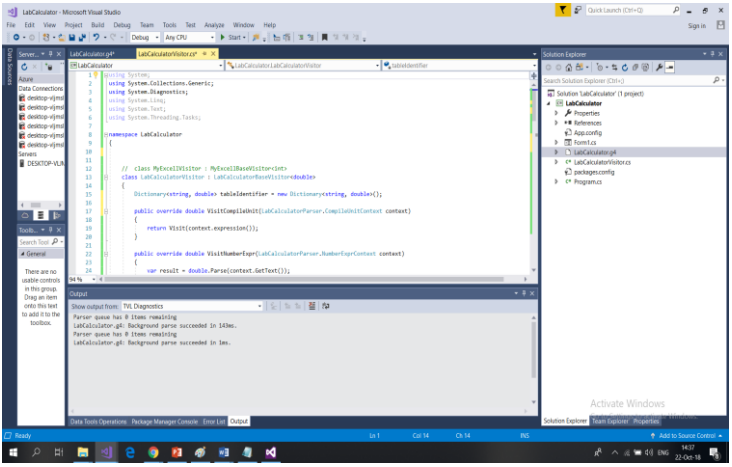




Незалежно від встановлення ANTLR Language Support:





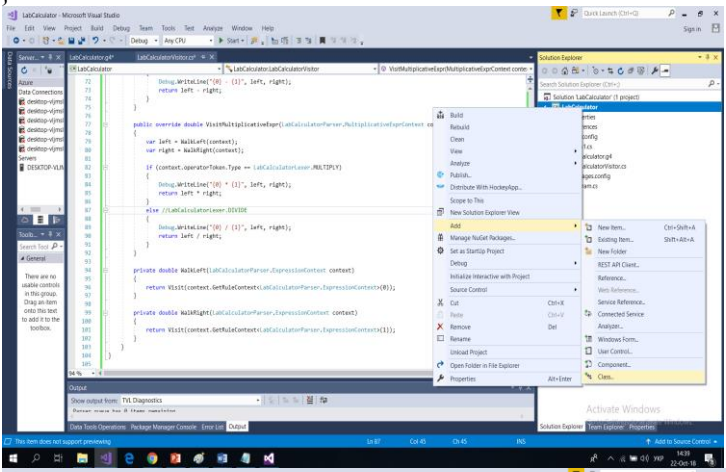


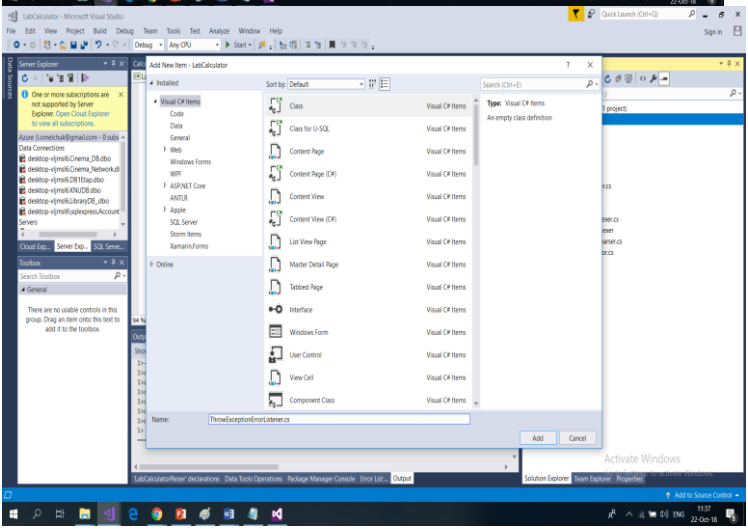
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LabCalculator
{
    class LabCalculatorVisitor : LabCalculatorBaseVisitor<double>
    {
        //таблиця ідентифікаторів (тут для прикладу)
        //в лабораторній роботі замінити на свою!!!!
        Dictionary<string, double> tableIdentifier = new
Dictionary<string, double>();

        public override double
VisitCompileUnit(LabCalculatorParser.CompileUnitContext context)
        {
            return Visit(context.expression());
        }

        public override double
VisitNumberExpr(LabCalculatorParser.NumberExprContext context)
        {
            var result = double.Parse(context.GetText());
            Debug.WriteLine(result);
```

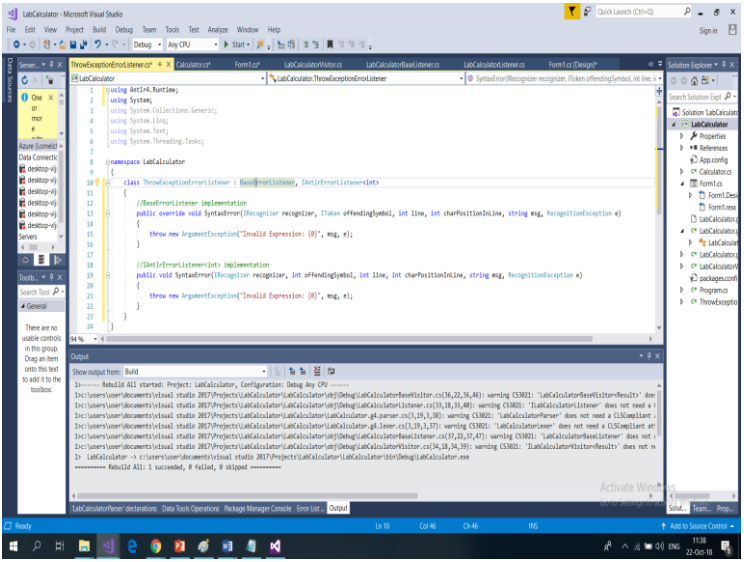
```
            return result;
        }

        //IdentifierExpr
        public override double
VisitIdentifierExpr(LabCalculatorParser.IdentifierExprContext
context)
        {
            var result = context.GetText();
            double value;
            //видобути значення змінної з таблиці
            if (tableIdentifier.TryGetValue(result.ToString(), out value))
            {
                return value;
            }
            else
            {
                return 0.0;
            }
        }

        public override double
VisitParenthesizedExpr(LabCalculatorParser.ParenthesizedExprCont
ext context)
        {
            return Visit(context.expression());
        }

        public override double
VisitExponentialExpr(LabCalculatorParser.ExponentialExprContext
context)
        {
            var left = WalkLeft(context);
            var right = WalkRight(context);

            Debug.WriteLine("{0} ^ {1}", left, right);
            return System.Math.Pow(left, right);
        }

        public override double
VisitAdditiveExpr(LabCalculatorParser.AdditiveExprContext context)
        {
            var left = WalkLeft(context);
```

```
            var right = WalkRight(context);

            if (context.operatorToken.Type == LabCalculatorLexer.ADD)
            {
                Debug.WriteLine("{0} + {1}", left, right);
                return left + right;
            }
            else //LabCalculatorLexer.SUBTRACT
            {
                Debug.WriteLine("{0} - {1}", left, right);
                return left - right;
            }
        }

        public override double
VisitMultiplicativeExpr(LabCalculatorParser.MultiplicativeExprConte
xt context)
        {
            var left = WalkLeft(context);
            var right = WalkRight(context);

            if (context.operatorToken.Type ==
LabCalculatorLexer.MULTIPLY)
            {
                Debug.WriteLine("{0} * {1}", left, right);
                return left * right;
            }
            else //LabCalculatorLexer.DIVIDE
            {
                Debug.WriteLine("{0} / {1}", left, right);
                return left / right;
            }
        }

        private double
WalkLeft(LabCalculatorParser.ExpressionContext context)
        {
            return
Visit(context.GetRuleContext<LabCalculatorParser.ExpressionConte
xt>(0));
        }

        private double
WalkRight(LabCalculatorParser.ExpressionContext context)
```

```
        {
            return
Visit(context.GetRuleContext<LabCalculatorParser.ExpressionConte
xt>(1));
        }
    }
}
```

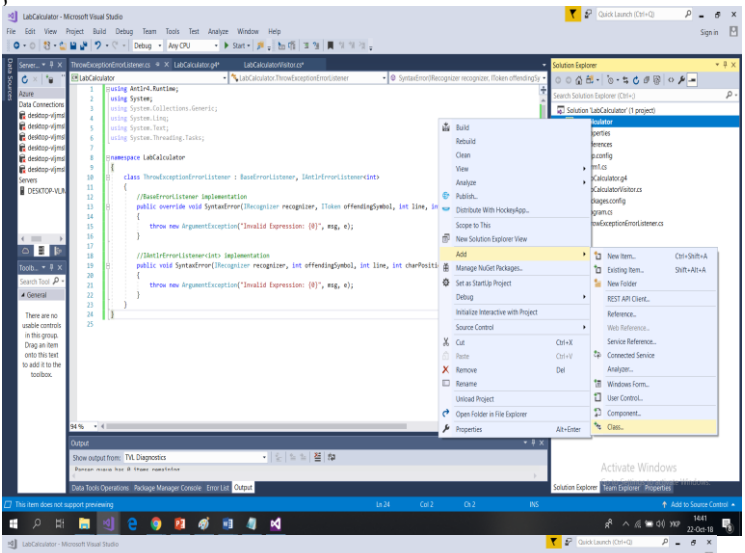


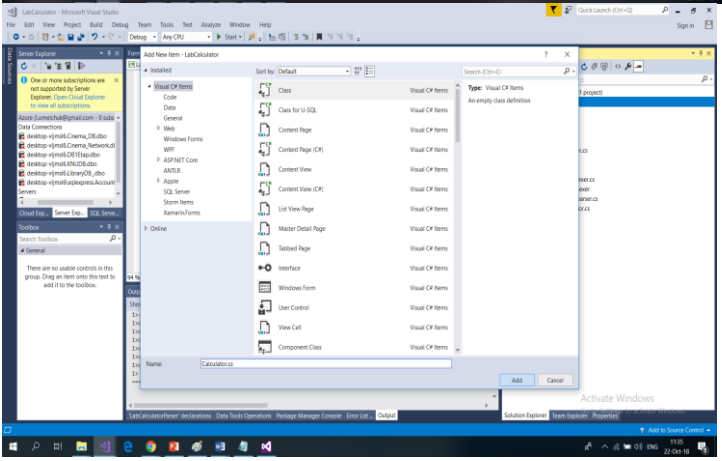


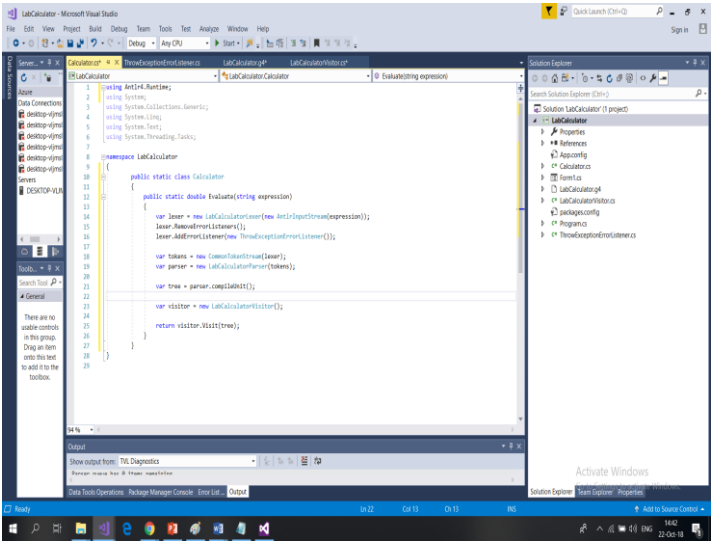


```
using Antlr4.Runtime;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LabCalculator
{
    class ThrowExceptionErrorListener : BaseErrorListener,
IAntlrErrorListener<int>
    {
        //BaseErrorListener implementation
        public override void SyntaxError(IRecognizer recognizer, IToken
offendingSymbol, int line, int charPositionInLine, string msg,
RecognitionException e)
        {
            throw new ArgumentException("Invalid Expression: {0}",
msg, e);
        }

        //IAntlrErrorListener<int> implementation
```

```
        public void SyntaxError(IRecognizer recognizer, int
offendingSymbol, int line, int charPositionInLine, string msg,
RecognitionException e)
        {
            throw new ArgumentException("Invalid Expression: {0}",
msg, e);
        }
    }
}
```

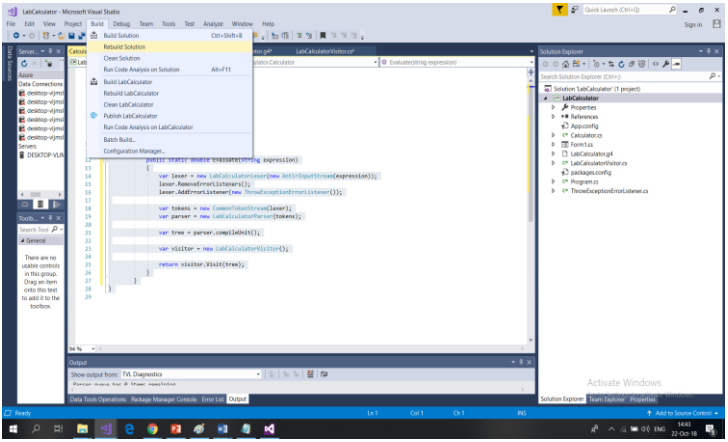


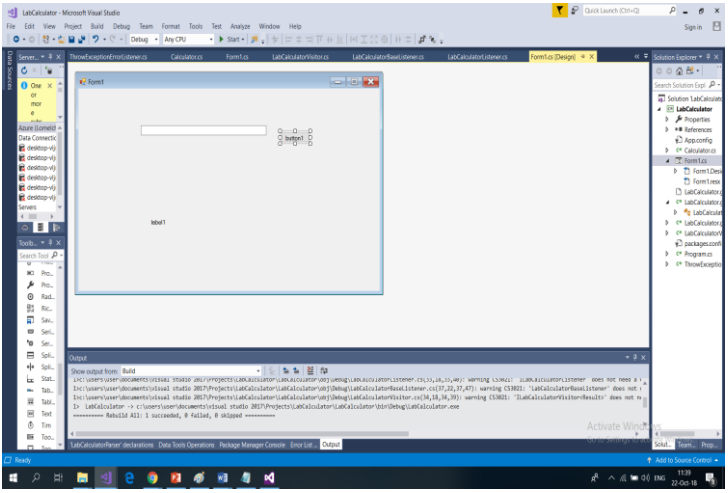


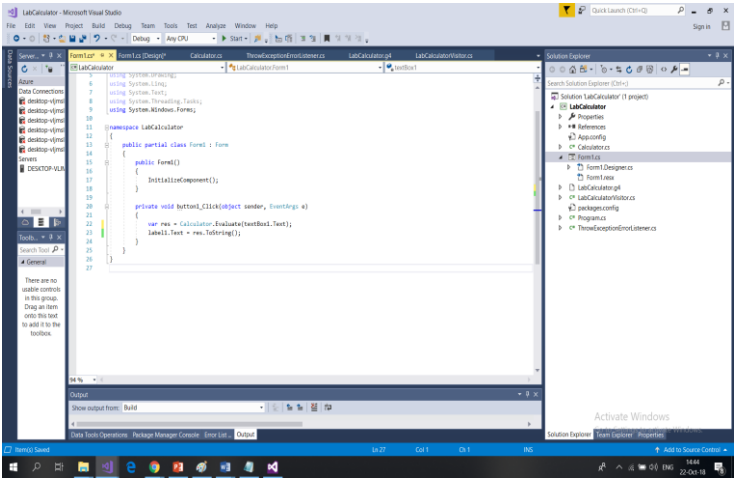


```
using Antlr4.Runtime;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LabCalculator
{
    public static class Calculator
    {
        public static double Evaluate(string expression)
        {
            var lexer = new LabCalculatorLexer(new
AntlrInputStream(expression));
            lexer.RemoveErrorListeners();
            lexer.AddErrorListener(new
ThrowExceptionErrorListener());

            var tokens = new CommonTokenStream(lexer);
            var parser = new LabCalculatorParser(tokens);

            var tree = parser.compileUnit();
```

```
            var visitor = new LabCalculatorVisitor();

            return visitor.Visit(tree);
        }
    }
}
```






LabCalculator - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Form1.cs*
Form1.cs [Design]*
Calculator.cs
ThrowExceptionErrorListener.cs
LabCalculator.g4
LabCalculatorVisitor.cs
LabCalculator
Form1
textBox1
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace LabCalculator
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
var res = Calculator.Evaluate(textBox1.Text);
label1.Text = res.ToString();
}
}
}
Solution Explorer
Solution 'LabCalculator' (1 project)
LabCalculator
Properties
References
App.config
Calculator.cs
Form1.cs
Form1.Designer.cs
Form1.resx
LabCalculator.g4
LabCalculatorVisitor.cs
packages.config
Program.cs
ThrowExceptionErrorListener.cs
Output
Show output from: Build
Data Tools Operations
Package Manager Console
Error List
Output
Item(s) Saved
Ln 27
Col 1
Ch 1
INS
Add to Source Control
Activate Windows
Solution Explorer
Team Explorer
Properties
ENG
16:44
22-Oct-18

ЧАСТИНА 2. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ C#. ШАБЛОНИ ПРОЄКТУВАННЯ. ОГЛЯД ТЕХНОЛОГІЇ .NET CORE

ТИЖДЕНЬ 7

Тема 7. КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС БАГАТОПЛАТФОРМНИХ ЗАСТОСУНКІВ .NET (.NET MAUI). ОГЛЯД

Огляд, мета і призначення теми: Користувацький інтерфейс багатоплатформних застосунків .NET (.NET MAUI). Огляд.

Вивчивши матеріал даної теми, студент зможе: вміти використовувати .NET (.NET MAUI). Розуміти архітектуру .NET (.NET MAUI).

Лабораторний практикум 7

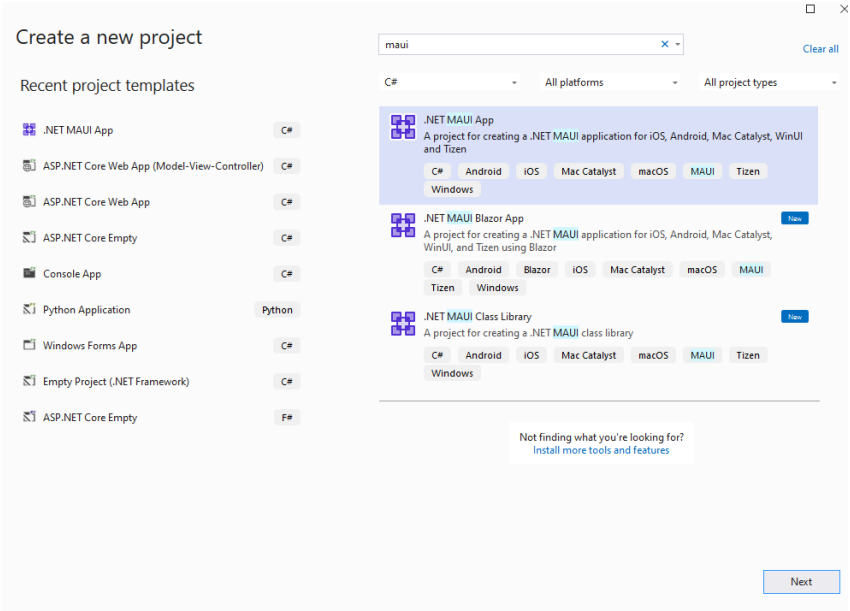
Навчитися розробляти простий користувацький інерфейс багатоплатформних застосунків .Net MAUI, Узгодження UML-діаграм до лабораторної роботи №1* (60 хв). Захист домашнього завдання № 6 (30 хв).

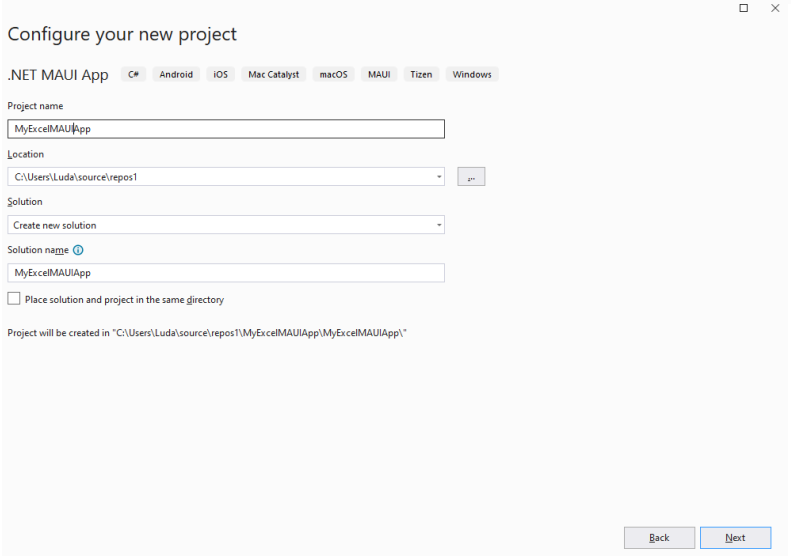
ТЕОРЕТИЧНА ЧАСТИНА ЛАБОРАТОРНОГО ЗАНЯТТЯ

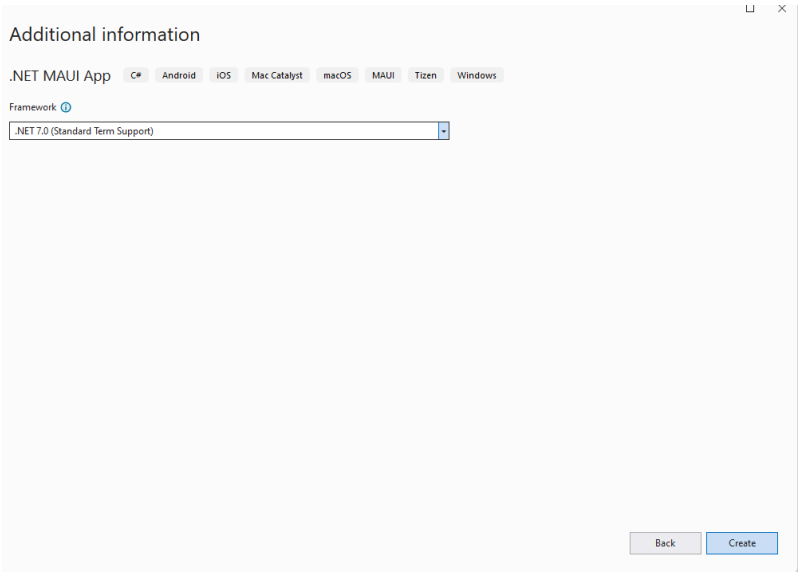
КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС БАГАТОПЛАТФОРМНИХ ЗАСТОСУНКІВ .NET (.NET MAUI). ОГЛЯД.

Крок 1: Створення нового проєкту

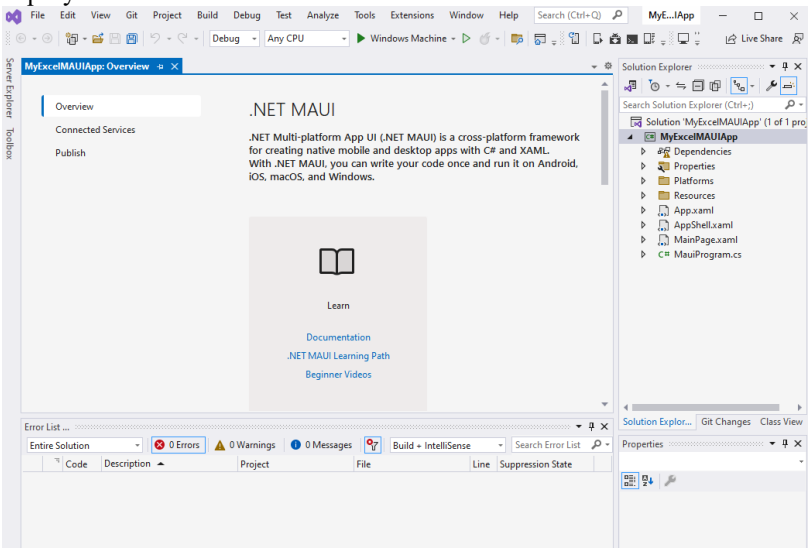
1.1. Відкрийте Visual Studio або інший IDE, який підтримує MAUI розробку.
1.2. Створіть новий проєкт MAUI App (.NET MAUI) з назвою "MyExcelMAUIApp" або іншою, яка вам подобається.


Create a new project
Recent project templates
.NET MAUI App
C#
ASP.NET Core Web App (Model-View-Controller)
C#
ASP.NET Core Web App
C#
ASP.NET Core Empty
C#
Console App
C#
Python Application
Python
Windows Forms App
C#
Empty Project (.NET Framework)
C#
ASP.NET Core Empty
F#
maui
Clear all
C#
All platforms
All project types
.NET MAUI App
A project for creating a .NET MAUI application for iOS, Android, Mac Catalyst, WinUI and Tizen
C# Android iOS Mac Catalyst macOS MAUI Tizen Windows
.NET MAUI Blazor App
New
A project for creating a .NET MAUI application for iOS, Android, Mac Catalyst, WinUI, and Tizen using Blazor
C# Android Blazor iOS Mac Catalyst macOS MAUI Tizen Windows
.NET MAUI Class Library
New
A project for creating a .NET MAUI class library
C# Android iOS Mac Catalyst macOS MAUI Tizen Windows
Not finding what you're looking for?
Install more tools and features
Next


Configure your new project
.NET MAUI App
C# Android iOS Mac Catalyst macOS MAUI Tizen Windows
Project name
MyExcelMAUIApp
Location
C:\Users\Luda\source\repos1
...
Solution
Create new solution
Solution name
MyExcelMAUIApp
Place solution and project in the same directory
Project will be created in "C:\Users\Luda\source\repos1\MyExcelMAUIApp\MyExcelMAUIApp\"
Back
Next



В результаті маємо:



Тут:

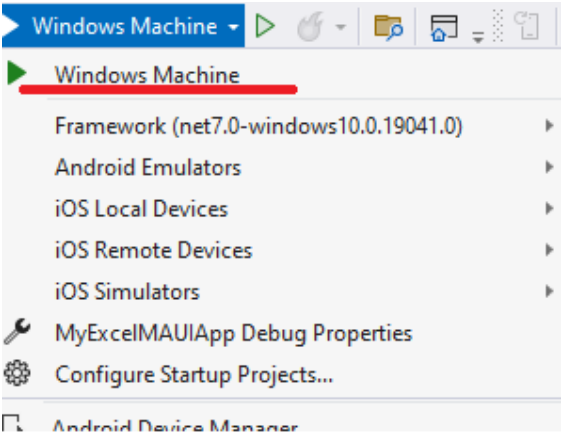
- Папка **Platforms** містить підкаталоги кожної окремої платформи. Кожна папка містить файли коду для взаємодії з певною платформою.
- Папка **Resources** містить файли ресурсів, що використовуються в програмі, зокрема файли шрифтів, іконок, зображень, тощо.
- **App.xaml**: визначає ресурси, загальні для всієї програми. Файл App.xaml є головним конфігураційним файлом застосунку.NET MAUI і містить ресурси та налаштування для застосунку
- **App.xaml.cs**: файл із кодом C#, з якого починається виконання програми.
  Файл App.xaml.cs є класом застосунку, який наслідується від Application і відповідає за головний клас застосунку.NET MAUI. У цьому файлі описано клас **App**, який є головним класом застосунку.
  Файл містить:
  - **partial class App : Application**: Оголошення класу **App**, який наслідується від **Application**. Цей клас визначає головний клас застосунку, де відбувається його ініціалізація та налаштування.
  - **public App()**: Конструктор класу **App**, який викликається при створенні нового об'єкта застосунку. В даному випадку, він містить наступні кроки:
  - **InitializeComponent()**: Цей метод ініціалізує компоненти застосунку. Це автоматично згенерований метод, який підключає XAML-код та код, що визначений в файлі App.xaml.
  - **MainPage = new AppShell()**; Встановлення головної сторінки застосунку. У цьому випадку, встановлюється AppShell, який визначає головний інтерфейс застосунку.
- **AppShell.xaml**: визначає загальний візуальний інтерфейс програми.
  У цьому файлі описано елемент Shell, який є головним контейнером для застосунку:

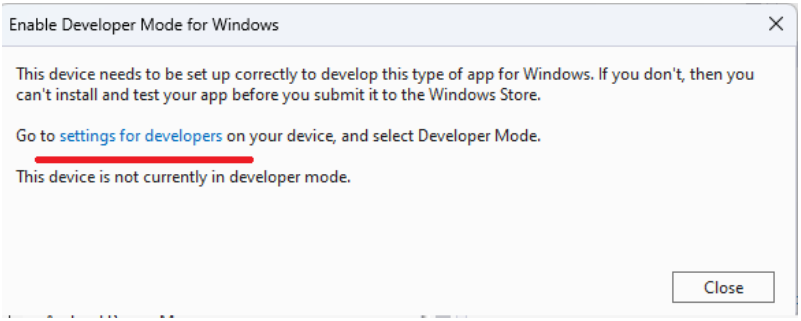
- **x:Class="MyExcelMAUIApp.AppShell"**: Вказує на клас, який визначає логіку коду для цього XAML-файлу. У цьому випадку, це клас AppShell в просторі імен MyExcelMAUIApp.
- **xmlns**: Визначає простір імен, в якому застосунок буде працювати. У даному випадку, це http://schemas.microsoft.com/dotnet/2021/maui, що вказує на використання .NET MAUI.
- **xmlns:x**: Визначає простір імен для елементів XAML. Це http://schemas.microsoft.com/winfx/2009/xaml, який використовується для XAML-синтаксису.
- **xmlns:local**: Визначає простір імен для локальних типів. У даному випадку, це вказує на те, що типи з простору імен MyExcelMAUIApp використовуються в XAML.
- **Shell.FlyoutBehavior="Disabled"**: Визначає поведінку лівого флай-ауту (бокового меню). У даному випадку, флай-аут відключений.
- **ShellContent**: Елемент, який визначає вмістову сторінку для Shell. В даному випадку, встановлюється головна сторінка застосунку.
- **Title**: Заголовок сторінки.
- **ContentTemplate**: Шаблон вмісту сторінки. В цьому випадку, встановлюється шаблон з класу MainPage.
- **Route**: Маршрут для сторінки.

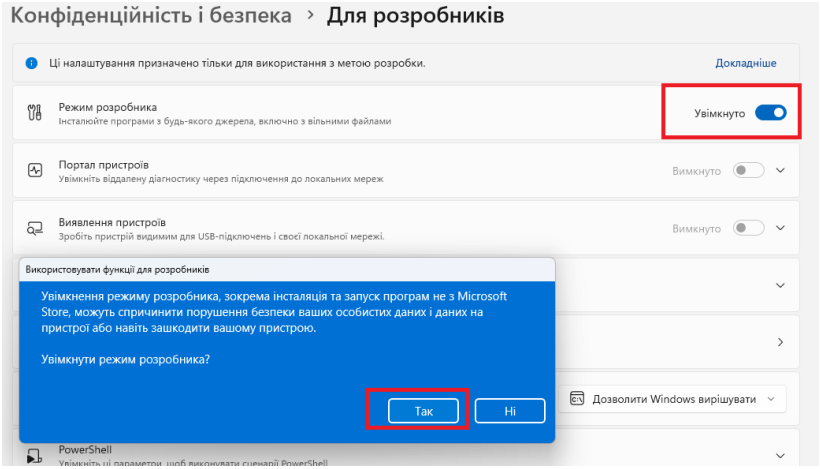
- **AppShell.xaml.cs**: файл із кодом C#, який пов'язаний із файлом AppShell.xaml і визначає пов'язану з ним програмну логіку.
  У цьому файлі описано клас **AppShell**, який наслідується від **Shell**:
  - **public partial class AppShell : Shell**: Оголошення класу **AppShell**, який наслідується від класу **Shell**. Цей клас визначає головний контейнерний елемент **Shell**, де встановлюються основні структура та вигляд застосунку.
  - **public AppShell():** Конструктор класу **AppShell**, який викликається при створенні нового об'єкта **AppShell**. В даному випадку, він містить наступні кроки:

    - **InitializeComponent():** Цей метод ініціалізує компоненти застосунку. Це автоматично згенерований метод, який підключає код, що визначений в файлі AppShell.xaml.
- **MainPage.xaml**: файл із візуальним інтерфейсом для єдиної сторінки MainPage у вигляді xaml.
- **MainPage.xaml.cs**: файл, який містить логіку сторінки MainPage мовою C#.
- **MauiProgram.cs**: містить клас MauiProgram, який визначає стартовий клас програми (за замовчуванням клас App) та ряд загальних для програми налаштувань. Файл MauiProgram.cs є важливою частиною архітектури .NET MAUI і відповідає за створення та налаштування застосунку:
  Цей файл визначає статичний клас MauiProgram, який містить метод CreateMauiApp(). Основна задача цього методу - створити об'єкт MauiApp, який представляє застосунок. Основні етапи роботи методу:
    - Створення будівельника застосунку за допомогою MauiApp.CreateBuilder().
    - Виклик методу UseMauiApp<App>(), де App - клас, який визначає головний клас застосунку. Виходячи з відомого нам файлу App.xaml.cs, це є основний клас застосунку, який має метод OnMauiInit(), виклик якого відбувається при ініціалізації застосунку.
    - Налаштування шрифтів за допомогою методу ConfigureFonts(), який додає різні шрифти до застосунку.
    - Опціонально, якщо програма працює в режимі відладки (DEBUG), то додається можливість логування через builder.Logging.AddDebug().
    - Завершення створення об'єкта застосунку за допомогою методу Build().

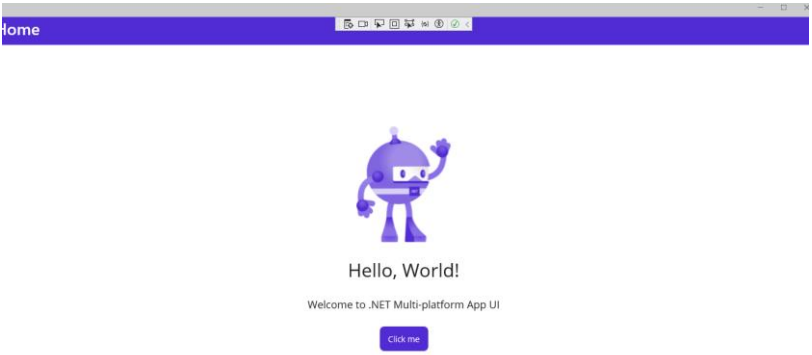
Будемо розробляти застосунок під Windows.


Windows Machine
Windows Machine
Framework (net7.0-windows10.0.19041.0)
Android Emulators
iOS Local Devices
iOS Remote Devices
iOS Simulators
MyExcelMAUIApp Debug Properties
Configure Startup Projects...
Android Device Manager


Enable Developer Mode for Windows
This device needs to be set up correctly to develop this type of app for Windows. If you don't, then you can't install and test your app before you submit it to the Windows Store.
Go to settings for developers on your device, and select Developer Mode.
This device is not currently in developer mode.
Close

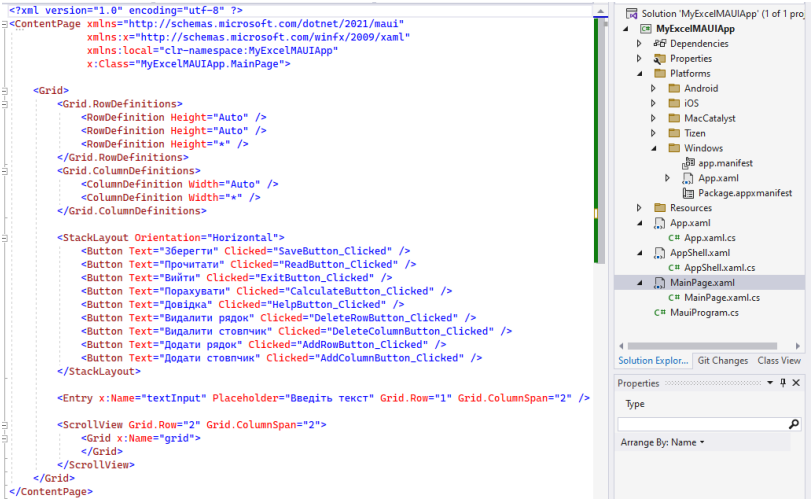


Запустимо застосунок:



Крок 2: Додавання XAML-розмітки

2.1. Відкрийте файл MainPage.xaml.
2.2. Додайте вміст XAML-розмітки, який наведено нижче.



Цей файл містить розмітку (UI) сторінки, на якій користувач взаємодіє з програмою. Ось декілька важливих елементів з цього файла:

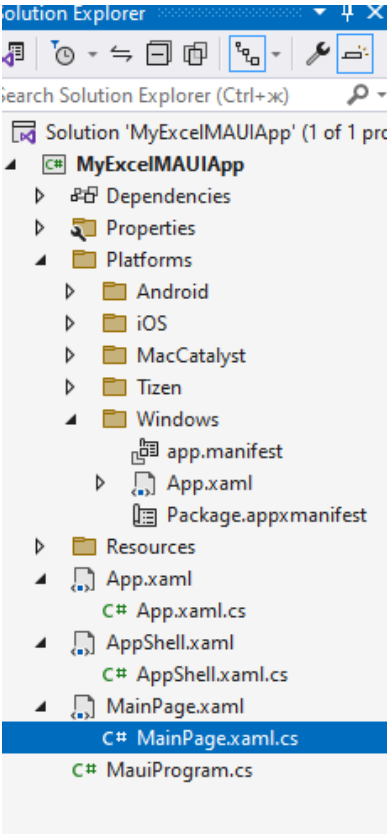
- <ContentPage> - це кореневий елемент, який визначає сторінку. Він містить різні елементи розмітки та обробники подій.
- <Grid> - це контейнер для розміщення інших UI елементів у вигляді рядків та стовпців.
- <StackLayout> - це контейнер для групування кнопок разом у горизонтальному напрямку.
- <Button> - це елемент для створення кнопки з текстом, яку користувач може натиснути.
- <Entry> - це елемент для введення тексту.
- <ScrollView> - це контейнер, який дозволяє прокручувати вміст, який не вміщається на екрані.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:MyExcelMAUIApp"
             x:Class="MyExcelMAUIApp.MainPage">

    <Grid>
```

```
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <StackLayout Orientation="Horizontal">
            <Button Text="Зберегти" Clicked="SaveButton_Clicked" />
            <Button Text="Прочитати" Clicked="ReadButton_Clicked" />
            <Button Text="Вийти" Clicked="ExitButton_Clicked" />
            <Button Text="Порахувати" Clicked="CalculateButton_Clicked" />
            <Button Text="Довідка" Clicked="HelpButton_Clicked" />
            <Button Text="Видалити рядок" Clicked="DeleteRowButton_Clicked"
/>
            <Button Text="Додати рядок" Clicked="AddRowButton_Clicked" />
        </StackLayout>

        <Entry x:Name="textInput" Placeholder="Введіть текст" Grid.Row="1"
Grid.ColumnSpan="2" />

        <ScrollView Grid.Row="2" Grid.ColumnSpan="2">
            <Grid x:Name="grid">
            </Grid>
        </ScrollView>
    </Grid>
</ContentPage>
```

Крок 3: Додавання коду до MainPage.xaml.cs

3.1. Відкрийте файл MainPage.xaml.cs.



3.2. Змініть вміст файлу MainPage.xaml.cs.

Цей файл містить код C#, який визначає поведінку та функціональність програми. Основні елементи цього файла:
public partial class MainPage : ContentPage - це клас, який відображає сторінку і успадковує властивості та методи від **ContentPage**.
CountColumn та **CountRow** - це константи, що визначають кількість стовпців та рядків у вашій таблиці.

```
4 references
public partial class MainPage : ContentPage
{
    const int CountColumn = 20; // кількість стовпчиків (A to Z)
    const int CountRow = 50;    // кількість рядків
```

Конструктор **MainPage()** - цей метод викликається при створенні об'єкта **MainPage**. У ньому викликається метод ініціалізації компонент **InitializeComponent()** та метод **CreateGrid()**, який створює початкову розмітку таблиці.

```
public MainPage()
{
    InitializeComponent();
    CreateGrid();
}
```

CreateGrid() - цей метод створює таблицю (**Grid**) з відповідними рядками, стовпцями, мітками та вхідними полями. Він викликає два допоміжних методи **AddColumnsAndColumnLabels ()** – для додавання стовпців і їх підписів, а також метод **AddRowsAndCellEntries()** – для додавання рядків та комірок. Він також додає обробник події **Unfocused (втрати фокусу)** для вхідних полів.

```
//створення таблиці
1 reference
private void CreateGrid()
{
    AddColumnsAndColumnLabels();
    AddRowsAndCellEntries();
}
```

```
1 reference
private void AddColumnsAndColumnLabels()
{
    // Додати стовпці та підписи для стовпців
    for (int col = 0; col < CountColumn + 1; col++)
    {
        grid.ColumnDefinitions.Add(new ColumnDefinition());

        if (col > 0)
        {
            var label = new Label
            {
                Text = GetColumnName(col),
                VerticalOptions = LayoutOptions.Center,
                HorizontalOptions = LayoutOptions.Center
            };
            Grid.SetRow(label, 0);
            Grid.SetColumn(label, col);
            grid.Children.Add(label);
        }
    }
}
```

```
1 reference
private void AddRowsAndCellEntries()
{
    // Додати рядки, підписи для рядків та комірки
    for (int row = 0; row < CountRow; row++)
    {
        grid.RowDefinitions.Add(new RowDefinition());

        // Додати підпис для номера рядка
        var label = new Label
        {
            Text = (row + 1).ToString(),
            VerticalOptions = LayoutOptions.Center,
            HorizontalOptions = LayoutOptions.Center
        };
        Grid.SetRow(label, row + 1);
        Grid.SetColumn(label, 0);
        grid.Children.Add(label);

        // Додати комірки (Entry) для вмісту
        for (int col = 0; col < CountColumn; col++)
        {
            var entry = new Entry
            {
                Text = "",
                VerticalOptions = LayoutOptions.Center,
                HorizontalOptions = LayoutOptions.Center
            };
            entry.Unfocused += Entry_Unfocused;  //  обробник події Unfocused
            Grid.SetRow(entry, row + 1);
            Grid.SetColumn(entry, col + 1);
            grid.Children.Add(entry);
        }
    }
}
```

GetColumnName(int colIndex) - цей метод перетворює числовий індекс стовпця на буквенну назву, подібну до Excel.

```
2 references
private string GetColumnName(int colIndex)
{
    int dividend = colIndex;
    string columnName = string.Empty;

    while (dividend > 0)
    {
        int modulo = (dividend - 1) % 26;
        columnName = Convert.ToChar(65 + modulo) + columnName;
        dividend = (dividend - modulo) / 26;
    }

    return columnName;
}
```

Entry_Unfocused (object sender, TextChangedEventArgs e) - це обробник події, який викликається, коли користувач вийде зі зміненої клітинки (втратить фокус).

```
// викликається, коли користувач вийде зі зміненої клітинки (втратить фокус)
3 references
private void Entry_Unfocused(object sender, FocusEventArgs e)
{
    var entry = (Entry)sender;
    var row = Grid.GetRow(entry) - 1;
    var col = Grid.GetColumn(entry) - 1;
    var content = entry.Text;

    // Додайте додаткову логіку, яка виконується при виході зі зміненої клітинки
}
```

Інші методи - це обробники подій для кнопок, такі як **CalculateButton_Clicked**, **SaveButton_Clicked**, **ReadButton_Clicked** тощо. Вони викликаються при натисканні відповідних кнопок.

Кожен метод відповідає за конкретну функціональність, а також взаємодіє з елементами розмітки через їх ідентифікатори (**x:Name**). Наприклад, метод **ExitButton_Clicked** взаємодіє з елементом **<Button>** з ідентифікатором "**ExitButton**".

Ці методи доповнюють один одного, створюючи повноцінний функціонал програми для роботи з таблицями, зберіганням та завершенням роботи.
Методи для кнопок **CalculateButton_Clicked**, **SaveButton_Clicked**, **ReadButton_Clicked** викликаються, при натисканні відповідних кнопок, що визначені у файлі MainPage.xaml. Ви повинні самостійно додати логіку обробки даних для цих методів.

```
0 references
private void CalculateButton_Clicked(object sender, EventArgs e)
{
    // Обробка кнопки "Порахувати"
}

0 references
private void SaveButton_Clicked(object sender, EventArgs e)
{
    // Обробка кнопки "Зберегти"
}

0 references
private void ReadButton_Clicked(object sender, EventArgs e)
{
    // Обробка кнопки "Прочитати"
}
```

Метод **ExitButton_Clicked** відображає спливаюче вікно для підтвердження виходу з програми.

```
0 references
private async void ExitButton_Clicked(object sender, EventArgs e)
{
    bool answer = await DisplayAlert("Підтвердження", "Ви дійсно хочете вийти?", "Так", "Ні");
    if (answer)
    {
        System.Environment.Exit(0);
    }
}
```

Метод **HelpButton_Clicked** відображає спливаюче вікно з короткою інформацією про програму. Вам потрібно цю інформацію змінити та доповнити.

```
0 references
private async void HelpButton_Clicked(object sender, EventArgs e)
{
    await DisplayAlert("Довідка", "Лабораторна робота 1. Студента Василя Іваненка", "ОК");
}
```

Метод **DeleteRowButton_Clicked** видаляють останній рядок або стовпець з таблиці (Grid).

```
0 references
private void DeleteRowButton_Clicked(object sender, EventArgs e)
{
    if (grid.RowDefinitions.Count > 1)
    {
        int lastRowIndex = grid.RowDefinitions.Count - 1;
        grid.RowDefinitions.RemoveAt(lastRowIndex);
        grid.Children.RemoveAt(lastRowIndex * (CountColumn + 1)); // Remove label
        for (int col = 0; col < CountColumn; col++)
        {
            grid.Children.RemoveAt((lastRowIndex * CountColumn) + col + 1); // Remove entry
        }
    }
}
0 references
private void DeleteColumnButton_Clicked(object sender, EventArgs e)
{
      if (grid.ColumnDefinitions.Count > 1)
      {
          int lastColumnIndex = grid.ColumnDefinitions.Count - 1;
          grid.ColumnDefinitions.RemoveAt(lastColumnIndex);
          grid.Children.RemoveAt(lastColumnIndex); // Remove label
          for (int row = 0; row < CountRow; row++)
          {
              grid.Children.RemoveAt(row * (CountColumn + 1) + lastColumnIndex + 1); // Remove e
          }
      }
}
```

Метод **AddRowButton_Clicked** додають новий рядок або стовпець до таблиці (Grid).

```
0 references
private void AddRowButton_Clicked(object sender, EventArgs e)
{
    int newRow = grid.RowDefinitions.Count;

    // Add a new row definition
    grid.RowDefinitions.Add(new RowDefinition());

    // Add label for the row number
    var label = new Label
    {
        Text = newRow.ToString(),
        VerticalOptions = LayoutOptions.Center,
        HorizontalOptions = LayoutOptions.Center
    };
    Grid.SetRow(label, newRow);
    Grid.SetColumn(label, 0);
    grid.Children.Add(label);

    // Add entry cells for the new row
    for (int col = 0; col < CountColumn; col++)
    {
        var entry = new Entry
        {
            Text = "",
            VerticalOptions = LayoutOptions.Center,
            HorizontalOptions = LayoutOptions.Center
        };

        entry.Unfocused += Entry_Unfocused;

        Grid.SetRow(entry, newRow);
        Grid.SetColumn(entry, col + 1);
        grid.Children.Add(entry);
    }
}
```

Повний текст файлу:

```
using Microsoft.Maui.Controls;
using Microsoft.Maui.Controls.Compatibility;
using System;
using System.Collections.Generic;
using Grid = Microsoft.Maui.Controls.Grid;
```

```
namespace MyExcelMAUIApp
{
    public partial class MainPage : ContentPage
    {
        const int CountColumn = 20; // кількість стовпчиків (A to Z)
        const int CountRow = 50;    // кількість рядків


        public MainPage()
        {
            InitializeComponent();
            CreateGrid();
        }

        //створення таблиці
        private void CreateGrid()
        {
            AddColumnsAndColumnLabels();
            AddRowsAndCellEntries();
        }

        private void AddColumnsAndColumnLabels()
        {
            // Додати стовпці та підписи для стовпців
            for (int col = 0; col < CountColumn + 1; col++)
            {
                grid.ColumnDefinitions.Add(new ColumnDefinition());

                if (col > 0)
                {
                    var label = new Label
                    {
                        Text = GetColumnName(col),
                        VerticalOptions = LayoutOptions.Center,
                        HorizontalOptions = LayoutOptions.Center
                    };
                    Grid.SetRow(label, 0);
                    Grid.SetColumn(label, col);
                    grid.Children.Add(label);
                }
            }
        }

        private void AddRowsAndCellEntries()
```

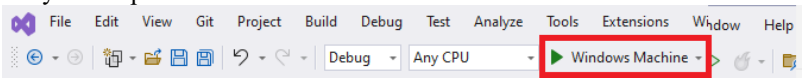
```
        {
            // Додати рядки, підписи для рядків та комірки
            for (int row = 0; row < CountRow; row++)
            {
                grid.RowDefinitions.Add(new RowDefinition());

                // Додати підпис для номера рядка
                var label = new Label
                {
                    Text = (row + 1).ToString(),
                    VerticalOptions = LayoutOptions.Center,
                    HorizontalOptions = LayoutOptions.Center
                };
                Grid.SetRow(label, row + 1);
                Grid.SetColumn(label, 0);
                grid.Children.Add(label);

                // Додати комірки (Entry) для вмісту
                for (int col = 0; col < CountColumn; col++)
                {
                    var entry = new Entry
                    {
                        Text = "",
                        VerticalOptions = LayoutOptions.Center,
                        HorizontalOptions = LayoutOptions.Center
                    };
                    entry.Unfocused += Entry_Unfocused;  // обробник
події Unfocused
                    Grid.SetRow(entry, row + 1);
                    Grid.SetColumn(entry, col + 1);
                    grid.Children.Add(entry);
                }
            }
        }


        private string GetColumnName(int colIndex)
        {
            int dividend = colIndex;
            string columnName = string.Empty;

            while (dividend > 0)
            {
                int modulo = (dividend - 1) % 26;
```

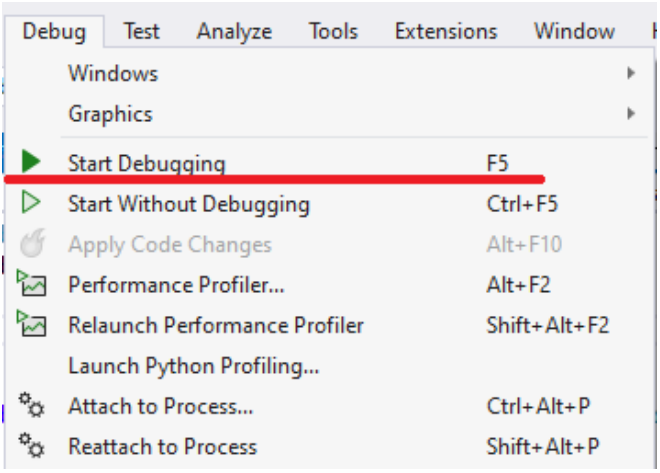
```
            columnName = Convert.ToChar(65 + modulo) +
columnName;
            dividend = (dividend - modulo) / 26;
        }

        return columnName;
    }


    // викликається, коли користувач вийде зі зміненої
клітинки (втратить фокус)
    private void Entry_Unfocused(object sender, FocusEventArgs e)
    {
        var entry = (Entry)sender;
        var row = Grid.GetRow(entry) - 1;
        var col = Grid.GetColumn(entry) - 1;
        var content = entry.Text;

        // Додайте додаткову логіку, яка виконується при виході
зі зміненої клітинки
    }

    private void CalculateButton_Clicked(object sender, EventArgs
e)
    {
        // Обробка кнопки "Порахувати"
    }

    private void SaveButton_Clicked(object sender, EventArgs e)
    {
        // Обробка кнопки "Зберегти"
    }

    private void ReadButton_Clicked(object sender, EventArgs e)
    {
        // Обробка кнопки "Прочитати"
    }

    private async void ExitButton_Clicked(object sender, EventArgs
e)
    {
        bool answer = await DisplayAlert("Підтвердження", "Ви
дійсно хочете вийти?", "Так", "Ні");
        if (answer)
```

```
        {
            System.Environment.Exit(0);
        }
    }

    private async void HelpButton_Clicked(object sender,
EventArgs e)
    {
        await DisplayAlert("Довідка", "Лабораторна робота 1.
Студента Василя Іваненка", "ОК");
    }

    private void DeleteRowButton_Clicked(object sender, EventArgs
e)
    {
        if (grid.RowDefinitions.Count > 1)
        {
            int lastRowIndex = grid.RowDefinitions.Count - 1;
            grid.RowDefinitions.RemoveAt(lastRowIndex);
            grid.Children.RemoveAt(lastRowIndex * (CountColumn +
1)); // Remove label
            for (int col = 0; col < CountColumn; col++)
            {
                grid.Children.RemoveAt((lastRowIndex * CountColumn)
+ col + 1); // Remove entry
            }
        }
    }
```

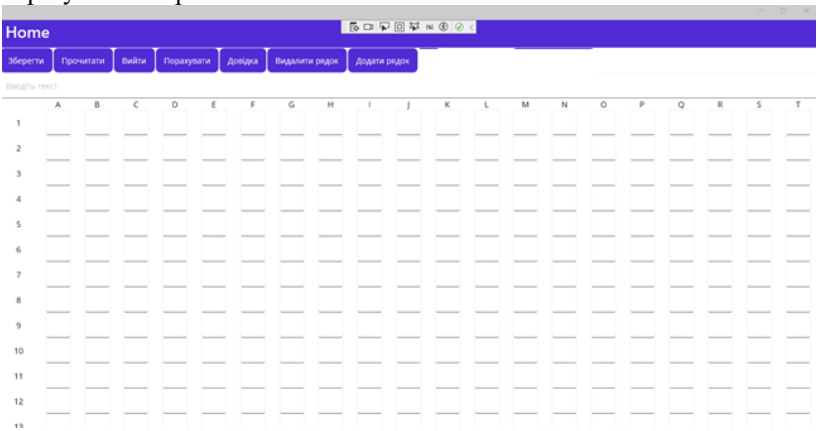
```
private void AddRowButton_Clicked(object sender, EventArgs e)
{
    int newRow = grid.RowDefinitions.Count;

    // Add a new row definition
    grid.RowDefinitions.Add(new RowDefinition());

    // Add label for the row number
    var label = new Label
    {
        Text = newRow.ToString(),
        VerticalOptions = LayoutOptions.Center,
        HorizontalOptions = LayoutOptions.Center
    };
    Grid.SetRow(label, newRow);
    Grid.SetColumn(label, 0);
    grid.Children.Add(label);

    // Add entry cells for the new row
    for (int col = 0; col < CountColumn; col++)
    {
        var entry = new Entry
        {
            Text = "",
            VerticalOptions = LayoutOptions.Center,
            HorizontalOptions = LayoutOptions.Center
        };

        entry.Unfocused += Entry_Unfocused;

        Grid.SetRow(entry, newRow);
        Grid.SetColumn(entry, col + 1);
        grid.Children.Add(entry);
    }
}
```

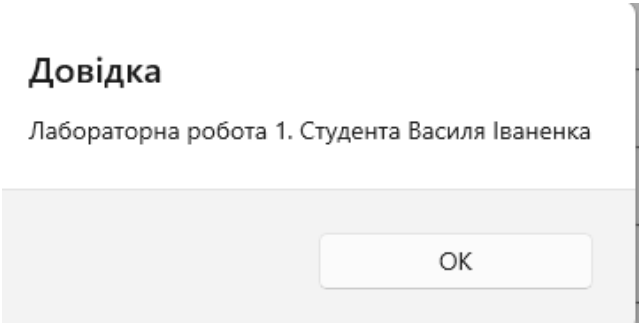
Запустіть проєкт:

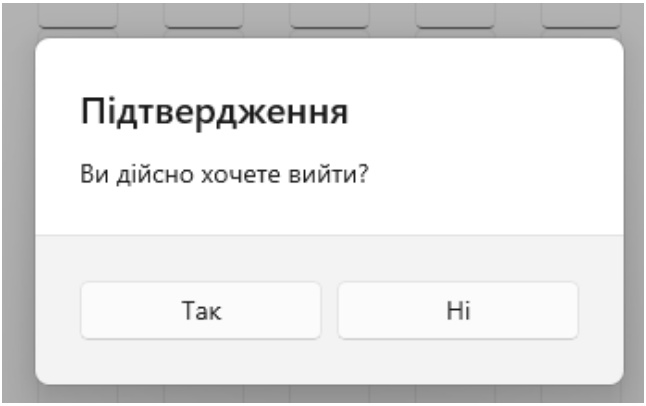




В результаті отримаємо:







Реалізуйте функціонал лабораторної роботи 1 (1*).

Детально з MAUI можна ознайомитися за посиланням: https://learn.microsoft.com/uk-ua/dotnet/maui/get-started/first-app?pivots=devices-windows&tabs=vswin

ТИЖДЕНЬ 8

Тема 8. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ C#

Огляд, мета і призначення теми: ознайомитися з функціональними можливостями C#. Лямбда-вирази. Застосування лямбда-виразів.

Вивчивши матеріал даної теми, студент зможе: вміти використовувати LINQ.

Огляд, мета і призначення теми: XML - мова розмітки документів. Основні компоненти XML-документа. Декларація XML. XML-елементи. Застосування XML. Моделі DOM та SAX для обробки XML. Рівні DOM. Порівняння технологій DOM та SAX. Робота з XML. Застосування LINQ to XML. Мова запитів XPath. Трансформація XML. Трансформація XML з використанням XSLT.

Лабораторний практикум 8

Робота з XML та JSON (для лабораторних робіт) (30 хв.), захист лабораторної роботи №1 (30 хв.), контрольна робота №1 (30 хв.).

Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторного заняття. Виконання домашнього завдання №6. Захист лабораторної роботи 1.

Завдання для опрацювання матеріалу:

Завдання для самостійної роботи

Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторного заняття. Виконання домашнього завдання №7. Проєктування та виконання лабораторної роботи №1*, 2

ТЕОРЕТИЧНА ЧАСТИНА ЛАБОРАТОРНОГО ЗАНЯТТЯ
РОБОТА З XML
XML – МОВА РОЗМІТКИ ДОКУМЕНТІВ

XML (англ. *Extensible Markup Language* – розширювана мова розмітки) – запропонований консорціумом World Wide Web (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними програмами.

Мова розмітки документів – це набір спеціальних інструкцій (*тегів*), що призначені для формування структури документів і визначення відношень між різними елементами цієї структури.

Теги являють собою *імена елементів* в кутових дужках.

Як правило, теги застосовуються в парі. Перший з яких називається *початковим тегом*, а другий – *кінцевим*. Наприклад: <пункт> </пункт>.

Документ мовою XML (*XML документ*) – набір XML елементів. Він називається *коректним*, якщо відповідає всім синтаксичним правилам XML.

Основні вимоги до XML документів:

- в заголовку документа міститься оголошення XML;
- існує єдиний *кореневий елемент* (тобто, текст та інші дані всього документа повинні бути розміщені між початковим кореневим тегом та відповідним йому заключним);
- кожному відкриваючому тегу, що визначає деяку область даних повинен відповідати закриваючий тег;
- в XML враховується регістр символів;
- всі значення атрибутів, що використовуються у визначенні тегів, повинні бути в лапках;
- при вкладеності тегів потрібно чітко дотримуватися порядку відкриваючих та закриваючих тегів.

XML документ складається з наступних основних компонент: декларація, теги, атрибути, посилання на сутності, інструкції з обробки, коментарі.

Додати коментар в XML документ дозволяє конструкція: позначаються:

<!-- Текст коментаря -->.

Вказівки щодо обробки документа (наприклад декларація) позначаються:

<? Вказівки ?>.

Для створення області документа, яку аналізатор буде розглядати як текст, ігноруючи спеціальні символи та інструкції, але, на відміну від коментарів, буде існувати можливість використовувати цей текст в програмі, потрібно помістити його між відкриваючим тегом **<![CDATA]** та закриваючим **]]>**. Вміст цього елемента не повинен містити послідовність символів **]]**.

В декларації документа вказується використана версія XML (атрибут **version**). Крім того, можуть бути зазначені деякі інші параметри, зокрема:

- атрибут **encoding** відповідає за вказання кодування символів. Якщо в тегах не використовуються не латинські літери (напр. українські, російські), то зручним є кодування UTF-8 (об'єм менший, ніж у UTF-16);
- атрибут **standalone, що відповідає за зовнішню залежність документа.** Про автономність документа (не імпортування інших документів) свідчить значення **"yes"**.

Приклад:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>.

Імена елементів, як і імена атрибутів, не можуть містити прогалики, але можуть містити літери (будь якої мови, що підтримується кодуванням документа вказаному в декларації), цифри, дефіс, крапка, підкреслювання, двокрапка. Починатися ім'я може з літери, підкреслювання чи двокрапки.

XML елементом називається комбінація початкового, кінцевого тегу та вмісту елемента. Елементи, які не мають вмісту, називаються *порожніми елементами*. Вмістом елементів може бути текст, інші, вкладені, елементи документа, секції CDATA, вказівки щодо обробки документа, коментарі.

XML елементи можуть мати список *атрибутів*, які задаються в початковому тегу після імені елемента та перераховуються через прогалик. Атрибути дозволяють задати параметри елемента. Вони складаються з *імені* та *значення атрибута*, значення атрибута обов'язково повинно міститися в лапках (**" "** чи **' '**). В іменах та значеннях атрибутів не можуть використовуватися такі спеціальні символи, як "**&**", "<" та ">".

<?xml version = "1.0" encoding="windows-1251" ?>

```
<libraryDatabase>
   <book>
      <title>"12 стільців"</title>
      <author>”Ільф”</author>
      <author>”Петров”</author>
   </book>
   <book>
      <title> "Золоте теля" </title>
      <author>”Ільф”</author>
      <author>”Петров”</author>
   </book>
</libraryDatabase>
```

Застосування XML

Налаштування програми в форматі XML

Зручно зберігати налаштування вашої програми в форматі XML. Синтаксис інтуїтивно зрозумілий, тому для редагування підходить довільний текстовий редактор, наявність бібліотек програмних компонент, які допомагають виконувати синтаксичний розбір. Крім того, збереження налаштувань в окремому файлі може позбавити необхідності внесення змін в код.

Наприклад, файли налаштувань проєктів App.config та Web.config.

XML – зручний інструмент для формування даних табличних звітів, які складаються з однієї чи більше табличних форм. Для форматування звіту в зручну для сприйняття форму можна застосовувати XML розширення – XSLT (трансформація), застосовуючи яке, можна налаштувати перетворення XML звіту в XHTML документ прийнятного вигляду. Документы Microsoft Office можна зберігати в формате XML. Для кожного застосування з Microsoft Office існує окрема схема документа, а відповідно і схема *трансформації*, яка перетворює звіт в документ Microsoft Excel чи Microsoft Word в форматі XML.

XML, в першу чергу створено для збереження та передачі наборів структурованої інформації.

Існує два основних підходи до обробки XML документів: DOM та SAX.

DOM (Document Object Model) – це технологія, яка базується на формуванні в оперативній пам’яті ієрархічних структур даних, які відповідають всьому XML документу. DOM – це побудова об’єктної

моделі документа (всі елементи та їх атрибути представляються в пам'яті окремими об'єктами).

SAX (Simple API for XML) – це аналіз вмісту XML документа в процесі послідовного читання даних файлу. Іншими словами, SAX - це найпростіший інтерфейс обробки XML, який не потребує додаткових витрат.

Ці два підходи не конкурують між собою, а доповнюють один одного, оскільки їх достоїнства та недоліки практично не перетинаються.

Переваги XML DOM:

▪ Простота. Усі вузли XML документа доступні одразу. Легко виконувати пошук вузлів.

▪ Можна легко редагувати XML документ: додавати, видаляти, переміщувати вузли, тощо.

Переваги SAX:

▪ Економія пам'яті.

▪ Повний контроль над процесом розбору документа.

▪ XML документ використовується для збереження кеша даних. Вміст цього кеша може постійно змінюватися в процесі виконання програми.

▪ Об'єми документів XML, з якими ви маєте справу, невеликі. Вам важлива прозорість та наглядність програмного коду формування XML документів. Швидкість виконання запитів вас влаштовує.

▪ Необхідно сформувати звіт невеликого обсягу.

▪ XML документ використовується, як деяка прикладна структура даних, наприклад, для представлення в пам'яті алгоритму роботи інтерпретатора, кеша метаданих чи іншої серіалізуємої структури.

▪ Потрібно прочитати всі чи здійснити пошук лише потрібних вам даних з достатньо об'ємного XML документа. Редагувати документ немає потреби. Представлення XML у вигляді об'єктної моделі занадто велике.

▪ Потрібно перекласти великі об'єми інформації довільного формату в XML представлення.

Робота з XML

XML-класи в просторі імен System.Xml утворюють повнофункціональний інтегрований набір класів, який дозволяє працювати з XML-документами та даними. XML-класи підтримують синтаксичний аналіз і запис XML-кода, зміну XML-даних в пам'яті, перевірку даних.

Обробка XML в пам’яті (DOM технологія)

Платформа .NET Framework пропонує три способи обробки даних XML в пам’яті: LINQ to XML, клас XPathNavigator та клас XmlDocument.

Обробка XML на основі потоку (SAX технологія)

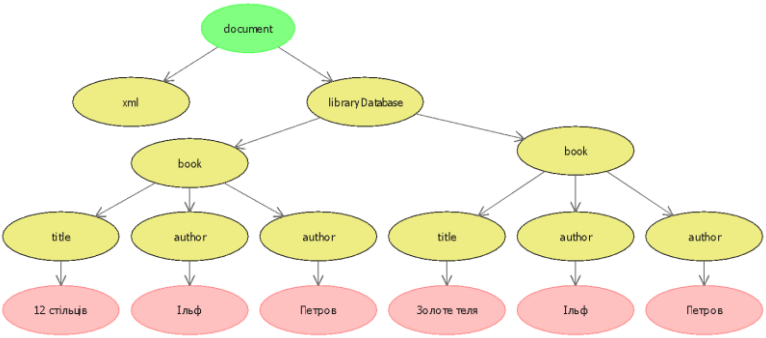
Класи XmlWriter та XmlReader забезпечують однопрохідний спосіб обробки XML-даних без кешування.

Модель DOM для обробки XML

Нехай є наступний XML документ:

```
<?xml version = "1.0" encoding="windows-1251" ?>
<libraryDatabase>
   <book>
      <title>"12 стільців"</title>
      <author>”Ільф”</author>
      <author>”Петров”</author>
   </book>
   <book>
      <title> "Золоте теля" </title>
      <author>”Ільф”</author>
      <author>”Петров”</author>
   </book>
</libraryDatabase>
```

Дерево представленого вище документу:



Кожен овал являє собою вузол в структурі XML-документа, що називається об’єктом XmlNode. Об’єкт XmlNode є базовим об’єктом дерева DOM. Клас XmlDocument, розширює клас XmlNode, підтримує методи для виконання операцій над документом (наприклад, завантаження його в пам’ять чи збереження XML в файл). Крім

того, XmlDocument надає можливості для перегляду вузлів всього XML-документу та виконання операцій над ними

Об’єкти Node мають набір методів та властивостей, а також базових, характеристик.

У кожного вузла є один батьківський вузол. Єдиний вузол, який не має батьківського — кореневий вузол документа, так як це вузол верхнього рівня, який містить сам документ та його фрагменти.

У більшості вузлів може бути декілька дочірніх вузлів. Дочірні вузли можуть мати наступні типи:

- Document
- DocumentFragment
- EntityReference
- Element
- Attribute

Вузли XmlDeclaration, Notation, Entity, CDATASection, Text, Comment, ProcessingInstruction та DocumentType не можуть мати дочірніх вузлів.

Вузли, які знаходяться на одному рівні, — як вузли title and author на схемі — називаються однорівневими.

Рівні DOM

Існує декілька версій DOM, які отримали назву рівнів (англ. Level). Кожен рівень складається із кількох обов'язкових та необов'язкових модулів. Для того, щоб стверджувати про підтримку DOM певного рівня, програма має задовольняти всім вимогам стандарту DOM заявленого рівня, та всім вимогам нижчих рівнів. Також, реалізація інтерфейсу може підтримувати певні розширення, якщо вони не суперечать вимогам стандарту. У 2005 році, рівні 1 та 2 (Level 1, Level 2) та деякі модулі 3-го рівня (Level 3) було визнано як W3C Recommendation, що означає, що вони набули кінцевої форми.

Level 0. Не було стандартизовано, став основою для появи DOM Level 1.

Level 1. Обхід структури (дерева) документа (XML та HTML), та модифікація змісту (включаючи додавання елементів). Також включаються специфічні елементи HTML.

Level 2. Підтримка просторів імен XML, фільтрованих представлень та подій.

Level 3. Складається із 6 різних специфікацій:

DOM Level 3 Core;

DOM Level 3 Load and Save;

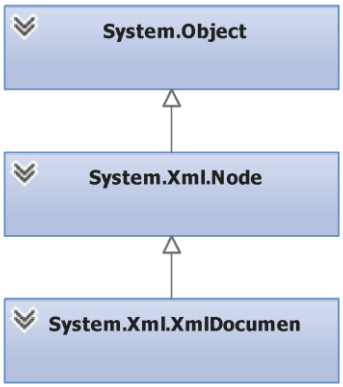
DOM Level 3 XPath;
DOM Level 3 Views and Formatting;
DOM Level 3 Requirements;
DOM Level 3 Validation.

Модель DOM розглядає XML-дані як стандартний набір об’єктів та використовується для обробки XML-даних в пам’яті. Простір **System.Xml** забезпечує програмне представлення XML-документів, фрагментів, вузлів та наборів вузлів. Воно базується на рекомендаціях базової моделі DOM рівня 1 та моделі DOM рівня 2 консорціуму W3C.

Для роботи з XML необхідно встановити посилання на **System.Xml.dll** файл і додати в код рядок підключення

using System.Xml;

Клас **XmlDocument** забезпечує представлення XML документа в пам'яті за допомогою DOM 1-го та 2-го рівнів. Цей клас може бути використаний для навігації та редагування XML-вузлів.



Розглянемо основні методи класу XmlDocument

Метод	Опис
CreateNode	Створює XML-вузол в документі. Є також спеціалізовані методи для створення кожного типу вузла, такі як CreateElement, або CreateAttribute.
CloneNode	Створює дублікат вузла XML. Цей метод приймає

	Boolean аргумент, який називається глибиною (deep). Якщо deep false, то копіюється лише вузол, якщо deep true, всі дочірні вузли також рекурсивно копіюються.
GetElementByID	Знаходить і повертає один вузол на основі ID атрибута.
GetElementsByTagName	Знаходить і повертає список XmlNodeList, що містить всі елементи-нащадки які відповідають імені.
ImportNode	Імпортує вузол з другого документа в поточний документ.
InsertBefore	Вставляє заданий вузол одразу перед зазначеним вузлом-посиланням. (Успадковано від XmlNode)
InsertAfter	Вставляє заданий вузол одразу після зазначеного вузла-посилання. (Успадковано від XmlNode)
Load	Завантажує XML документ з файлу, URL-адреси, XMLReader чи TextReader.
LoadXml	Завантажує XML документ з рядка.
Normalize	Розміщує усі вузли XmlText на максимальну глибину піддерева, розміщеного під даним вузлом XmlNode, в формат, де вузли XmlText розділяються лише розміткою (теги, коментарі, інструкції обробки, тощо). Суміжні вузли XmlText відсутні. (Успадковано від XmlNode)
PrependChild	Додає зазначений вузол в початок списку дочірніх вузлів даного вузла. (Успадковано від XmlNode)
ReadNode	Завантажує вузол з XML документа за допомогою XmlReader.
RemoveAll	Видаляє всі дочірні вузли і (чи) атрибути поточного вузла. (Успадковано від XmlNode)

RemoveChild	Видаляє зазначений дочірній вузол. (Успадковано від XmlNode)
ReplaceChild	Заміняє дочірній вузол oldChild на вузол newChild. (Успадковано від XmlNode)
Save	Зберігає XML документ в зазначеному файлі, потоці XmlWriter чи TextWriter.
SelectNodes	Вибирає список вузлів, які відповідають виразу XPath.
SelectSingleNode	Вибирає перший вузол, який відповідає виразу XPath.
WriteTo	Зберігає вузол XmlDocument в заданому XmlWriter.
WriteContentsTo	Зберігає всі дочірні елементи вузла XmlDocument в заданий XmlWriter.

Створення об'єкта XmlDocument

Для створення об'єкту XmlDocument, створимо екземпляр класу XmlDocument.

XmlDocument об'єкт містить CreateElement і CreateAttribute методи, які додають вузли в XmlDocument об'єкт. XmlElement містить властивості такі, серед яких XmlAttributeCollection. XmlAttributeCollection успадковується від XmlNamedNodeMap класу, який представляє собою набір імен з відповідними значеннями.

Наступний приклад показує, як створити XmlDocument клас та зберегти його у файл.

Не забудьте проімпортувати System.Xml (using System.Xml;) та System.IO (using System.IO;).

```
private static void createAndSaveXmlDocument()
    {
        //Створення нового XmlDocument
        var xmlDoc = new XmlDocument();
        XmlElement el;
        int childCounter;
        int grandChildCounter;
        //Створення xml декларації

xmlDoc.AppendChild(xmlDoc.CreateXmlDeclaration("1.0", "utf-8",
null));
//Створення кореневого елементу і додавання його в документ
        el = xmlDoc.CreateElement("MyRoot");
```

```
                xmlDoc.AppendChild(el);
        for (childCounter = 1; childCounter <= 3; childCounter++)
//дочірні елементи
          {
                    XmlElement childelmt;
                    XmlAttribute childattr;
                    //Створенння дочірніх елементів з ID
                    childelmt = xmlDoc.CreateElement("MyChild");
                    childattr = xmlDoc.CreateAttribute("ID");
                    childattr.Value = childCounter.ToString();
                    childelmt.Attributes.Append(childattr);
                    //Додавання елементу в кореневий елемент
                    el.AppendChild(childelmt);
                    for (grandChildCounter = 1; grandChildCounter <=
4; grandChildCounter++)        {
//Створення дочірніх вузлів для дочірніх вузлів
                      XmlElement grandchilde;
                      XmlAttribute grandchildeattr;
                      grandchilde                                    =
xmlDoc.CreateElement("MyGrandChild");
                      grandchildeattr                                =
xmlDoc.CreateAttribute("NAME");
                      grandchildeattr.Value = childattr.Value + " " +
grandChildCounter.ToString();
                      grandchilde.Attributes.Append(grandchildeattr);
                      childelmt.AppendChild(grandchilde);              }
}
        //Збереження файлу
                xmlDoc.Save(getFilePath("XmlDocumentTest.xml"));
                Console.WriteLine("XmlDocumentTest.xml
Created\r\n");
            }
            //Шлях на робочий стіл
            private static string getFilePath(string fileName)
            {
               return Path.Combine(Environment.GetFolderPath(

                  Environment.SpecialFolder.Desktop), fileName);
            }
            public static
               void Main(string[] args)
            {
               createAndSaveXmlDocument();
               Console.ReadKey();
```

```
}
```

Згенеровано файл:

```
<?xml version="1.0" encoding="utf-8"?>
<MyRoot>
  <MyChild ID="1">
    <MyGrandChild NAME="1 1" />
    <MyGrandChild NAME="1 2" />
    <MyGrandChild NAME="1 3" />
    <MyGrandChild NAME="1 4" />
  </MyChild>
  <MyChild ID="2">
    <MyGrandChild NAME="2 1" />
    <MyGrandChild NAME="2 2" />
    <MyGrandChild NAME="2 3" />
    <MyGrandChild NAME="2 4" />
  </MyChild>
  <MyChild ID="3">
    <MyGrandChild NAME="3 1" />
    <MyGrandChild NAME="3 2" />
    <MyGrandChild NAME="3 3" />
    <MyGrandChild NAME="3 4" />
  </MyChild>
</MyRoot>
```

Розбір об'єкта XmlDocument за допомогою DOM

Об'єкт XmlDocument може бути проаналізований рекурсивним перебором елементі. Наступний код містить приклад розбору XmlDocument.

Зверніть увагу на імпорт System.Text (using System.Text;)

```
private static void parsingXmlDocument()
    {
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.Load(getFilePath("XmlDocumentTest.xml"));
        RecurseNodes(xmlDoc.DocumentElement);
    }
    private static void RecurseNodes(XmlNode node)
    {
        var sb = new StringBuilder();
        //починаємо рекурсивний перегляд з рівня 0
        RecurseNodes(node, 0, sb);
        //друкуємо сформований рядок
        Console.WriteLine(sb.ToString());
```

```
        }
    private static void RecurseNodes(XmlNode node, int level,
StringBuilder sb)
        {
            sb.AppendFormat("{0,-2}  Type:{1,-9}  Name:{2,-13}
Attr:",
            level, node.NodeType, node.Name);
            foreach (XmlAttribute attr in node.Attributes)
            {
                sb.AppendFormat("{0}={1} ", attr.Name, attr.Value);
            }
            sb.AppendLine();
            foreach (XmlNode n in node.ChildNodes)
            {
                RecurseNodes(n, level + 1, sb);
            }
        }
    //Шлях на робочий стіл
        private static string getFilePath(string fileName)
        {
            return Path.Combine(Environment.GetFolderPath(
            Environment.SpecialFolder.Desktop), fileName);
        }

      public static void Main(string[] args)
      {
            parsingXmlDocument();
            Console.ReadKey();
      }
```

```
0  Type:Element   Name:MyRoot        Attr:
1  Type:Element   Name:MyChild       Attr:ID=1
2  Type:Element   Name:MyGrandChild  Attr:NAME=1 1
2  Type:Element   Name:MyGrandChild  Attr:NAME=1 2
2  Type:Element   Name:MyGrandChild  Attr:NAME=1 3
2  Type:Element   Name:MyGrandChild  Attr:NAME=1 4
1  Type:Element   Name:MyChild       Attr:ID=2
2  Type:Element   Name:MyGrandChild  Attr:NAME=2 1
2  Type:Element   Name:MyGrandChild  Attr:NAME=2 2
2  Type:Element   Name:MyGrandChild  Attr:NAME=2 3
2  Type:Element   Name:MyGrandChild  Attr:NAME=2 4
1  Type:Element   Name:MyChild       Attr:ID=3
2  Type:Element   Name:MyGrandChild  Attr:NAME=3 1
2  Type:Element   Name:MyGrandChild  Attr:NAME=3 2
2  Type:Element   Name:MyGrandChild  Attr:NAME=3 3
2  Type:Element   Name:MyGrandChild  Attr:NAME=3 4
```

Пошук в XmlDocument

У наступному прикладі для пошуку елементу з ID = 2 викличемо метод SelectSingleNode за допомогою запиту XPath. Додамо цей метод до попереднього класу:

```
private static void searchingInXmlDocument()
    {
        var xmlDoc = new XmlDocument();
        xmlDoc.Load(getFilePath("XmlDocumentTest.xml"));
        var node =
xmlDoc.SelectSingleNode("//MyChild[@ID='2']");
        RecurseNodes(node);
    }
```

```
0  Type:Element   Name:MyChild       Attr:ID=2
1  Type:Element   Name:MyGrandChild  Attr:NAME=2 1
1  Type:Element   Name:MyGrandChild  Attr:NAME=2 2
1  Type:Element   Name:MyGrandChild  Attr:NAME=2 3
1  Type:Element   Name:MyGrandChild  Attr:NAME=2 4
```

Метод GetElementsByTagName повертає XmlNode список, що містить всі елементи набору. Наступний код повертає список вузлів з іменем тегу MyGrandChild.

```
private static void getElementsByTagName()
    {
        var xmlDoc = new XmlDocument();
        xmlDoc.Load(getFilePath("XmlDocumentTest.xml"));
        var elmts =
xmlDoc.GetElementsByTagName("MyGrandChild");
        var sb = new StringBuilder();
        foreach (XmlNode node in elmts)
        {
            RecurseNodes(node, 0, sb);
        }
        Console.WriteLine(sb.ToString());
    }
```

```
0  Type:Element   Name:MyGrandChild  Attr:NAME=1 1
0  Type:Element   Name:MyGrandChild  Attr:NAME=1 2
0  Type:Element   Name:MyGrandChild  Attr:NAME=1 3
0  Type:Element   Name:MyGrandChild  Attr:NAME=1 4
0  Type:Element   Name:MyGrandChild  Attr:NAME=2 1
0  Type:Element   Name:MyGrandChild  Attr:NAME=2 2
0  Type:Element   Name:MyGrandChild  Attr:NAME=2 3
0  Type:Element   Name:MyGrandChild  Attr:NAME=2 4
0  Type:Element   Name:MyGrandChild  Attr:NAME=3 1
0  Type:Element   Name:MyGrandChild  Attr:NAME=3 2
0  Type:Element   Name:MyGrandChild  Attr:NAME=3 3
0  Type:Element   Name:MyGrandChild  Attr:NAME=3 4
```

Пошук в XmlDocument (DOM)

За допомогою методу SelectNodes, який вимагає запиту XPath, що передаються в метод, можна також отримати список XmlNode. Змінимо попередній приклад, щоб з використанням методу SelectNodes досягти того ж результату:

```
private static void selectNodesTool()
    {
        var xmlDoc = new XmlDocument();
        xmlDoc.Load(getFilePath("XmlDocumentTest.xml"));
    var elmts = xmlDoc.SelectNodes("//MyGrandChild");
        var sb = new StringBuilder();
        foreach (XmlNode node in elmts)
        {
            RecurseNodes(node, 0, sb);
        }
        Console.WriteLine(sb.ToString());
    }
```

Клас XmlTextReader (SAX)

Клас XmlTextReader це абстрактний базовий клас, який надає методи для читання і синтаксичного аналізу XML.

Клас XmlTextReader забезпечує найшвидший і найекономічніший до пам'яті спосіб для читання і

розбору XML-даних. Наступний код зчитує XML-файл, створений у попередньому прикладі, і відображає інформацію про кожен вузол:

```
private static void parsingWithXmlTextReader()
    {
        var sb = new StringBuilder();
        var xmlReader = new
XmlTextReader(getFilePath("XmlDocumentTest.xml"));
        while (xmlReader.Read())
        {
switch (xmlReader.NodeType)
            {
                case XmlNodeType.XmlDeclaration:
                case XmlNodeType.Element:
                case XmlNodeType.Comment:
                    sb.AppendFormat("{0}: {1} = {2}",
                                xmlReader.NodeType,
                                xmlReader.Name,
                                xmlReader.Value);
```

```
                    sb.AppendLine();
                    break;
                case XmlNodeType.Text:
                    sb.AppendFormat(" - Value: {0}", xmlReader.Value);
                    sb.AppendLine();
                    break;
            }
        if (xmlReader.HasAttributes)
            {
                while (xmlReader.MoveToNextAttribute())
                {
                    sb.AppendFormat(" - Attribute: {0} = {1}",
                                xmlReader.Name,
                                xmlReader.Value);
                    sb.AppendLine();
                }
            }
        }
        xmlReader.Close();
        Console.WriteLine(sb.ToString());
    }
```

```
XmlDeclaration: xml = version="1.0" encoding="utf-8"
 - Attribute: version = 1.0
 - Attribute: encoding = utf-8
Element: MyRoot =
Element: MyChild =
 - Attribute: ID = 1
Element: MyGrandChild =
 - Attribute: NAME = 1 1
Element: MyGrandChild =
 - Attribute: NAME = 1 2
Element: MyGrandChild =
 - Attribute: NAME = 1 3
Element: MyGrandChild =
 - Attribute: NAME = 1 4
Element: MyChild =
 - Attribute: ID = 2
Element: MyGrandChild =
 - Attribute: NAME = 2 1
Element: MyGrandChild =
 - Attribute: NAME = 2 2
Element: MyGrandChild =
 - Attribute: NAME = 2 3
Element: MyGrandChild =
 - Attribute: NAME = 2 4
Element: MyChild =
 - Attribute: ID = 3
Element: MyGrandChild =
 - Attribute: NAME = 3 1
Element: MyGrandChild =
 - Attribute: NAME = 3 2
Element: MyGrandChild =
 - Attribute: NAME = 3 3
Element: MyGrandChild =
 - Attribute: NAME = 3 4
```

LINQ to XML

LINQ to XML забезпечує інтерфейс програмування для роботи з XML в пам’яті на основі платформи .NET LINQ Framework.

LINQ to XML подібний до моделі DOM в тому відношенні, що завантажує XML-документ в пам'ять. До такого документу можна направити запит, його можна змінити, а після зміни його можна зберегти.

Інтеграція з LINQ надає можливість створювати запити до завантаженого в пам'ять XML-документа з метою отримання колекцій елементів та атрибутів.

Для представлення XML-документа в System.Xml.Linq визначено клас XDocument.

Для представлення XML-елементів в System.Xml.Linq визначено клас XElement.

XML- атрибутів в System.Xml.Linq визначено клас XAttribute.

Для представлення простору імен XML в System.Xml.Linq визначено клас XNamespace.

Для використання цих класів потрібно додати посилання на збірку System.Xml.Linq.dll, а потім додати using System.Xml.Linq.

LINQ to XML. Клас XObject

Абстрактний клас. Представляє вузол чи атрибут в XML-дереві. Містить наступні елементи, які будуть успадковані:

BaseUri	Отримує базовий універсальний код ресурса (uri).
Document	Повертає XDocument
NodeType	Повертає тип вузла
Parent	Повертає батьківський елемент Xelement
AddAnnotation	Додає об'єкт до списку анотацій
Annotation	Повертає перший об'єкт анотації зазначеного типу
Annotations	Повертає колекцію анотацій зазначеного типу
RemoveAnnotations	Видаляє анотації зазначеного типу
Changed	Викликається, коли даний об'єкт, чи довільний з його нащадків були змінені.

LINQ to XML. Клас XAttribute

Представляє атрибути. Є похідним від XObject, а отже успадковує всі його елементи. Крім того, містить наступні елементи.

EmptySequence	Отримання порожньої колекції атрибутів
IsNameSpaceDeclaration	Визначає, чи є цей атрибут оголошенням простору імен

Name	Розгорнуте ім’я атрибута
NextAttribute	Отримання наступного атрибута батьківського елемента
NodeType	Тип вузла для цього елемента
PreviousAttribute	Попередній атрибут батьківського елемента
Value	Значення цього атрибута
Remove	Видаляє атрибут з батьківського елемента
SetValue	Задає значення атрибуту
ToString	Перетворює поточний об’єкт XAttribute до рядка

LINQ to XML. Клас XNode

Представляє вузли. Є похідним від XObject, а отже успадковує всі його елементи. Крім того, містить наступні елементи:

DocumentOrderComparer	Отримує об’єкт компаратор, який може порівняти відносну позицію двох елементів.
EqualityComparer	Отримує об’єкт компаратор, який може порівняти два вузли на рівність значень.
NextNode	Наступний споріднений вузол.
PreviousNode	Попередній споріднений вузол.
AddAfterSelf	Додає зазначений вміст безпосередньо після даного вузла.
AddBeforeSelf	Додає зазначений вміст безпосередньо перед даним вузлом.
Ancestors	Повертає колекцію елементів попередників даного вузла
CompareDocumentOrder	Порівнює два вузли з метою визначення їх відносного порядку.
CreateReader	Створює об’єкт XmlReader для даного вузла
DeepEquals	Порівнює значення двох вузлів, включаючи значення усіх підлеглих вузлів.
ElementsAfterSelf	Повертає колекцію споріднених елементів після цього вузла

ElementsBeforeSelf	Повертає колекцію споріднених елементів перед цим вузлом
IsAfter	Визначає, чи з'являється поточний вузол після зазначеного вузла
IsBefore	Визначає, чи з'являється поточний вузол перед зазначеним вузлом
NodeAfterSelf	Повертає колекцію споріднених вузлів після даного вузла
NodeBeforeSelf	Повертає колекцію споріднених вузлів перед даним вузлом
ReadFrom	Створює об'єкт XNode з XmlReader.
Remove	Видаляє вузол з батьківського об'єкта.
ReplaceWith	Заміняє даний вузол на зазначений вміст.
ToString	Повертає призначений для даного вузла XML.
WriteTo	Записує даний вузол в XmlWriter.

LINQ to XML. Клас XContainer

Абстрактний клас. Вузол, який може містити інші вузли. Похідний від класу XNode. Він успадковує члени XNode та XObject. XDocument і XElement класи успадковують від цього класу. Нижче наведено список його членів.

FirstNode	Повертає перший дочірній вузол
LastNode	Повертає останній дочірній вузол.
Add	Додає заданий вміст, як дочірні елементи.
AddFirst	Додає заданий вміст, як перші дочірні елементи.
CreateWriter	Створює XmlWriter який можна використовувати для додавання вузлів в XContainer
DescendantNodes	Повертає колекцію вузлів, які є потомками в порядку слідування документа
Descendants	Повертає колекцію елементів, які є потомками в порядку слідування документа
Element	Отримує перший дочірній елемент з зазначеним XName
Elements	Повертає фільтровану колекцію дочірніх елементів з зазначеним XName в порядку слідування документа.

Nodes	Повертає колекцію дочірніх вузлів в порядку слідування документа
RemoveNodes	Видаляє дочірні вузли
ReplaceNodes	Заміняє дочірні вузли зазначеним вмістом

LINQ to XML. Клас XElement

Клас XElement представляє XML елемент з іменем властивості типу XName. XName клас складається з локального імені та простору імент Додатково XElement містять XML атрибути типу XAttribute. Клас XElement походить від XContainer і успадковує всі елементи з XContainer, XNode, XObject. Крім того, XElement явно реалізує інтерфейс IXmlSerializable, який містить GetSchema, ReadXml, WriteXml методи. Таким чином, цей об'єкт може бути серіалізований. Наступні елементи є членами класу XElement.

LINQ to XML. Клас XElement

EmptySequence	Повертає порожню колекцію елементів
FirstAttribute	Повертає перший атрибут елемента
HasAttributes	Отримує значення, яке вказує, чи має даний елемент принаймні один атрибут.
HasElements	Отримує значення, яке вказує, чи має даний елемент принаймні один елемент.
IsEmpty	Повертає значення, яке вказує, чи даний елемент не має вмісту
LastAttribute	Отримує останній атрибут
Name	Отримує чи задає назву
NodeType	Повертає тип вузла для даного вузла.
Value	Отримує чи задає значення тексту
AncestorsAndSelf	Повертає колекцію елементів (можливе задання фільтру по імені), які містять цей елемент і елементів предків.
Attribute	Повертає XAttribute цього елемента
Attributes	Повертає колекцію атрибутів цього елемента (можливо фільтровано по імені)

DescendantNodesAndSelf	Повертає колекцію вузлів, які містять цей елемент і усі вузли, які є нащадками цього елемента в порядку слідування в документі
DescendantsAndSelf	Повертає колекцію елементів, що містять цей елемент і усі елементи-потомки цього елемента в порядку слідування в документі.
GetDefaultNamespace	Отримує значення по замовчуванню XNamespace
GetPrefixOfNamespace	Повертає префікс простору імен, пов'язаний з простором імен даного елемента.
Load	Завантажує XElement з файлу чи потоку.
Parse	Завантаження з рядка, що містить XML
RemoveAll	Видаляє вузли та атрибути з елемента
RemoveAttributes	Видаляє атрибути
ReplaceAll	Заміняє дочірні вузли та атрибути цього елемента зазначеним вмістом
ReplaceAttributes	Заміняє атрибути цього елемента вказаним вмістом
Save	Виводить даний елемент в Stream, файл, TextWriter чи серіалізує
SetElementValue	Задає значення дочірнього атрибута
SetValue	Встановлює значення елемента

LINQ to XML. Клас XDocument

Клас XDocument являє собою XML-документ, який може містити DTD, XML один корінь.

Елемент, XML коментарі, та інструкції обробки XML.

Клас XDocument є похідним від XContainer і, отже, успадковує члени XContainer, XNode, і XObject. Наступні елементами є членами класу XDocument:

Declaration	Повертає чи задає оголошення XML для документа
DocumentType	Отримує визначення типу документа (DTD)
NodeType	Повертає тип вузла для даного вузла

Root	Повертає кореневий елемент дерева XML для цього документа
Load	Створює новий XDocument з файла, TextReader, XmlReader.
Parse	Створює новий XDocument з рядка
Save	Виводить дані в Stream, TextWriter, XmlWriter чи серіалізує.
WriteTo	Записує документ

Наступний приклад використовує метод Parse для завантаження XML рядка в об'єкт XDocument. Після цього, метод Save зберігає XML документ. Не забудьте додати using System.Xml.Linq;

```
        private static void parseXDocument()     {
                string xml = @"
                     <libraryDatabase>
                       <book BK_ID = '1' BK_NAME = 'Програмування'
        BK_INFO = 'Інформація про Програмування' BK_DC = '2' DC_NAME
= 'ПРОГРАМУВАННЯ' >
                        <author AU_ID = '1' AU_NAME = 'Зубенко В.В.'/>
                        <author AU_ID = '2' AU_NAME = 'Омельчук Л.Л.'/>
                            </book>
                      <book BK_ID = '2' BK_NAME = 'Математична логіка'
        BK_INFO = 'Інформація про МЛ' BK_DC = '2' DC_NAME = 'Логіка' >
                      <author AU_ID = '3' AU_NAME = 'Нікітченко М.С.'/>
                      <author AU_ID = '4' AU_NAME = 'Шкільняк С.С.'/>
                            </book>
                    <book BK_ID = '2' BK_NAME = 'Програмна інженерія'
        BK_INFO = 'Інформація про Програмна інженерія' BK_DC = '2'
        DC_NAME = 'ПРОГРАМУВАННЯ' >
                              <author AU_ID = '5' AU_NAME = 'Лавріщева К.М.'/>
                            </book>
                           </libraryDatabase> ";
        var doc = XDocument.Parse(xml);
                doc.Save(getFilePath("XDocumentTest.xml"));
                Console.WriteLine("XDocument Saved");
            }
        //Шлях на робочий стіл
              private static string getFilePath(string fileName)
              {
                 return Path.Combine(Environment.GetFolderPath(
                 Environment.SpecialFolder.Desktop), fileName);
              }
              public static   void Main(string[] args)
              {
                 parseXDocument();
                 Console.ReadKey();
```

```
}
```

Файл має вигляд:

```
<?xml version="1.0" encoding="utf-8"?>
<libraryDatabase>
  <book BK_ID="1" BK_NAME="Програмування" BK_INFO="Інформація про Програмування" BK_DC="2" DC_NAME="ПРОГРАМУВАННЯ">
    <author AU_ID="1" AU_NAME="Зубенко В.В." />
    <author AU_ID="2" AU_NAME="Омельчук Л.Л." />
  </book>
  <book BK_ID="2" BK_NAME="Математична логіка" BK_INFO="Інформація про МЛ" BK_DC="2" DC_NAME="Логіка">
    <author AU_ID="3" AU_NAME="Нікітченко М.С." />
    <author AU_ID="4" AU_NAME="Шкільняк С.С." />
  </book>
  <book BK_ID="2" BK_NAME="Програмна інженерія" BK_INFO="Інформація про Програмна інженерія" BK_DC="2" DC_NAME="ПРОГРАМУВАННЯ">
    <author AU_ID="5" AU_NAME="Лавріщева К.М." />
  </book>
</libraryDatabase>
```

Можна було створити XDocument і по-іншому:

```
  private static void xDocumentConstructor()
      {
          var doc = new XDocument(
              new XElement("libraryDatabase",
                      new XElement("book",
                              new XAttribute("BK_ID", "1"),
                              new XAttribute("BK_NAME", "Програмування"),
                      new XAttribute("BK_INFO", "Інформація про Програмування"),
                              new XAttribute("BK_DC", "1"),
                              new XAttribute("DC_NAME", "ПРОГРАМУВАННЯ"),
                              new XElement("author",
                                      new XAttribute("AU_ID", "1"),
                                      new XAttribute("AU_NAME", "Зубенко В.В.")),
                              new XElement("author",
                                      new XAttribute("AU_ID", "2"),
                                      new XAttribute("AU_NAME", "Омельчук Л.Л."))),
new XElement("book",
                              new XAttribute("BK_ID", "2"),
                              new XAttribute("BK_NAME", "Математична логіка"),
                              new XAttribute("BK_INFO", "Інформація про МЛ"),
                              new XAttribute("BK_DC", "2"),
                              new XAttribute("DC_NAME", "Логіка"),
                              new XElement("author",
```

```
                                    new XAttribute("AU_ID", "3"),
                                new      XAttribute("AU_NAME",      "Нікітченко
М.С.")),
                                new XElement("author",
                                    new XAttribute("AU_ID", "4"),
                                  new      XAttribute("AU_NAME",      "Шкільняк
С.С."))),
new XElement("book",
                                new XAttribute("BK_ID", "3"),
                               new         XAttribute("BK_NAME",         "Програмна
інженерія"),
            new      XAttribute("BK_INFO",      "Інформація      про      Програмну
інженерію"),
                                new XAttribute("BK_DC", "1"),
                                new                              XAttribute("DC_NAME",
"ПРОГРАМУВАННЯ"),
                                new XElement("author",
                                    new XAttribute("AU_ID", "5"),
                                new      XAttribute("AU_NAME",      "Лавріщева
К.М.")))
                    ));
        doc.Save(getFilePath("XDocumentTest.xml"));
        Console.WriteLine("XDocument Saved");
    }
```

Пошук елементів

```
private static void lINQQuery()
    {
        var doc = XDocument.Load(getFilePath("XDocumentTest.xml"));
        var result = (from book in doc.Descendants("book")
                    where book.Attribute("BK_ID").Value == "1"
                    select new   {
                        bk_id = (int)book.Attribute("BK_ID"),
                        bk_name = (string)book.Attribute("BK_NAME"),
                        bk_info = (string)book.Attribute("BK_INFO"),
                        bk_dc = (int)book.Attribute("BK_DC"),
                        dc_name = (string)book.Attribute("DC_NAME")
                    }).FirstOrDefault();
Console.WriteLine(string.Format("BK_ID:{0}\nBK_NAME:{1}      \nBK_INFO:{2}
\nDC_NAME:{3}",
            result.bk_id, result.bk_name, result.bk_info, result.dc_name));
    }
private static void lINQQuery1()
    {
        var doc = XDocument.Load(getFilePath("XDocumentTest.xml"));
        var result = (from book in doc.Descendants("book")
                    where book.Attribute("BK_DC").Value == "1"
                    select new
                    {
                        bk_id = (int)book.Attribute("BK_ID"),
```

```
                    bk_name = (string)book.Attribute("BK_NAME"),
                    bk_info = (string)book.Attribute("BK_INFO"),
                    bk_dc = (int)book.Attribute("BK_DC"),
                    dc_name = (string)book.Attribute("DC_NAME")
                }).ToList();
        foreach (var b in result)
        {
            Console.WriteLine(string.Format("BK_ID:{0}\nBK_NAME:{1}
\nBK_INFO:{2} \nDC_NAME:{3}",
            b.bk_id, b.bk_name, b.bk_info, b.dc_name));
        }
    }
```

Пошук категорії для автора з ідентифікатором 4:

```
 private static void lINQQuery2()
    {
        var doc = XDocument.Load(getFilePath("XDocumentTest.xml"));
        var result = (from book in doc.Descendants("book")
            where book.Descendants("author").Attributes("AU_ID").Any(z =>
z.Value=="4")
                    select new
                    {
                        bk_dc = (int)book.Attribute("BK_DC"),
                        dc_name = (string)book.Attribute("DC_NAME")
                    }).Distinct().ToList();
        foreach (var b in result)
        {
            Console.WriteLine(string.Format("BK_DC:{0}\nDC_NAME:{1}",
            b.bk_dc, b.dc_name));
        }
    }
```

ТРАНСФОРМАЦІЯ XML
Мова запитів XPath

XPath (XML Path Language) — мова запитів до елементів XML-документа. Розроблена для організації доступу до частин документа XML в файлах трансформації XSLT і є стандартом консорціуму W3C. XPath реалізує навігацію по DOM в XML.

По відношенню до XML документа XPath виступає в тій же ролі, що і SQL по відношенню до реляційної бази даних.

Рядок XPath – це шлях до деякого вузла чи вузлів XML документа відносно заданого контексту. Контекстом запиту може бути сам документ, а також кореневий чи довільний інший елемент документа.

XML має деревовидну структуру.

В документі завжди є кореневий елемент (інструкція <?xml version="1.0"?> до дерева відношення не має).

У елемента дерева завжди існують нащадки та потомки, крім кореневого елемента, а також листків дерева.

Кожен елемент дерева знаходиться на певному рівні вкладеності.

У елементів на одному рівні бувають попередні та наступні елементи.

```
<?xml version="1.0" encoding="utf-8"?>
<libraryDatabase>
 <book BK_ID="1" BK_NAME="Програмування" BK_INFO="Інформація про Програмування" BK_DC="1" DC_NAME="ПРОГРАМУВАННЯ">
  <author AU_ID="1" AU_NAME="Зубенко В.В." />
  <author AU_ID="2" AU_NAME="Омельчук Л.Л." />
 </book>
 <book BK_ID="2" BK_NAME="Математична логіка" BK_INFO="Інформація про МЛ" BK_DC="2" DC_NAME="Логіка">
  <author AU_ID="4" AU_NAME="Нікітченко М.С." />
  <author AU_ID="4" AU_NAME="Шкільняк С.С." />
 </book>
 <book BK_ID="2" BK_NAME="Математична логіка" BK_INFO="Інформація про МЛ" BK_DC="2" DC_NAME="Логіка">
  <author AU_ID="4" AU_NAME="Нікітченко М.С." />
  <author AU_ID="4" AU_NAME="Шкільняк С.С." />
 </book>
 <book BK_ID="3" BK_NAME="Програмна інженерія" BK_INFO="Інформація про Програмну інженерію" BK_DC="1" DC_NAME="ПРОГРАМУВАННЯ">
  <author AU_ID="5" AU_NAME="Лавріщева К.М." />
```

```
  </book>
</libraryDatabase>
```

Запит (шлях) XPath:

```
 /libraryDatabase/book/author[@AU_ID]
```

Аналіз здійснюється зліва направо.

Запит (шлях) ділиться на кроки адресації, які розділяються символом «/». Шлях починається відносно деякого контексту. Кожен крок складається з трьох частин:

На кожному кроці зазначається **критерій відбору вузлів**. Таким критерієм можуть виступати імена вузлів (елементів, атрибутів) чи спеціальні позначення множин вузлів, наприклад text() (усі текстові вузли) чи node() чи * - довільний вузол з підмножини вузлів поточного кроку.

На кожному кроці є можливість визначити **додаткові критерії відбору** (предикат кроку). Вираз предикату зазначається в квадратних дужках.

На кожному кроці обов'язково зазначається **вісь кроку** (підмножина вузлів документа, яка визначається контекстом, на якій здійснюється пошук).

В попередньому прикладі:

Крок 1: контекст = документ, вісь=дочірні вузли (елемент документа), критерій відбору = "libraryDatabase"

Крок 2: контекст = елемент libraryDatabase, вісь=дочірні вузли (елемент документа), критерій відбору = "book".

Крок 3: контекст = елемент book, вісь=дочірні вузли (елемент документа), критерій відбору = "author", предикат attribute::AU_ID.

Значенням осі по замовчуванню на кожному кроці пошуку є перехід до дочірніх для поточного контекста вузлів і позначається child з двома двокрапками: child::.

Осі:

ancestor:: — множина предків.

ancestor-or-self:: — множина предків та поточний елемент.

attribute:: — множина атрибутів поточного елемента (скорочення «@»).

child:: — множина нащадків поточного елемента (часто випускають).

descendant:: — повна множина нащадків (скорочення «.//»).

descendant-or-self:: —повна множина нащадків та поточний елемент.

following:: — необроблена множина, нижче поточного елемента.

following-sibling:: — множина елементів на одному рівні з поточним.

namespace:: — повертає множину, яка має простір імен (присутній атрибут xmlns).

parent:: — предок на один рівень назад (скорочення «..»).

preceding:: — повертає множину оброблених елементів включаючи множину предків.

preceding-sibling:: — множина елементів, які знаходяться на одному рівні з поточним та передують йому.

self:: — поточний елемент (скорочення «.»).

Базові вирази для представлення шляху:

Поточний контекст. Вираз з префіксом «./» явним чином використовує в якості контекста поточний контекст.

Наприклад ./book (аналогічно book)

Корінь документа. Вираз з префіксом «/» використовує в якості контексту корінь дерева.

Наприклад /libraryDatabase

Кореневий елемент. Вираз «/*» використовує в якості контексту кореневий елемент.

Рекурсивний спуск. Вираз «//», вказує на пошук, який може включати нуль чи більше рівнів ієрархії. Якщо зазначено на початку шаблону, то шлях відносний по відношенню до кореня.

Наприклад //book (аналогічно book)

Конкретний елемент. Вираз, який починається з імені елемента, посилається на запит конкретного елемента, який починається з поточного вузла контекста.

Наприклад libraryDatabase /book (колекція елементів book всередині libraryDatabase)

Вирази XPath створюються за допомогою операторів та спеціальних символів:

/	Оператор «дочірній елемент». Якщо цей оператор шляху стоїть на початку шаблону, то будуть вибрані усі дочірні елементи кореневого вузла.
//	Рекурсивний спуск; пошук заданого елемента на довільній глибині. Якщо цей оператор шляху стоїть на початку шаблону, рекурсивний спуск буде проводитись з кореневого вузла.
.	Поточний контекст.
..	Батьківський вузол поточного контексту.

*****	Символ підстановки; обирає усі елементи, незалежно від імені.
@	Атрибут; префікс імені атрибута.
@*	Символ підстановки для атрибута; вибирає всі атрибути незалежно від імені.
:	Роздільник простору імен. Відділяє префікс простору імені від самого імені елемента чи атрибута.
( )	Групує операції для явного задання порядку їх виконання.
[ ]	Застосовує шаблон фільтру.
[ ]	Оператор Subscript; використовується для індексування колекції.
+	Додавання.
-	Віднімання.
div	Ділення з плаваючою точкою.
*****	Множення.
mod	Залишок від ділення.
Логічні та множинні оператори:	
and	Логічне І
or	Логічне АБО
not()	Заперечення
=	Рівність
!=	Не рівність
< *	Менше
<= *	Менше чи дорівнює
> *	Більше
>= *	Більше чи дорівнює
|	Об’єднання двох множин вузлів

Приклади XPath виразів

./author **еквівалентно author**	Всі елементи <author> поточного контексту
/libraryDatabase	Елемент документа <libraryDatabase>
//author	Всі елементи <author> даного документу
//author [../@BK_ID=@AU_ID]	Всі елементи <author>, для яких значення AU_ID дорівнює значенню BK_ID батьківського елемента
//book/author	Всі елементи <author>, дочірні для <book>

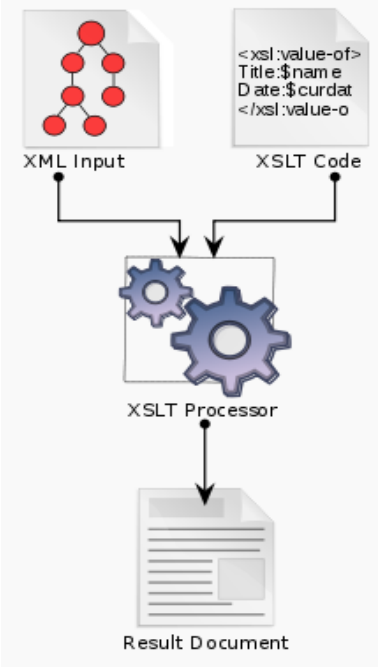
/libraryDatabase//author	Всі елементи <author>, дочірні на першому чи більш глибоких рівнях для <libraryDatabase>
/libraryDatabase/*/author	Всі елементи <author>, які є онуками для <libraryDatabase>
//book/*	Всі елементи, дочірні для <book>
//*[@AU_ID]	Всі елементи з атрибутом AU_ID
//book/@BK_ID	Атрибут BK_ID елементів <book>
//book/@BK_ID/author	Порожній набір вузлів, адже атрибути не містять елементів.
@*	Всі атрибути контекста поточного елемента
//author[2]	Другий елемент <author> в поточному контексті
//author[last()]	Останній елемент <author> в поточному контексті
//book/author[last()]	Останній дочірній елемент <author>, дочірній по відношенню до <book>
//book[author]	Всі елементи <book>, які містять хоча б один дочірній елемент <author>
//book[position() <=3]	Перші три книги
//book[not(author)]	Книги, які не містять атрибутів <author>
//author[@AU_ID !=1]	Елементи <author>, у яких значення атрибуту AU_ID не 1
//author[@AU_ID !=1]| //author[@AU_ID =1]	Об'єднання множини елементів <author>, у яких значення атрибуту AU_ID не 1 з множиною елементів <author>, у яких значення атрибуту AU_ID = 1
//author[@AU_ID >1]	Елементи <author>, у яких значення атрибуту AU_ID > 1

Трансформація з використанням XSLT

Для трансформації XML документів існує два основних підходи. Перший підхід полягає в застосування мови ***XSLT (extensible Stylesheet Language Transformations)***. Другий підхід передбачає використання для трансформацій API-інтерфейса LINQ to XML за рахунок функціонального конструювання цільового документа XML і вбудовування запиту LINQ to XML в деякий документ XML.

Extensible Stylesheet Language Transformations, або **XSLT** — мова програмування, яка використовується для програмування обробки XML документів. При цьому вихідний документ не змінюється, натомість, на основі результатів переробки створюється новий. Новий документ може бути серіалізовано (виведено) обробником в стандартний синтаксис XML, або інший формат, такий як HTML або простий текст. Найчастіше, XSLT використовується для перетворення структурованих XML документів із однієї XML схеми в іншу, або для перетворення у веб сторінки або PDF документи.

XSLT з'явився як результат розвитку технології Extensible Stylesheet Language (XSL) в W3C протягом 1998-1999 років. Також було створено XSL Formatting Objects (XSL-FO) та XML Path Language (XPath). Головним редактором першої версії (та, як наслідок, головним розробником мови програмування) був Джеймс Кларк. Найпоширенішою сьогодні версією є XSLT 1.0, яку було опубліковано як *Recommendation* (рекомендацію) W3C 16 листопада 1999 року. Значно розширена і доповнена версія 2.0, за редакцією Міхаеля Кея, отримала статус *Candidate Recommendation* від W3C 3 листопада 2005 року.



Мова перетворень XSLT. Заголовок схеми XSL трансформації

Визначення заголовка схеми трансформації XSL є кореневим елементом відповідного XML документа. Файлам із схемами трансформації найчастіше надають розширения .xsl. Крім визначення простору імен xsl, заголовок схеми трансформації може містити параметри виведення (output) результата трансформації: xml, html чи text. Якщо елемент output не зазначено явно, то його параметри визначаються інтерпретатором в процесі розбору схеми трансформації.

В XML документі стиль представлено елементом xsl:stylesheet. В якості синоніма xsl:stylesheet можна використовувати xsl:transform.

Заголовок схеми XSL трансформації

Обов'язково в заголовку повинен бути визначений елемент xsl:stylesheet, який повинен обов'язково мати атрибут version, що зазначає версія XSLT необхідну для цього стиля. Будемо використовувати версію XSLT 1.0.

<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

```
<xsl:output method='html'/>
...
</xsl:stylesheet>
```

Шаблон XSL

Шаблон XSL використовується для застосування низки перетворень до певного фрагменту XML документа. Група перетворень розміщується всередині визначення шаблону, а цільовий фрагмент документу відбирається за допомогою XPath запиту, який зазначено в атрибуті match.

```
<xsl:template match="book">
  <TR>
    <TD>
      <b>
        <xsl:value-of select="@BK_NAME"/>
      </b>
    </TD>
    <TD><xsl:value-of select="@BK_INFO"/></TD>
    <TD><xsl:value-of select="@DC_NAME"/></TD>
    <TD>
      <xsl:for-each select="author[@AU_ID >1]">
        <p>
          <xsl:value-of select="@AU_NAME"/>
        </p>
      </xsl:for-each>
    </TD>
  </TR>
</xsl:template>
```

Умовний оператор в XSL

Для умови використовується елемент if.

```
<xsl:for-each select ="author">
      <xsl:sort select="@AU_NAME" />
      <xsl:if test="@AU_ID > 1">
        <p>
          <xsl:value-of select="@AU_NAME"/>
        </p>
      </xsl:if>
  </xsl:for-each>
```

Цикл в XSL

Для перебору використовується елемент for-each. Які елементи перебирати визначає атрибут select елемента for-each. Значення цього атрибуту є XPath вираз.

```
<xsl:for-each select="author">
```

```
  <p>
    <xsl:value-of select="@AU_NAME"/>
  </p>
</xsl:for-each>
```

Сортування в XSL

Сортування доступне лише в контексті перебору елементів, організованого циклом for-each. За якими даними сортувати визначає атрибут select елемента sort. Значення цього атрибуту є XPath вираз.

```
<xsl:for-each select ="author">
    <xsl:sort select="@AU_NAME" />
    <p>
      <xsl:value-of select="@AU_NAME"/>
    </p>
</xsl:for-each>
```

Трансформації з використанням XSLT

Для виконання трансформації з використанням XSLT слід використовувати простір імен **System.Xml.Xsl**. Цей простір імен містить, зокрема, наступні класи:

XslCompiledTransform	Перетворення xml-даних за допомогою таблиці стилів XSLT
XsltArgumentList	Аргументи параметри XSLT чи об'єкти розширення
XsltCompileException	Виключення при знаходженні помилок в таблиці стилів XSLT при виконанні завантаження
XsltException	Виключення, яке формується в процесі обробки перетворення.
XsltSettings	Визначає функції XSLT для підтримки під час виконання таблиці стилів XSLT

Клас XslCompiledTransform

Клас XslCompiledTransform відповідає за перетворення xml-даних за допомогою таблиці стилів XSLT. Метод Load() завантажує таблицю стилів, після чого метод Transform() виконує xslt-перетворення.

В наступному прикладі виконується перетворення і запис результатів в файл:

Приклад. Трансформація XML в HTML

```
static void Transform()
        {
            // Завантаження стилів
            XslCompiledTransform xslt = new
XslCompiledTransform();
            string f1 = getFilePath("output1.xsl");
            xslt.Load(f1);
            // Виконання перетворення і виведення результатів
у файл.
            string f2 = getFilePath("books.xml");
            string f3 = getFilePath("books.html");
            xslt.Transform(f2, f3);
        }
        //Шлях на робочий стіл
        private static string getFilePath(string fileName)
        {
            return Path.Combine(Environment.GetFolderPath(
            Environment.SpecialFolder.Desktop), fileName);
        }
```

Запропонований вище приклад використовує два вхідних файли, розміщені на робочому столі:

books.xml

та

output1.xsl

books.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This file represents a fragment of a library database -->
<libraryDatabase>
  <book BK_ID="1" BK_NAME="Програмування"
BK_INFO="Інформація про Програмування" BK_DC="1"
DC_NAME="ПРОГРАМУВАННЯ">
    <author AU_ID="1" AU_NAME="Зубенко В.В." />
    <author AU_ID="2" AU_NAME="Омельчук Л.Л." />
  </book>
  <book BK_ID="2" BK_NAME="Математична логіка"
BK_INFO="Інформація про МЛ" BK_DC="2" DC_NAME="Логіка">
    <author AU_ID="4" AU_NAME="Нікітченко М.С." />
    <author AU_ID="4" AU_NAME="Шкільняк С.С." />
  </book>
  <book BK_ID="2" BK_NAME="Математична логіка"
BK_INFO="Інформація про МЛ" BK_DC="2" DC_NAME="Логіка">
    <author AU_ID="4" AU_NAME="Нікітченко М.С." />
```

```
        <author AU_ID="4" AU_NAME="Шкільняк С.С." />
      </book>
      <book BK_ID="3" BK_NAME="Програмна інженерія"
BK_INFO="Інформація про Програмну інженерію" BK_DC="1"
DC_NAME="ПРОГРАМУВАННЯ">
        <author AU_ID="5" AU_NAME="Лавріщева К.М." />
      </book>
    </libraryDatabase>
```

output.xsl

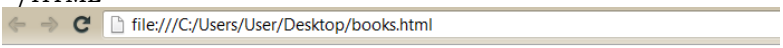
```
    <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
    <xsl:template match="libraryDatabase">
      <HTML>
        <BODY>
          <p>
            <H2>Список книжок</H2> </p>
        </BODY>
        <BODY>
          <TABLE BORDER="2">
            <TR>
              <TD>
                <b>Назва</b>
              </TD>
              <TD>
                <b>Інформація</b>
              </TD>
             <TD>
                <b>Категорія</b>
              </TD>
              <TD>
                <b>Автори</b>
              </TD>
            </TR>
            <xsl:apply-templates select="book"/>
          </TABLE>
        </BODY>
      </HTML>
    </xsl:template>
    <xsl:template match="book">
      <TR>
        <TD>
          <b>
            <xsl:value-of select="@BK_NAME"/>
```

```
    </b>
   </TD>
   <TD><xsl:value-of select="@BK_INFO"/></TD>
   <TD><xsl:value-of select="@DC_NAME"/></TD>
   <TD>
    <xsl:for-each select="author">
     <p>
      <xsl:value-of select="@AU_NAME"/>
     </p>
    </xsl:for-each>
   </TD>
  </TR>
</xsl:template>
</xsl:stylesheet>
```

Дане перетворення поверне нам наступний HTML:

```
<HTML>
 <BODY>
  <p>
   <H2>Список книжок</H2>
  </p>
 </BODY>
 <BODY>
  <TABLE BORDER="2">
   <TR>
    <TD><b>Назва</b></TD>
    <TD><b>Інформація</b></TD>
    <TD><b>Категорія</b></TD>
    <TD><b>Автори</b></TD>
   </TR>
   <TR>
    <TD><b>Програмування</b></TD>
    <TD>Інформація про Програмування</TD>
    <TD>ПРОГРАМУВАННЯ</TD>
    <TD>
     <p>Зубенко В.В.</p>
     <p>Омельчук Л.Л.</p>
    </TD>
   </TR>
   <TR>
    <TD><b>Математична логіка</b></TD>
    <TD>Інформація про МЛ</TD>
    <TD>Логіка</TD>
    <TD>
     <p>Нікітченко М.С.</p>
     <p>Шкільняк С.С.</p>
```

```
      </TD>
    </TR>
    <TR>
      <TD><b>Програмна інженерія</b></TD>
      <TD>Інформація про Програмну інженерію</TD>
      <TD>ПРОГРАМУВАННЯ</TD>
      <TD>
        <p>Лавріщева К.М.</p>
      </TD>
    </TR>
  </TABLE>
 </BODY>
</HTML>
```



Список книжок

Назва	Інформація	Категорія	Автори
Програмування	Інформація про Програмування	ПРОГРАМУВАННЯ	Зубенко В.В. Омельчук Л.Л.
Математична логіка	Інформація про МЛ	Логіка	Нікітченко М.С. Шкільняк С.С.
Програмна інженерія	Інформація про Програмну інженерію	ПРОГРАМУВАННЯ	Лавріщева К.М.

Приклад. Трансформація XML в XML

```
static void Transform()
        {
            // Завантаження стилів
            XslCompiledTransform xslt = new
XslCompiledTransform();
            string f1 = getFilePath("xml_xml.xsl");
            xslt.Load(f1);
// Виконання перетворення і виведення результатів у файл.
            string f2 = getFilePath("books.xml");
            string f3 = getFilePath("books11.xml");
             xslt.Transform(f2, f3);
        }
        static void Main(string[] args)
        {
            Transform();
            Console.ReadKey();
        }
```

Запропонований вище приклад використовує два вхідних файли, що розміщені на робочому столі:

xml_xml.xsl

та

books.xml

xml_xml.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="libraryDatabase">
      <transform>
        <xsl:apply-templates/>
      </transform>
     <xsl:text>&#13;&#10;&#x9;&#x9;</xsl:text>
    </xsl:template>
  <xsl:template match="book">
   <book>
    <BK_ID>
     <xsl:value-of select="@BK_ID"/>
    </BK_ID>
    <xsl:text>&#13;&#10;&#x9;&#x9;</xsl:text>
    <BK_NAME>
     <xsl:value-of select="@BK_NAME"/>
    </BK_NAME>
    <xsl:text>&#13;&#10;&#x9;&#x9;</xsl:text>
```

xml_xml.xsl

```
    <BK_INFO>
      <xsl:value-of select="@BK_INFO"/>
    </BK_INFO>
    <xsl:text>&#13;&#10;&#x9;&#x9;</xsl:text>
    <BK_DC>
     <xsl:value-of select="@BK_DC"/>
    </BK_DC>
    <xsl:text>&#13;&#10;&#x9;&#x9;</xsl:text>
    <DC_NAME>
     <xsl:value-of select="@DC_NAME"/>
    </DC_NAME>
    <xsl:text>&#13;&#10;&#x9;&#x9;</xsl:text>
    <authors>
     <xsl:for-each select ="author">
```

```
            <xsl:apply-templates select="@AU_NAME"/>
            <xsl:if test="position() != last()">
            <xsl:text>, </xsl:text>
            </xsl:if>
          </xsl:for-each>
        </authors>
        <xsl:text>&#13;&#10;&#x9;</xsl:text>
      </book>
    </xsl:template>
    </xsl:stylesheet>
```

Дане перетворення поверне нам наступний XML:

```
    <?xml version="1.0" encoding="utf-8"?>
    <transform>
      <book><BK_ID>1</BK_ID>
    <BK_NAME>Програмування</BK_NAME>
    <BK_INFO>Інформація про Програмування</BK_INFO>
    <BK_DC>1</BK_DC>
    <DC_NAME>ПРОГРАМУВАННЯ</DC_NAME>
    <authors>Зубенко В.В., Омельчук Л.Л.</authors>
    </book>
      <book><BK_ID>2</BK_ID>
    <BK_NAME>Математична логіка</BK_NAME>
    <BK_INFO>Інформація про МЛ</BK_INFO>
    <BK_DC>2</BK_DC>
    <DC_NAME>Логіка</DC_NAME>
    <authors>Нікітченко М.С., Шкільняк С.С.</authors>
    </book>
    <book><BK_ID>3</BK_ID>
    <BK_NAME>Програмна інженерія</BK_NAME>
    <BK_INFO>Інформація про Програмну
інженерію</BK_INFO>
    <BK_DC>1</BK_DC>
    <DC_NAME>ПРОГРАМУВАННЯ</DC_NAME>
    <authors>Лавріщева К.М.</authors>
    </book>
      <book><BK_ID>4</BK_ID>
    <BK_NAME>Програмна інженерія</BK_NAME>
    <BK_INFO>Інформація</BK_INFO>
    <BK_DC>1</BK_DC>
    <DC_NAME>ПРОГРАМУВАННЯ</DC_NAME>
    <authors />
    </book>
    </transform>
```

Завдання для опрацювання матеріалу:

Завдання 1. Опрацюйте усі приклади з лекції.

Завдання 2. В наступному прикладі, використовуючи LINQ to Object знайти всі елементи більші за 5:

```
public void Linq2()
    {
        int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

        var lowNums = // написати Linq запит

        Console.WriteLine("Numbers < 5:");
        foreach (var x in lowNums)
        {
            Console.WriteLine(x);
        }
    }
```

Завдання 3. Створення послідовності усі елементи якої більші на 1, ніж значення у вхідному масиві:

```
public void Linq3()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var numsPlusOne = //Linq

    Console.WriteLine("Numbers + 1:");
    foreach (var i in numsPlusOne)
    {
        Console.WriteLine(i);
    }
}
```

Ви повинні отримати наступний результат:

```
Numbers + 1:
6
5
2
4
10
9
7
8
```

```
3
1
```

Завдання 4. З використанням XMLDocument створіть XML-документ наступного вигляду:

```
<?xml version="1.0" encoding="utf-8"?>
<MyRoot>
  <MyChild ID="1">
    <MyGrandChild NAME="1 1" />
    <MyGrandChild NAME="1 2" />
  </MyChild>
  <MyChild ID="2">
    <MyGrandChild NAME="2 1" />
    <MyGrandChild NAME="2 2" />
  </MyChild>
  <MyChild ID="3">
    <MyGrandChild NAME="3 1" />
    <MyGrandChild NAME="3 2" />
  </MyChild>
</MyRoot>
```

1) Реалізуйте метод виведення на екран значення елемента.

2) Проаналізуйте створений документ за допомогою DOM та виведіть його на екран з використанням реалізованого вище методу.

3) Реалізуйте метод пошуку елементу з ID, значення якого введено з клавіатури.

Завдання 5. Виконайте попереднє завдання з використанням SAX.

Завдання 6. Виконайте попереднє завдання з використанням LINQ to XML.

Завдання 7. Трансформувати створений у завданні 4 файл до формату HTML, де у табличному вигляді представте інформацію.

Запитання до теми:

1. Основні компоненти XML-документа.
2. Моделі DOM та SAX для обробки XML.
3. Застосування XmlDocument. Застосування XmlReader.
4. LINQ to XML.
5. Порівняння LINQ to XML та DOM.
6. Мова запитів XPath.
7. Трансформація XML.
8. Трансформація XML з використанням XSLT.

9. Шаблон XSL.

Завдання для самостійної роботи.

Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторного заняття. Підготовка до **контрольної** роботи. Використати XML для свого варіанту лабораторної роботи №2 .

- ознайомитися з умовою;
- створити новий проєкт за шаблоном .NET MAUI (або іншим за власним вибором).
- використовуючи матеріали теоретичної частини лабораторних занять написати фрагменти лабораторної роботи, що відповідають за розбір XML та трансформацію XML в HTML.

ТИЖДЕНЬ 9-10

Тема 9-10. ШАБЛОНИ ПРОЄКТУВАННЯ. ПРИЗНАЧЕННЯ. КЛАСИФІКАЦІЯ. ШАБЛОНИ СТВОРЕННЯ, СТРУКТУРНІ ТА ШАБЛОНИ ПОВЕДІНКИ

Лекційні заняття 9-10

Огляд, мета і призначення теми: ознайомлення з шаблонами в розробці ПС. Породжуючі патерни (шаблони) проєктування.

Вивчивши матеріал даної теми, студент зможе: охарактеризувати основні шаблони проєктування.

Лабораторний практикум 9

Опанування шаблонів проєктування, студентські доповіді (45 хв.). Захист лабораторної роботи 1* та домашнього завдання (45 хв.)..

Лабораторний практикум 10

Опанування шаблонів проєктування (45 хв.). Захист UML до ЛР 2. Захист лабораторних робіт та домашнього завдання (45 хв.).

Завдання для опрацювання матеріалу

Завдання 1. В прикладі до шаблону «Абстрактна фабрика» додати конкретну фабрику `Mersedes`.

Текст програми прикладу до шаблону «Абстрактна фабрика»:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AbstractFactory
{
   public class AbstractFactory
   {
      // AbstractProductA
      abstract class Car
      {
         public abstract void Info();
      }

      // ConcreteProductA1
```

```
class Ford : Car
{
    public override void Info()
    {
        Console.WriteLine("Ford");
    }
}

//ConcreteProductA2
class Toyota : Car
{
    public override void Info()
    {
        Console.WriteLine("Toyota");
    }
}

// AbstractProductB
abstract class Engine
{
    public virtual void GetPower()
    {
    }
}

// ConcreteProductB1
class FordEngine : Engine
{
    public override void GetPower()
    {
        Console.WriteLine("Ford Engine 4.4");
    }
}

//ConcreteProductB2
class ToyotaEngine : Engine
{
    public override void GetPower()
    {
        Console.WriteLine("Toyota Engine 3.2");
    }
}

// AbstractFactory
```

```
interface ICarFactory
{
    Car CreateCar();
    Engine CreateEngine();
}

// ConcreteFactory1
class FordFactory : ICarFactory
{
    // from CarFactory
    Car ICarFactory.CreateCar()
    {
        return new Ford();
    }

    Engine ICarFactory.CreateEngine()
    {
        return new FordEngine();
    }
}

// ConcreteFactory2
class ToyotaFactory : ICarFactory
{
    // from CarFactory
    Car ICarFactory.CreateCar()
    {
        return new Toyota();
    }

    Engine ICarFactory.CreateEngine()
    {
        return new ToyotaEngine();
    }
}

static void Main(string[] args)
{
    ICarFactory carFactory = new ToyotaFactory();

    Car myCar = carFactory.CreateCar();
    myCar.Info();
    Engine myEngine = carFactory.CreateEngine();
    myEngine.GetPower();
```

```
            carFactory = new FordFactory();
            myCar = carFactory.CreateCar();
            myCar.Info();
            myEngine = carFactory.CreateEngine();
            myEngine.GetPower();

            Console.ReadKey();
        }

    }
}
```

Завдання 2. В прикладі до шаблону «Builder» додати клас MargaritaPizzaBuilder.

Текст програми прикладу до шаблону «Будівельник»:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Builder
{
    class Program
    {
        class Pizza
        {
            string dough;
            string sauce;
            string topping;
            public Pizza() { }
            public void SetDough(string d) { dough = d; }
            public void SetSauce(string s) { sauce = s; }
            public void SetTopping(string t) { topping = t; }
            public void Info()
            {
                Console.WriteLine("Dough: {0}\nSause: {1}\nTopping: {2}",
dough, sauce, topping);
            }
        }

        //Abstract Builder
```

```
    abstract class PizzaBuilder
    {
        protected Pizza pizza;
        public PizzaBuilder() {}
        public Pizza GetPizza() { return pizza; }
        public void CreateNewPizza() { pizza = new Pizza(); }

        public abstract void BuildDough();
        public abstract void BuildSauce();
        public abstract void BuildTopping();
    }
    //Concrete Builder
    class HawaiianPizzaBuilder : PizzaBuilder
    {
        public override void BuildDough() { pizza.SetDough("cross"); }
        public override void BuildSauce() { pizza.SetSauce("mild"); }
        public override void BuildTopping() {
pizza.SetTopping("ham+pineapple"); }
    }
    //Concrete Builder
    class SpicyPizzaBuilder : PizzaBuilder
    {
        public override void BuildDough() { pizza.SetDough("pan
baked"); }
        public override void BuildSauce() { pizza.SetSauce("hot"); }
        public override void BuildTopping() {
pizza.SetTopping("pepparoni+salami"); }
    }
    /** "Director" */
    class Waiter
    {
        private PizzaBuilder pizzaBuilder;
        public void SetPizzaBuilder(PizzaBuilder pb) { pizzaBuilder =
pb; }
        public Pizza GetPizza() { return pizzaBuilder.GetPizza(); }
        public void ConstructPizza()
        {
            pizzaBuilder.CreateNewPizza();
            pizzaBuilder.BuildDough();
            pizzaBuilder.BuildSauce();
            pizzaBuilder.BuildTopping();
        }
    }
    /** A customer ordering a pizza. */
```

```
    class BuilderExample
    {
        public static void Main(String[] args)
        {
            Waiter waiter = new Waiter();
            PizzaBuilder hawaiianPizzaBuilder = new
HawaiianPizzaBuilder();
            PizzaBuilder spicyPizzaBuilder = new SpicyPizzaBuilder();

            waiter.SetPizzaBuilder(hawaiianPizzaBuilder);
            waiter.ConstructPizza();

            Pizza pizza = waiter.GetPizza();
            pizza.Info();

            Console.ReadKey();
        }
    }

  }
}
```

Завдання 3. В прикладі до шаблону «Mediator» додати клас ConcreteCollegue3 та додати повідомлення від нього «Hello!».

Текст програми прикладу до шаблону «Посередник»:

```
using System;
using System.Collections;

namespace Mediator.Examples
{
  // Mainapp test application
  class MainApp
  {
    static void Main()
    {
      ConcreteMediator m = new ConcreteMediator();
      ConcreteColleague1 c1 = new ConcreteColleague1(m);
      ConcreteColleague2 c2 = new ConcreteColleague2(m);

      m.Colleague1 = c1;
      m.Colleague2 = c2;

      m.Send("How are you?", c1);
```

```
        m.Send("Fine, thanks", c2);

        // Wait for user
        Console.Read();
    }
}
// "Mediator"
abstract class Mediator
{
    public abstract void Send(string message,
      Colleague colleague);
}
// "ConcreteMediator"
class ConcreteMediator : Mediator
{
    private ConcreteColleague1 colleague1;
    private ConcreteColleague2 colleague2;

    public ConcreteColleague1 Colleague1
    {
        set { colleague1 = value; }
    }

    public ConcreteColleague2 Colleague2
    {
        set { colleague2 = value; }
    }
    public override void Send(string message,
      Colleague colleague)
    {
        if (colleague == colleague1)
        {
            colleague2.Notify(message);
        }
        else
        {
            colleague1.Notify(message);
        }
    }
}

// "Colleague"
abstract class Colleague
{
```

```
    protected Mediator mediator;

    // Constructor
    public Colleague(Mediator mediator)
    {
        this.mediator = mediator;
    }
}

// "ConcreteColleague1"
class ConcreteColleague1 : Colleague
{
    // Constructor
    public ConcreteColleague1(Mediator mediator)
        : base(mediator)
    {
    }

    public void Send(string message)
    {
        mediator.Send(message, this);
    }

    public void Notify(string message)
    {
        Console.WriteLine("Colleague1 gets message: "
          + message);
    }
}
// "ConcreteColleague2"
class ConcreteColleague2 : Colleague
{
    // Constructor
    public ConcreteColleague2(Mediator mediator)
        : base(mediator)
    {
    }

    public void Send(string message)
    {
        mediator.Send(message, this);
    }

    public void Notify(string message)
```

```
        {
            Console.WriteLine("Colleague2 gets message: "
              + message);
        }
    }
}
```

Завдання 4. Написати реалізацію для класу “Ялинка”, який можна за необхідності декорувати ялинковими прикрасами (поля) та/або гірляндами (метод, що відповідає за те, що “Ялинка” може світитися). Використати шаблон **Decorator**.

Завдання 5. Додати реалізацію класу “Трикутник” в прикладі до шаблону **Prototype**.

Текст програми прикладу до шаблону **Prototype**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PrototypeFigure
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.OutputEncoding = Encoding.UTF8;

            IFigure figure = new Rectangle(10, 20);
            IFigure clonedFigure = figure.Clone();
            figure.GetInfo();
            clonedFigure.GetInfo();
            figure = new Circle(15);
            clonedFigure = figure.Clone();
            figure.GetInfo();
            clonedFigure.GetInfo();

            Console.Read();
        }
    }

    interface IFigure
```

```
    {
        IFigure Clone();
        void GetInfo();
    }
    class Rectangle : IFigure
    {
        int width;
        int height;
        public Rectangle(int w, int h)
        {
            width = w;
            height = h;
        }
        public IFigure Clone()
        {
            return new Rectangle(this.width, this.height);
        }
        public void GetInfo()
        {
            Console.WriteLine("Прямокутник довжиною {0} и шириною
{1}", height, width);
        }
    }
    class Circle : IFigure
    {
        int radius;
        public Circle(int r)
        {
            radius = r;
        }
        public IFigure Clone()
        {
            return new Circle(this.radius);
        }
        public void GetInfo()
        {
            Console.WriteLine("Круг радіусом {0}", radius);
        }
    }
}
```

Завдання 6. За аналогією з завданнями 1-5 описати та виконати завдання для двох шаблонів із переліку: Factory Method, Facade, Composite, Adapter.

Завдання для самостійної роботи. Опрацювання лекційного матеріалу та власного проєкту лабораторного практикуму № 1, 2, 3. Підготувати доповідь про один із шаблонів:

- Заступник (Proxy).
- Міст (Bridge).
- Пристосуванець (Flyweight).
- Інтерпретатор (Interpreter).
- Шаблонний метод (Template Method).
- Ітератор (Iterator).
- Команда (Command).
- Спостерігач (Observer).
- Відвідувач (Visitor).
- Стан (State).
- Стратегія (Strategy).
- Зберігач (Memento).
- Ланцюжок обов'язків (Chain of Responsibility).

Запитання до теми:

1. Різновиди шаблонів проєктування.
2. Шаблон «Абстрактна фабрика».
3. Шаблон «Прототип».
4. Шаблон «Одиночний об'єкт».
5. Шаблон «Міст».
6. Шаблон «Ланцюг обов'язків».
7. Шаблон «Декоратор».
8. Шаблон «Ітератор».

Завдання для самостійної роботи

Опрацювання лекційного матеріалу та власного проєкту лабораторного практикуму № 2, 3. Виконання домашнього завдання №6. Аналіз задачі та розробка проєкту.

ТИЖДЕНЬ 11
Тема 11. ШАБЛОНИ ПРОЄКТУВАННЯ (продовження)

Лекційні заняття 11.

Огляд, мета і призначення теми: ознайомлення зі структурними шаблонама (патернами) проєктування.

Доповіді студентів про шаблони проєктування:

- Заступник (Proxy).
- Міст (Bridge).
- Пристосуванець (Flyweight).
- Інтерпретатор (Interpreter).
- Шаблонний метод (Template Method).
- Ітератор (Iterator).
- Команда (Command).
- Спостерігач (Observer).
- Відвідувач (Visitor).
- Стан (State).
- Стратегія (Strategy).
- Зберігач (Memento).
- Ланцюжок обов'язків (Chain of Responsibility).

Вивчивши матеріал даної теми, студент зможе: охарактеризувати основні шаблони.

Лабораторний практикум 11

Опанування шаблонів проєктування; прийом UML до ЛР 2* та 3, лабораторної роботи 2 та домашнього завдання (45 хв.).

Завдання для самостійної роботи

Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторного заняття. Виконання домашнього завдання №8. Проєктування та виконання лабораторної роботи №2* або 3.

ТИЖДЕНЬ 12
Тема 12. ШАБЛОНИ ПРОЄКТУВАННЯ (продовження). АНТИШАБЛОНИ

Лекційні заняття 12.

Огляд, мета і призначення теми: ознайомлення з Патерни поведінки та антишаблонами в розробці ПС, типові антишаблони, типові випадки некоректних підходів до розробки ПС.

Вивчивши матеріал даної теми, студент зможе: охарактеризувати основні антишаблони.

Лабораторний практикум 12.

Вивчення антишаблонів проєктування (30 хв.). Захист лабораторних робіт №3, 4 або 5 (60 хв.).

Запитання до теми:

1. Охарактеризуйте типові антишаблони.
2. Опишіть типові випадки некоректних підходів до розробки ПС.

Завдання для самостійної роботи

Опрацювання лекційного матеріалу та власного проєкту лабораторного практикуму № 2, 3. Аналіз задачі та розробка проєкту.

ТИЖДЕНЬ 13-14
Тема 13-14. ПРИНЦИПИ ПРОЄКТУВАННЯ «S.O.L.I.D.»

Лекційні заняття 13-14.

Огляд, мета і призначення теми: ознайомлення з принципом єдиності відповідальності (The Single Responsibility Principle), принципом відкритості/закритості (The Open Closed Principle), принципом заміщення Лісков (The Liskov Substitution Principle), принципом розділення інтерфейсу (The Interface Segregation Principle), принципом інверсії залежності (The Dependency Inversion Principle).

Вивчивши матеріал даної теми, студент зможе: охарактеризувати принципи проєктування S.O.L.I.D. та вміти їх застосовувати на практиці.

Лабораторний практикум 13-14

Опанування принципів проєктування (45 хв.). Здача лабораторних робіт 2* та 3, а також домашнього завдання (45 хв.)

Завдання для опрацювання матеріалу

Завдання 1. Нехай є клас для представлення замовлення в магазині. Який принцип порушено? Виправте.

```
class Item
{

}
class Order
{
    private List<Item> itemList;

    internal List<Item> ItemList
    {
        get
        {
            return itemList;
        }

        set
        {
            itemList = value;
        }
    }
```

```
    public void CalculateTotalSum() {/*...*/}
    public void GetItems() {/*...*/}
    public void GetItemCount() {/*...*/}
    public void AddItem(Item item) {/*...*/}
    public void DeleteItem(Item item) {/*...*/}

    public void PrintOrder() {/*...*/}
    public void ShowOrder() {/*...*/}

    public void Load() {/*...*/}
    public void Save() {/*...*/}
    public void Update() {/*...*/}
    public void Delete() {/*...*/}
}
```

Завдання 2. Який принцип порушено? Виправте.

Зверніть увагу на клас `EmailSender`. Крім того, що за допомогою методу `Send`, він відправляє повідомлення, він ще і вирішує як буде вестися лог. В даному прикладі лог ведеться через консоль.

Якщо трапиться так, що нам доведеться змінювати спосіб логування, то ми будемо змушені внести правки в клас `EmailSender`. Хоча, здавалося б, ці правки не стосуються відправки повідомлень. Очевидно, `EmailSender` виконує кілька обов'язків.

```
using System;

//Який принцип порушено? Виправте!
class Email
{
    public String Theme { get; set; }
    public String From { get; set; }
    public String To { get; set; }
}

class EmailSender
{
    public void Send(Email email)
    {
        // ... sending...
        Console.WriteLine("Email from '" + email.From + "' to
'" + email.To + "' was send");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Email e1 = new Email() { From = "Me", To = "Vasya",
Theme = "Who are you?" };
        Email e2 = new Email() { From = "Vasya", To = "Me",
Theme = "vacuum cleaners!" };
        Email e3 = new Email() { From = "Kolya", To =
"Vasya", Theme = "No! Thanks!" };
        Email e4 = new Email() { From = "Vasya", To = "Me",
Theme = "washing machines!" };
        Email e5 = new Email() { From = "Me", To = "Vasya",
Theme = "Yes" };
        Email e6 = new Email() { From = "Vasya", To =
"Petya", Theme = "+2" };

        EmailSender es = new EmailSender();
        es.Send(e1);
        es.Send(e2);
        es.Send(e3);
        es.Send(e4);
        es.Send(e5);
        es.Send(e6);

        Console.ReadKey();
    }
}
```

Завдання 3. Порушення якого принципу привело до невірного результату? Які шляхи вирішення?

```
using System;

class Rectangle
{
    public virtual int Width { get; set; }
    public virtual int Height { get; set; }
    public int GetRectangleArea()
    {
        return Width * Height;
    }
}
```

```
//квадрат наслідується від прямокутника!!!
class Square : Rectangle
{
    public override int Width
    {
        get { return base.Width; }
        set
        {
            base.Height = value;
            base.Width = value;
        }
    }
    public override int Height
    {
        get { return base.Height; }
        set
        {
            base.Height = value;
            base.Width = value;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Square();
            rect.Width = 5;
            rect.Height = 10;

            Console.WriteLine(rect.GetRectangleArea());
            Console.ReadKey();
        }
    }
}
```

Відповідь 100?

Завдання 4. Є код:

```
interface IItem
{
    void ApplyDiscount(String discount);
    void ApplyPromocode(String promocode);
```

```
    void SetColor(byte color);
    void SetSize(byte size);

    void SetPrice(double price);
}
```

Даний інтерфейс поганий тим, що він включає занадто багато методів. Що, якщо наш клас товарів не може мати знижок або промокодом, або для нього немає сенсу встановлювати матеріал з якого зроблений (наприклад, для книг). Таким чином, щоб не реалізовувати в кожному класі невикористовувані в ньому методи, краще розбити інтерфейс на кілька дрібних і кожним конкретним класом реалізовувати потрібні інтерфейси.

Перепишіть, розбивши інтерфейс на декілька інтерфейсів, керуючись принципом розділення інтерфейсу. Опишіть класи книжки (розмір та колір не потрібні, але потрібна ціна та знижки) та верхній одяг (колір, розмір, ціна знижка), які реалізують притаманні їм інтерфейси.

Запитання до теми:

1. Дайте визначення та охарактеризуйте принцип єдиної відповідальності (The Single Responsibility Principle).
2. Дайте визначення та охарактеризуйте принцип відкритості/закритості (The Open Closed Principle).
3. Дайте визначення та охарактеризуйте принцип заміщення Лісков (The Liskov Substitution Principle).
4. Дайте визначення та охарактеризуйте принцип розділення інтерфейсу (The Interface Segregation Principle).
5. Дайте визначення та охарактеризуйте принцип інверсії залежності (The Dependency Inversion Principle).

Завдання для самостійної роботи

Опрацювання лекційних матеріалів (прочитати, вивчити основні поняття) та матеріалів лабораторних занять. Виконання домашнього завдання №8. Підготовка до контрольної роботи.

ПЕРЕЛІК ВЖИВАНИХ СКОРОЧЕНЬ

CLR Common Language Runtime – вихідний файл виконання мов;

FCL Framework Class Library – стандартна бібліотека класів платформи.NET Framework;

JSON JavaScript Object Notation – Об'єктна нотація JavaScript.

HTML HyperText Markup Language – мова розмітки гіпертекстових документів;

IDE Integrated development environment – інтегроване середовище розробки;

LINQ Language Integrated Query – мова інтегрованих запитів;

MAUI Multi-platform App UI – мультиплатформний інтерфейс застосунків.

UML Unified Modeling Language – уніфікована мова моделювання;

XML Extensible Markup Language – розширювана мова розмітки;

БД база даних;

БНФ нотація Бекуса-Наура;

ЖЦ життєвий цикл;

ООП об’єктно-орієнтоване програмування;

ПС програмна система.

РЕКОМЕНДОВАНІ ДЖЕРЕЛА

1. Омельчук Л.Л. Об'єктно-орієнтоване програмування. Лабораторний практикум: навчальний посібник / Л.Л. Омельчук, А.С. Белова. – Київ. електронна публікація на сайті факультету, 2022 - 273 с.
2. Омельчук Л.Л. Об’єктно-орієнтоване програмування. Лабораторний практикум:навчальний посібник / Л.Л.Омельчук. – Київ: електронна публікація на сайті факультету, 2021. - 265 с. http://csc.knu.ua/uk/filer/canonical/1631712733/1373/
3. В.В. Зубенко, Л.Л. Омельчук. Програмування : навчальний посібник (гриф МОН України) / - К. : ВПЦ "Київський університет", 2011. - 623 с.
4. http://msdn.microsoft.com/ru-RU/
5. Роберт С. Мартін. Чиста архітектура. – Фабула. 2019 – 368 .с, ISBN 978-617-09-5286-8.
6. Роберт С. Мартін. Чистий код. Створення, аналіз і рефакторинг. – Фабула. 2019 – 416 с.., ISBN 978-617-09-5285-1.
7. Е. Робсон, Е. Фрімен, Head First. Патерни проєктування. – Фабула. 2020 – 672 с., ISBN 978-617-09-6159-4.
8. Jacobson, Ivar; Grady Booch; James Rumbaugh (1998). The Unified Software Development Process. Addison Wesley Longman. ISBN 0-201-57169-2.
9. Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides Released October 1994, Publisher(s): Addison-Wesley Professional.
10. Ставровский А.Б, Карнаух Т.О. Перші кроки програмування. - К.:Диалектика.-2005, с.389.

Навчальне видання

Омельчук Людмила Леонідівна
Свистунов Антон Олександрович

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ
Лабораторний практикум

Навчальний посібник

Підписано до друку 22.09.2023 р.
Формат 64×90/16. Папір офс. Друк офс. Ум. друк. арк 21
Гарнітура Times New Roman.

319