КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Б.І. Бойко
Л.Л. Омельчук
Н.Г. Русіна

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Лабораторний практикум

Навчальний посібник

Київ 2016

УДК 519.85 (075.8)

ББК 32.973.2-018я73

Б72

Рецензенти:

д-р фіз.-мат. наук, проф. *А.Ю. Дорошенко,*

д-р фіз.-мат. наук, проф. *С.С. Шкільняк*

Рекомендовано до друку вченою радою факультету

комп'ютерних наук та кібернетики (протокол № 4 від 12 грудня 2016 року)

Б72 Об'єктно-орієнтоване програмування. Лабораторний практикум: навчальний посібник / Б.І. Бойко, Л.Л. Омельчук, Н.Г. Русіна – К.: 2016. – 90 с.

УДК 519.85 (075.8)

ББК 32.973.2-018я73

Б72

Викладено матеріали до лабораторного практикуму з обов'язкової навчальної дисципліни «Об'єктно-орієнтоване програмування». В посібнику описано мету і призначення кожної теми дисципліни, наведено умови лабораторних робіт, порядок розрахунку семестрової оцінки, завдання для самостійної роботи та роботи на лабораторних заняттях, а також 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 та інформаційні технології».

© Бойко Б.І., Омельчук Л.Л., Русіна Н.Г., 2016

Лектор:

к.ф.-м.н., доцент кафедри «Теорії та технології програмування» факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, Омельчук Л.Л.

Викладачі лабораторних занять:

к.ф.-.м.н., доцент кафедри «Теорії та технології програмування» факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, Бойко Б.І.

к.ф.-м.н., доцент кафедри «Теорії та технології програмування» факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, Омельчук Л.Л.

асистент кафедри «Теорії та технології програмування» факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка, Русіна Н.Г.

Засоби навчання: комп'ютер з встановленою Visual Studio.

ЗМІСТ

ВСТУП 7
УМОВИ ЛАБОРАТОРНИХ РОБІТ 9
Лабораторна робота № 1 11
Лабораторна робота № 2 18
Лабораторна робота № 3 21
ВИМОГИ І РЕКОМЕНДАЦІЇ З НАПИСАННЯ КОДУ 26
Вибір імені 26
Форматування тексту програми 27
Основні домовленості та рекомендації з написання коду 29
Стилі використання регістрів 30
ЗМІСТ КУРСУ «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ» 33
ЗМІСТОВИЙ МОДУЛЬ 1. 33
ОБ'ЄКТНО-ОРІЄНТОВАНА МЕТОДОЛОГІЯ. ОСНОВНІ ПРИНЦИПИ МОДЕЛЮВАННЯ ПРОГРАМНИХ СИСТЕМ. ШАБЛОНИ ПРОЕКТУВАННЯ 33
ТЕМА 1. ОСНОВНІ ПОНЯТТЯ ООП. ПОРІВНЯННЯ ООП З ПАРАДИГМОЮ ПРОЦЕДУРНОГО ПРОГРАМУВАННЯ. ВТІЛЕННЯ ООП В МОВАХ ПРОГРАМУВАННЯ 33
Лабораторний практикум 1. Інтегроване середовище розробки MS Visual Studio 33
ТЕМА 2. ОСНОВНІ ЕТАПИ ЖИТТЄВОГО ЦИКЛУ (ЖЦ) ПС. ПРИЗНАЧЕННЯ ТА ВИКОРИСТАННЯ UML. ВИДИ ДІАГРАМ UML ТА ЇХ ВИКОРИСТАННЯ 45
Лабораторний практикум 2. Призначення та викоритання UML 46
ТЕМА 3. ШАБЛОНИ ПРОЕКТУВАННЯ. ШАБЛОНИ СТВОРЕННЯ, СТРУКТУРНІ ТА ШАБЛОНИ ПОВЕДІНКИ 47
Лабораторні практикуми 3-4. Шаблони проектування 47
ТЕМА 4. АНТИШАБЛОНИ: «ІНВЕРСІЯ АБСТРАКЦІЇ», «БОЖЕСТВЕННИЙ ОБ'ЄКТ», «МАГІЧНА КНОПКА», «МАГІЧНІ ЧИСЛА» 56
Лабораторний практикум 5. Рефакторинг 56
ТЕМА 5. ПРИНЦИПИ ПРОЕКТУВАННЯ «S.O.L.I.D.» 68

Лабораторний практикум 6. Принципи проектування класів (S.O.L.I.D.) 68
ЗМІСТОВИЙ МОДУЛЬ 2. 73
ОГЛЯД ТЕХНОЛОГІЇ .NET FRAMEWORK. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ C#. РОБОТА З XML-ДОКУМЕНТАМИ 73
ТЕМА 6. ОГЛЯД ТЕХНОЛОГІЇ .NET FRAMEWORK. СЕРЕДОВИЩЕ CLR. БІБЛІОТЕКА КЛАСІВ FRAMEWORK CLASS LIBRARY (FCL) 73
ТЕМА 7. ТИПИ І ЗМІННІ 73
Лабораторні практикуми 7-8. Типи і змінні. Типи значень. Порядковий тип, структури. Типи посилань. Клас як тип даних. Рядки. Наслідування класів. Наслідування як модель відношення “загальне-конкретне”. Втілення в мовах програмування. Перетворення типів. 73
ТЕМА 8. СТРУКТУРИ ДАНИХ ЇХ ПОРІВНЯЛЬНИЙ АНАЛІЗ. СПИСКИ, СТЕКИ, ЧЕРГИ. МАСИВИ ТА ВИДИ КОЛЕКЦІЙ В МОВІ C#. ІТЕРАТОРИ. КЛАСИ. КОНСТРУКТОРИ ТА ДЕСТРУКТОРИ. КОНСТАНТИ. МЕТОДИ. ВЛАСТИВІСТЬ. ІНДЕКСАТОРИ. ПОДІЇ. ДЕЛЕГАТИ 77
Лабораторний практикум 9. Структури даних їх порівняльний аналіз. Списки, стеки, черги. Масиви та види колекцій в мові C#. Ітератори. Класи. Конструктори та деструктори. Константи. Методи. Властивість. Індексатори. Події. Делегати. 77
ТЕМА 9. ПОШУК В НЕВПОРЯДКОВАНОМУ ТА ВПОРЯДКОВАНОМУ МАСИВАХ. ХЕШ-ТАБЛИЦІ. АЛГОРИТМИ СОРТУВАННЯ. ЇХ РЕАЛІЗАЦІЯ В C# 78
ТЕМА 10. ПЕРЕТВОРЕННЯ ТИПІВ. ОПЕРАТОРИ ТА ВИРАЗИ. ОПЕРАТОРИ ОБРОБКИ ВИКЛЮЧЕНЬ. НЕЯВНІ ТА АНОНІМНІ ТИПИ 78
Лабораторний практикум 10. Пошук в невпорядкованому та впорядкованому масивах. Хеш-таблиці. Алгоритми сортування. Їх реалізація в C#. Перетворення типів. Оператори та вирази. Оператори обробки виключень. Неявні та анонімні типи. 79
ТЕМА 11. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ C#. ЛЯМБДА-ВИРАЗИ. ЗАСТОСУВАННЯ ЛЯМБДА-ВИРАЗІВ 80
Лабораторний практикум 11. Функціональні можливості C#. Лямбда-вирази. Застосування лямбда-виразів. 80
ТЕМА 12. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ C#. LINQ: МОВА ІНТЕРПРЕТОВАНИХ ЗАПИТІВ. ОПЕРАЦІЇ ЗАПИТІВ LINQ. ВИКОРИСТАННЯ LINQ ДЛЯ ДОСТУПУ ДО СТРУКТУР ДАНИХ В ПАМ’ЯТІ (LINQ TO OBJECTS)) 81

Лабораторний практикум 12. Функціональні можливості C#. LINQ: мова інтерпретованих запитів. Операції запитів LINQ. Використання LINQ для доступу до структур даних в пам'яті (LINQ to Objects) 81
ТЕМА 13. РОБОТА З ФАЙЛАМИ ТА ДИРЕКТОРІЯМИ 82
Лабораторний практикум 13. Робота з файлами та директоріями 82
ТЕМА 14. ОСНОВНІ КОМПОНЕНТИ XML ДОКУМЕНТА. ЗАСТОСУВАННЯ XML ДОКУМЕНТА. МОДЕЛІ DOM ТА SAX ДЛЯ ОБРОБКИ XML. ЗАСТОСУВАННЯ LINQ TO XML 83
Лабораторний практикум 14. Основні компоненти XML документа. Застосування XML документа. Моделі DOM та SAX для обробки XML. Застосування LINQ to XML. 83
Лабораторний практикум 15. Мова запитів XPath. Трансформація XML. 84
КОНТРОЛЬ ЗНАНЬ І РОЗПОДІЛ БАЛІВ, ЯКІ ОТРИМУЮТЬ СТУДЕНТИ 86
ПЕРЕЛІК ВЖИВАНИХ СКОРОЧЕНЬ 88
РЕКОМЕНДОВАНА ЛІТЕРАТУРА 89

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 та інформаційні технології». Дисципліна «Об'єктно-орієнтоване програмування» викладається студентам освітньо-професійної програми «Інформатика» у 3 семестрі 2 курсу в обсязі – 120 год.

(4 кредити ECTS) зокрема: лекції – 30 год., лабораторні – 28 год., консультації – 2 год., самостійна робота – 60 год. У курсі передбачено 2 змістових модулі та 2 модульні контрольні роботи. Завершується дисципліна – екзаменом.

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

Завдання – набуття знань, умінь та навичок (компетенцій) на рівні новітніх досягнень у програмуванні, відповідно до кваліфікації фахівець з інформаційних технологій. Зокрема, розвивати:

- здатність проектувати та розробляти програмне забезпечення із об'єктно-орієнтованої парадигми програмування з відповідними моделями, методами та алгоритмами обчислень, структурами даних і механізмами управління проектувати;
- застосовувати знання у практичних ситуаціях;
- знання та розуміння предметної області та розуміння професійної діяльності;
- здатність розробляти архітектури, модулі та компоненти програмних систем;
- здатність приймати участь у проектуванні програмного забезпечення, включаючи проведення моделювання його структури, поведінки та процесів функціонування.

В результаті вивчення навчальної дисципліни студент повинен:

знати: засоби проектування систем; стандарти моделювання, методи аналізу потреб; методи розробки програмного забезпечення; принципи проектування користувацьких інтерфейсів; інструментальні засоби для формалізації функціональних специфікацій; мови програмування; основні процеси, фази та ітерації життєвого циклу програмного забезпечення; технології та методи

проектування та програмування; ефективні підходи щодо проектування програмного забезпечення.

вміти: розробляти шаблони і моделі проектування; збирати, формалізувати і оцінювати функціональні та інші вимоги; розробляти специфікації з урахуванням встановлених вимог; роз'яснювати і представляти проекти / розробки замовникам; використовувати моделі даних; розробляти ПЗ на основі моделі; розробляти ПЗ з використанням сучасних технологій розробки; складати план тестування і контролювати його виконання; демонструвати процеси та результати професійної діяльності, розроблюючи презентації, звіти; методи компонентної розробки програмного забезпечення, виділяючи інтерфейси і реалізації та взаємодію між модулями, підсистемами і компонентами.

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 інших курсів та спецкурсів з програмування, окремих розділів теорії алгоритмів та математичної логіки.

УМОВИ ЛАБОРАТОРНИХ РОБІТ

На першому лабораторному занятті з «Об'єктно-орієнтованого програмування» кожен студент визначається з формою подальшої роботи, а саме:

форма 1) виконання та своєчасна здача лабораторних робіт № 1, 2, 3 з можливістю отримати максимум по 10 балів за кожну з них;

форма 2) виконання завдань на лабораторних заняттях (протягом усього семестру) з можливістю отримати 10 балів та своєчасна здача лабораторних робіт №2, 3 з можливістю отримати максимум по 10 балів за кожну з них. Надалі в розрахунках отримані на лабораторних заняттях бали вважаються балами отриманими за виконання лабораторної роботи № 1.

Перелік завдань для лабораторних занять не є вичерпним та визначається викладачем індивідуально.

Обирається лише один з варіантів роботи. Вибір студента фіксується на першому занятті. Отримання одночасно по 10 балів за виконання лабораторної роботи № 1 та за виконання завдань на лабораторних заняттях не можливо!

Наведені нижче умови завдань не є вичерпними чи абсолютно точними. Автору розробки надається можливість розширювати поставлені завдання та виходити за їх рамки.

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

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

Звіт має включати постановку задачі, обґрунтування обраних методів розв'язання та інструкцію користувача. Структура звіту має відображати плани (в тому числі потижневі) та наміри автора розробки у першій частині звіту та отримані результати – у другій його частині.

Текст звіту може бути поданий викладачу практикуму у письмовій формі чи як електронний документ за дозволом викладача. Розвинена допомога не заміняє звіт. Датою прийняття лабораторної роботи лектором є дата, що буде зафіксована скринькою електронної пошти лектора за електронною адресою, що повідомляється на лекції. Формат імені архіву, що додаються до електронного листа moroz_25_1.zip (rar) означає, що автором є студент Мороз із групи К25 і в ньому всі матеріали для перевірки лабораторної робот № 1.

Перевірка виконаних робіт здійснюється за розкладом занять викладача. Разом з проектом має бути набір тестових файлів та інших матеріалів, що свідчать про правильність створених програм. Автор проекту повинен щотижня інформувати викладача про виконані етапи робіт. Звіт є обов'язковою складовою частиною роботи. Для одержання найвищої оцінки обов'язковим є використання C# (чи іншого середовища розробки), ООП та виконання індивідуального завдання, що узгоджується студентом та викладачем (наприклад, написати функцію для спеціалізованого редагування HTML-сторінки й інше).

Терміни здачі лабораторних робіт першого семестру:

- 1 лабораторна робота – до 6 тижня семестру;
- 2 лабораторна робота – до 10 тижня семестру;
- 3 лабораторна робота – до 14 тижня семестру.

У разі неякісного виконання лабораторної роботи, викладач має право не зарахувати лабораторну роботу, або знизити за неї бали.

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

Лабораторна робота № 1

РОЗРОБКА ПРОГРАМ ОБРОБКИ ФАЙЛІВ ТА РЕДАГУВАННЯ ТЕКСТІВ

Обов'язковою частиною лабораторної роботи є використання UML при проектуванні та специфікації програмних систем.

Примітка: можна використовувати будь-які CASE-системи, що підтримують UML.

В межах лабораторної роботи № 1 мають бути розроблені 8 UML-діаграм:

- діаграми варіантів використання (діаграми прецедентів) - 2..* (щонайменше до трьох прецедентів описати потоки подій);
- діаграми класів - 1..*;
- діаграми діяльності - 0..*;.
- діаграми взаємодії - 3..*;
- діаграми станів - 0..*;
- діаграми компонентів - 1..*;
- діаграми розгортання - 1..*.

Для завдань лабораторної роботи 1 розділу 2 (опанування шаблонів проектування) повинні бути створені діаграми класів та взаємодії.

Лабораторна робота повинна базуватися на розроблених діаграмах UML. Використання антишаблону магічна кнопка (англ. Magic pushbutton) штрафується 3 балами. Грамотне використання шаблонів проектування при розробці проекту додає 2 додаткові бали.

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

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 (у тому числі написаних на мовах програмування чи інших формалізованих мовах), друк документів, перетворення формату документів, що містять тексти, таблиці та інше.

Розробити проект спеціалізованої програми обробки файлів (файлового менеджера) та вбудованого редактора (для редагування текстових та інших файлів). Розроблений проект повинен містити головне меню та кнопки, що дублюють

основні його елементи. Повинна бути подана коротка інформація про програму та надаватися допомога користувачу. Діалог україномовний. Приклад головного вікна проекту наведено на рисунку 1.

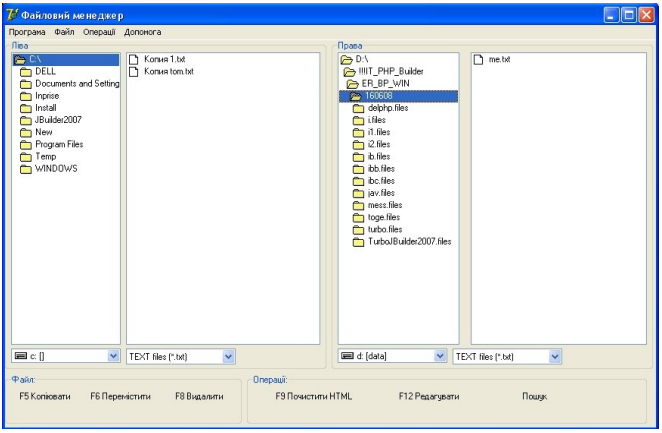

Рисунок 1. Приклад дизайну головного вікна проекту лабораторної роботи № 1

Розділ 0. Обробка файлів файловим менеджером

Файловий менеджер має виконувати основні операції з файлами. Він повинен мати дві панелі, а головне вікно займати не менше 80 відсотків площі екрана монітора. Приклад його головної форми подано на рис. 1. За погодженням з викладачем завдання може бути ускладненим (із нарахуванням додаткових балів). Вбудований текстовий редактор викликається натисканням F3 на імені текстового файлу, що відображено на активній панелі.

0.1. Обробка текстових файлів:

1. Пошук файлів, що мають особливості в їх іменах (напр. *.txt) та виділення слів, що належать першим трьом символам (крім 1- та 2-літерних).
2. Пошук файлів, що мають особливості у змісті (напр., мають якісь слова чи інші особливості) та виділення слів, що є скороченнями, складеними із комбінацій малих та великих літер.
3. Пошук файлів, що мають особливості у розташуванні, розмірі, даті та ін. (наприклад, усі вони є у якомусь каталозі) та виділення слів, що написані латиницею.

4. Пошук файлів формату HTML за ключовим словами (keywords) та виділення ключових слів.
5. Пошук файлів формату HTML за словами із заголовків розділів та виділення усіх слів із заголовків.
6. Пошук файлів формату HTML за кількома словами в основному тексті розділів та виділення найбільш вживаних слів.
7. Складання списку файлів, гіпертекстові посилання на які містяться у файлі формату HTML (інші файли формату HTML).
8. Складання списку файлів, гіпертекстові посилання на які містяться у файлі формату HTML (файли графічних форматів).
9. Копіювання та переміщення файлів формату HTML разом із пов'язаними файлами та виділення слів, що вказані в <TITLE>.
10. Пошук файлів формату HTML за кількома введеними користувачем словами (врахувати можливість розриву слова тегами HTML) та виділення найбільш вживаних слів.
11. Інші задачі обробки текстових файлів за пропозицією виконавця або викладача.

0.2. Обробка довільних файлів:

1. Поділ файлу на частини заданого розміру.
2. Злиття текстових та інших файлів.
3. Копіювання груп файлів, що мають особливості в іменах (наприклад *.txt) або у змісті.
4. Переміщення груп файлів , що мають особливості в іменах (наприклад *.txt) або у змісті.
5. Рознесення груп файлів у різні каталоги.
6. Очищення файлів формату HTML від зайвих тегів за пропозицією виконавця або викладача.
7. Спрощення тегів файлів формату HTML (напр., FONT color=#ff0000 замінити на FONT та ін.).
8. Злиття файлів HTML-формату із заміною стилю у другій частині на стиль, що прийнятий у першій.
9. Злиття файлів HTML-формату із переміщенням зображень другої частини до каталогу, що містить зображення першої частини.
10. Виділення із файлу HTML-формату частини тексту у новий файл із переміщенням зображень виділеної частини до нового каталогу.

Розділ 1. Редагування текстових та інших файлів

Текст, що складається з окремих слів, вводиться із клавіатури та зберігається у текстовому файлі. Алфавіт – довільний. Слова відокремлюються символами пропуску « » (щонайменше одним таким символом). Довжина окремого слова, рядка, тексту в цілому не обмежується. Передбачити використання меню, кнопок, допомоги, подачі інформації про програму та роботу у режимах обробки звичайних рядків та розширених рядків. Тестування може бути проведено на текстах, що записані у довільному текстовому файлі. Конкретний варіант завдання подається викладачем:

1. Складання списку тегів у файлі формату HTML та заміна кількох обраних на інші.
2. Для кожного зі слів тексту підрахувати, скільки разів воно зустрічається у тексті.
3. Знайти десять найдовших слів у тексті.
4. Знайти складені з латинських літер слова, які у тексті зустрічаються найчастіше.
5. Знайти слова, які зустрічаються у тексті лише по одному разу.
6. Заміна однієї послідовності літер іншою.
7. Заміна регістру у виділеному фрагменті тексту.
8. За заданими словами виділити в текстовому файлі найбільш схоже слово за першими кількома літерами. Передбачити повідомлення «неоднозначно», якщо є кілька схожих слів, та вивести їх. Список слів є у вхідному файлі test_lst.txt.
9. Копіювати текстовий файл у новий таким чином, щоб з кожної групи послідовно однакових рядків виводився тільки один рядок.
10. Видалити з кожного рядка файлу надлишкові пробіли та всі табуляції, а також видаляти рядки, що цілком складаються із пробілів і табуляцій.
11. Виконати контекстну заміну у всіх рядках файлу. Якщо рядок не задовольняє умові заміни – він залишається незмінним.
12. Виконати пошук літерних послідовностей в текстовому файлі. Слід виводити рядки (та номери рядків) файлу, у яких зустрілась задана літерна послідовність, що задається як аргумент.
13. Перетворити текст, що складається тільки з малих літер у текст, що складається з великих і малих літер. Перша літера і літера після кінця речення – великі, інші – малі.
14. Виправити помилки в словах: програмі задано список слів; вона перевіряє – чи є аналізоване слово є словом зі списку. Якщо ні – намагається знайти найбільш схоже слово зі списку, причому якщо є кілька схожих - видає усі варіанти. Розглянути випадки:

- дві сусідні літери переставлені місцями: патра => парта;
- подвоєна літера (літери): ппаарртаа => парта;
- загублена літера: прта => парта;
- змінена літера: шухта => шахта;
- зайва літера: партяа =>парта;
- літери не в тому регістрі – порівняти з кожним словом зі списку, перетворюючи всі літери у малі: парТа => парта .

Розділ 2. Робота з файловим менеджером. Опанування шаблонів проектування

Для кожного з наступних завдань створити діаграми класів та послідовності використання.

1. Застосування шаблону «Фабричний метод» (Factory Method) для збереження документа на диск в різних форматах: html, txt, бінарному. Створіть базовий клас, де буде код відкритого файлу і фабричний метод для створення конкретного об'єкту, який буде зберігати дані в цей файл.

2. Застосування шаблону «Абстрактна фабрика» (Abstract Factory) для відкриття файлів різних форматів. Завчасно не відомо, який файл буде відкриватися, але є список можливих форматів (xml, txt). Користувач зазначає файл, бібліотека пробує визначити тип файлу (за розширенням) та викликати відповідний завантажувач для нього.

3. Шаблон "Фасад". В проекті повинен бути реалізований переглядач файлів, який виконує основні функції з файлами (завантаження, створення, вилучення…) не розкриваючи клієнту деталі цих операцій.

4. Шаблон "Спостерігач" (Observer). В проекті повинна бути реалізована відправка повідомлень панелям файлового менеджера про редагування списку файлів у випадку додавання, видалення, переміщення, переіменування файлів чи папок (каталогів, директорій).

5. Використовуючи шаблон «Одинак» (Singleton), розробіть систему протоколювання подій файлового менеджера. Система повинна:

- підтримувати 3 рівні важливості подій (зміни, перехід, помилка) – забезпечити фіксацію події (з подією фіксуються час, важливість, текстове повідомлення).

Наприклад:

21.09.2017 13:02:22 Зміни. Створено файл c:\temp\aaa.txt
21.09.2017 14:02:22 Зміни. Переміщено файл з c:\temp\aaa.txt в c: \aaa.txt
21.09.2017 14:02:22. Перехід. Відкрито директорію c:\temp
21.09.2017 15:02:22 Помилка. Помилка доступу до файлу з c:\temp\ata.txt

6. З використанням шаблону «Будівельник» (Builder) реалізувати створення звітів про роботу файлового менеджера в одному з обраних при завантаженні програми форматів (html чи txt).

Повідомлення складається з таких складових: заголовок звіту, важливість (зміни, перехід, помилка), текст повідомлення, час, завершення створення повідомлення (по завершенню програми, чи по натисканню кнопки)). Результатом є збереження файлу відповідного формату.

Для txt-файлу просто запис тексту у наступному вигляді:

Звіт про роботу файлового менеджера. 21.09.2016 14:02:00

21.09.2017 14:02:22 Зміни. Переміщено файл з c:\temp\aaa.txt в c: \aaa.txt

21.09.2017 14:02:22. Перехід. Відкрито директорію c:\temp

Дякую! 21.09.2017 15:00:20

Для html файлу просто запис у наступному вигляді:

```
<HTML><BODY>
<H1> Звіт про роботу файлового менеджера. 21.09.2017 14:02:00</H1>
<p>21.09.2017 14:02:22 Зміни. Переміщено файл з c:\temp\aaa.txt в c: \aaa.txt</p>
<p>21.09.2017 14:02:22. Перехід. Відкрито директорію c:\temp</p>
<p><i> Дякую! 21.09.2017 15:00:20</i></p>
</BODY></HTML>
```

Розподіл варіантів лабораторної роботи № 1 наведено в таблиці 1.

Таблиця 1. Розподіл варіантів лабораторної роботи № 1

№ варіанту	Розділ 0.1	Розділ 0.2	Розділ 1 (2 завдання)		Розділ 2
1	1	5	1	8	2
2	2	6	2	9	3
3	3	7	3	10	4
4	4	8	4	11	5
5	5	9	5	12	6
6	6	10	6	13	6
7	7	1	7	14	1
8	8	2	1	8	2
9	9	3	2	9	3
10	10	4	3	10	4
11	1	5	4	11	5
12	2	6	5	12	6
13	3	7	6	13	5
14	4	8	7	14	1
15	5	9	1	9	2
16	6	10	2	10	3
17	7	1	3	11	4
18	8	2	4	12	5
19	9	3	5	13	6
20	10	4	6	14	4
21	1	5	7	14	2
22	2	6	1	13	3
23	3	7	2	12	4
24	4	8	3	11	5
25	5	9	4	10	6
26	6	10	5	9	2
27	7	1	6	8	1
28	8	2	7	8	2
29	9	3	8	9	3
30	10	4	9	10	4

Лабораторна робота № 2

РОЗШИРЕННЯ ФУНКЦІОНАЛЬНОСТІ СПЕЦІАЛІЗОВАНОГО ФАЙЛОВОГО МЕНЕДЖЕРА ФУНКЦІЯМИ ДЛЯ РОБОТИ З ЕЛЕКТРОННИМИ ТАБЛИЦЯМИ (АНАЛІЗ ТА ОБЧИСЛЕННЯ ВИРАЗІВ)

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

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

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.

Розширити функціональність файлового менеджера шляхом додавання форм для введення, обробки та збереження електронних таблиць.

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

Передбачити налаштування на два режими роботи:

- вузький набір операцій, коли можна використовувати виключно ті операції, що перераховані у виданому варіанті роботи. Використання інших має означати синтаксичну помилку.
- розширений набір операцій.

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.

Обов'язковим є включення до звіту формалізованого опису множини виразів за допомогою БНФ або діаграм Вірта.

Перелік варіантів операцій та функцій наведено в таблиці 2.

Таблиця 2. Перелік варіантів операцій до лабораторної роботи № 2

№ варіанту	Операції	№ варіанту	Операції
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) наведено в таблиці 3.

Таблиця 3. Розподіл варіантів лабораторної роботи № 2

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 № 3

РОЗШИРЕННЯ ФУНКЦІОНАЛЬНОСТІ СПЕЦІАЛІЗОВАНОГО ФАЙЛОВОГО МЕНЕДЖЕРА ФУНКЦІЯМИ ДЛЯ РОБОТИ ФАЙЛАМИ ФОРМАТУ XML

Реалізувати можливість запуску файлів правильної структури з файлового менеджера (лабораторна роботи № 1).

При реалізації лабораторної роботи обов'язковим є використання шаблонів проектування. Наприклад, використання шаблону «Стратегія» Strategy. Наявність діаграми класів обов'язкова!

Використання антишаблону магічна кнопка (англ. Magic pushbutton), відсутність діаграми класів чи невикористання шаблонів штрафується 3 балами.

Є інформаційна система (вважаємо, що вона вже існує і на даному етапі її розробляти непотрібно). Інформація з якої передається між клієнтами та системою у вигляді XML-документів. Необхідно забезпечити обробку цих документів. 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 інше.

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 інше.

4. «Електронний архів факультету»

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

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 інше.

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 інше.

8. «Розклад роботи лабораторій 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 правила використання, інформація при реєстрації користувачів.

11. «Електронна 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 інше.

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 інше.

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 інше.

18. «Кадри науковців (Публікації)»

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

19. «Кадри науковців (Заробітна плата)»

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 тез, вартість участі та інше.

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 інше. Використати допоміжні таблиці (класифікатори, довідники, словники та інше.)

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 інше.

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 інше 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.

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 їхній 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 програми

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 наіменуванні ідентифікаторів в C#.

Таблиця 4. Зведена таблиця використання регістрів та префіксів

Ідентифі-катор	**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	Pascal Casing	

тип		}

Делегат обробник події	Pascal Casing Закінчується суфіксом «EventHandler»	public delegate void AnswerCreatedEventHandler(object sender, AnswerCreatedEventArgs e);
		public class AnswerCreatedEventArgs: EventArgs
		{
		public int CreatedId;
		public int ParentId;
		public string CreatorName;
		}

Клас-нащадок від EventArgs	Pascal Casing Закінчується суфіксом «EventArgs»	

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;

ЗМІСТ КУРСУ «ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

ЗМІСТОВИЙ МОДУЛЬ 1.

ОБ'ЄКТНО-ОРІЄНТОВАНА МЕТОДОЛОГІЯ. ОСНОВНІ ПРИНЦИПИ МОДЕЛЮВАННЯ ПРОГРАМНИХ СИСТЕМ. ШАБЛОНИ ПРОЕКТУВАННЯ

ТЕМА 1. ОСНОВНІ ПОНЯТТЯ ООП. ПОРІВНЯННЯ ООП З ПАРАДИГМОЮ ПРОЦЕДУРНОГО ПРОГРАМУВАННЯ. ВТІЛЕННЯ ООП В МОВАХ ПРОГРАМУВАННЯ

Огляд, мета і призначення теми: огляд основних понять ООП, парадигми ООП, огляд класів і роботи з екземплярами класів, основні члени класу.

Вивчивши матеріал даної теми, студент зможе: розуміти та розрізняти основні парадигми ООП, розуміти поняття клас, об'єкт, екземпляр.

Лабораторний практикум 1. Інтегроване середовище розробки MS Visual Studio

При першому запуску Visual Studio, необхідно вибрати поєднання параметрів, яке застосовує набір визначених налаштувань до інтегрованого середовища розробки. Кожне поєднання параметрів призначене для спрощення розробки додатків (див. рис.1).

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 проект, на її місці будуть відображатися редактори і конструктори. При створенні програми ви будете проводити більшу частину часу в цій центральній області (див. рис. 2).

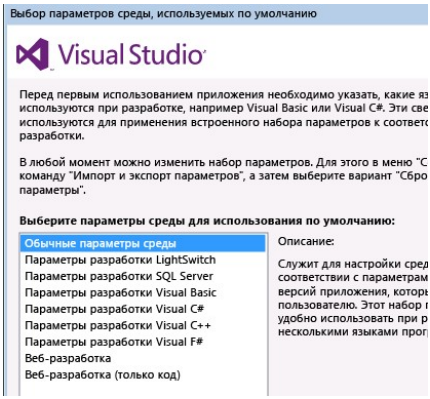

Рисунок 2. Налаштування IDE

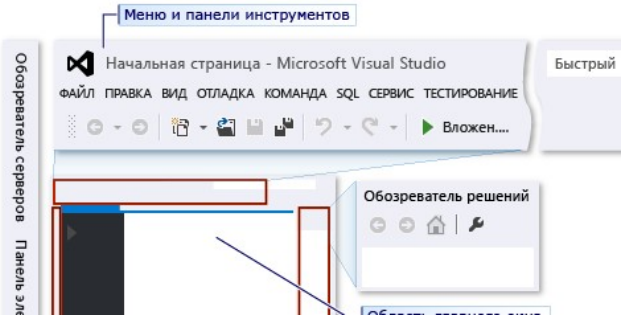

Рисунок 3. Меню, панелі інструментів, головне вікно Visual Studio

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" (див. рис.3).

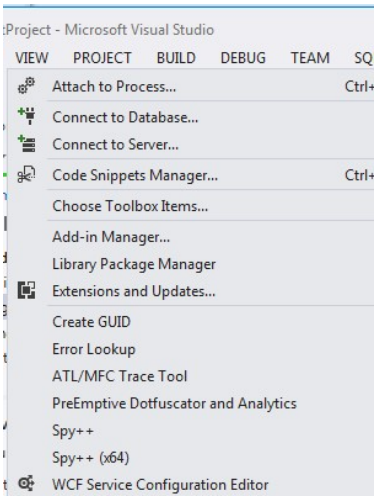

Рисунок 4. Налаштування в Visual Studio

При створенні програми в Visual Studio спочатку створюється проект і рішення (див. Рис. 4).

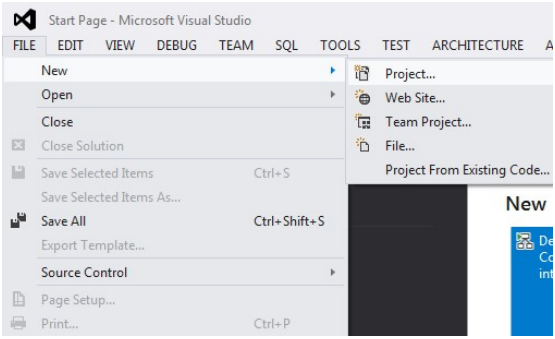

Рисунок 5. Створення нового проекту

Для запуску відлагодження проекту слід натиснути Debug-> Start Debugging (F5) (див. рис. 5).

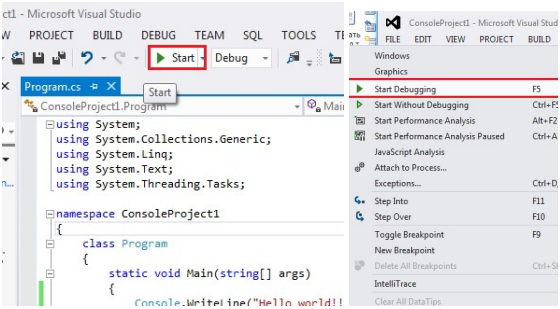

Рисунок 6. Варіанти запуску відлагодження

Нехай, при написанні програми ми допустилися синтаксичних помилок, тоді середовище розробки допоможе нам їх знайти й виправити (див. рис.7).

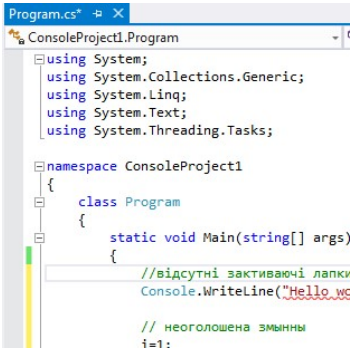

Рисунок 7. Синтаксична помилка

При виконання побудови (Build-> Build Solution (F6)), або при запуску (Debug-> Start Debugging F5) отримаємо помилки із зазначенням рядку (див. рис. 8). Подвійним натисканням миші на описі помилки переходимо в потрібний рядок.

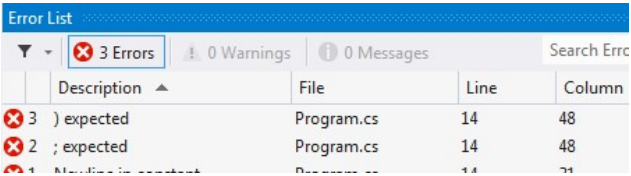

Рисунок 8. Список помилок

Виправимо першу помилку та знову перевіримо. При цьому отримаємо повідомлення про другу помилку, а саме про неоголошену змінну (див. рис. 9).

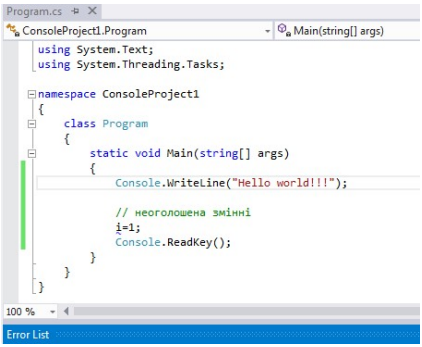

Рисунок 9. Синтаксичні помилки

Виправимо помилку. Запустимо, при цьому отримали попередження про невикористану змінну (див. рис. 10).

Якщо помилки компіляції нам допомагає знайти компілятор, то з помилками логіки справа йде трохи складніше.

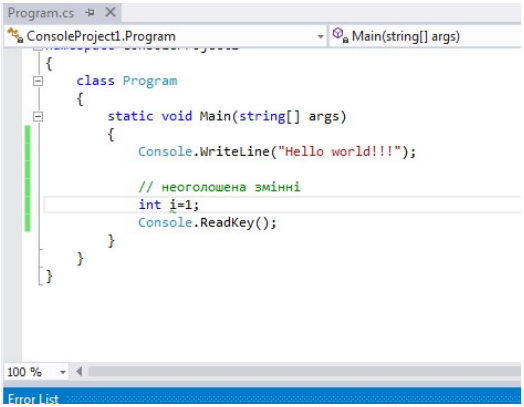

Рисунок 10. Список попереджень

Як дізнатися, чому якась змінна не змінюється або чому результат виконання якогось коду не такий, як ви очікували? Нехай маємо наступний код програми:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication2
{
    class Program
    {
        static int Double(int x)
        {
            return x * 2;
        }
        static void Main(string[] args)
        {
            int x = 10;
            int y = 15;
            x = Double(x);
            x = x / (y - 15);
            Console.WriteLine(x.ToString());
        }
    }
}
```

При побудові рішення (Build-> Build Solution (F6)) помилок не знайдено. Проте, при запуску на виконання (Debug-> Start Debugging F5) отримали помилку (див. рис.11).

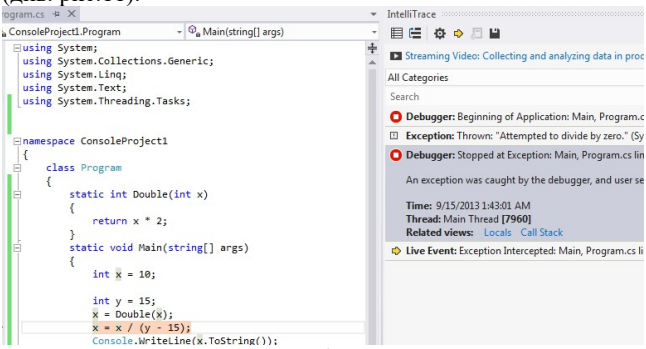

Рисунок 11. Виняткова ситуація

Сталася виняткова ситуація, 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 (див. Рис. 12). Трохи нижче під заголовком знаходиться опис помилки.

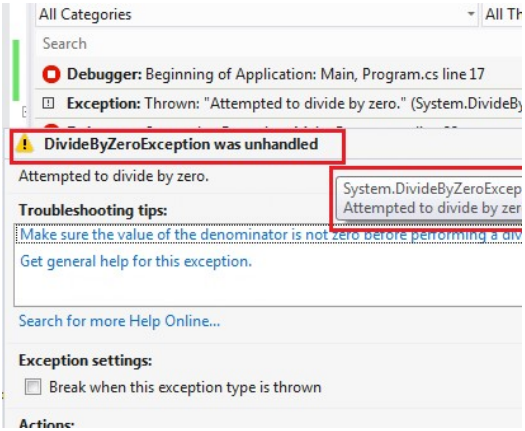

Рисунок 12. Виняткова ситуація

У списку з усунення неполадок цього вікна можна побачити можливі підказки, які можуть допомогти вам знайти можливе рішення. У даному випадку проблема проста і з нею можна розібратися без додаткових порад, тому що в рядку сталося ділення на нуль і наше завдання додати перевірку, яка запобіжить діленню на 0.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication2
{
    class Program
```

```
    {
        static int Double(int x)
        {
            return x * 2;
        }
        static void Main(string[] args)
        {
            int x = 10;
            int y = 15;
            if(y == 15)
            {
                Console.WriteLine("Змінна y не повинна бути 15");
                y = 16;
            }
            x = Double(x);

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

Але далеко не завжди так легко визначити помилку. Бувають випадки, коли потрібно пройти програму покроково, рядок за рядком, у пошуках вразливого місця, яке призвело до проблеми. Давайте спробуємо пройти програму покроково. Проходити всю велику програму крок за кроком проблематично. 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 запускає програму в налагоджувальному режимі. Щоб запустити програму без можливості налагодження, потрібно натиснути Ctrl + F5. Відповідні пункти меню можна знайти в меню Debug:

Потрібно відлагодити метод. Для цього потрібно поставити 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 в будь-якому місці програми, але тільки там, де є код і програма може перервати виконання

Для створення точки перейдіть на потрібний рядок і зробіть одне з наступного:

- 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 крапка зупинки (див. Рис.13). Якщо ще раз спробувати поставити точку зупину на цьому ж рядку, то точку зупинки буде знято. Точку зупинки можна ставити в будь-який момент, навіть під час виконання програми.

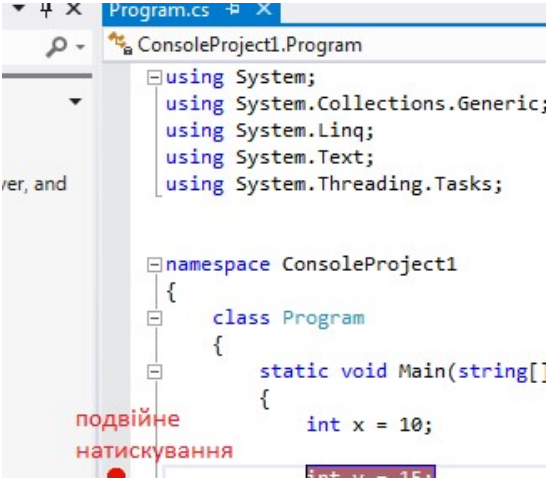

Рисунок 13. Точка зупинки

Якщо точку на поточному рядку встановити не можна, то в рядку стану середовища розробки з'явитися відповідне повідомлення (див. Рис. 14).

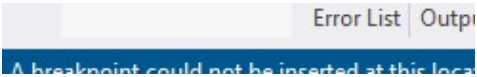

Рисунок 14. Точка зупинки

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) (див. Рис. 15). Внизу вікна середовища розробки з'явилася панель (див. рис. 15): **Call stack** - стек викликів. У цьому вікні перераховані методи, які вже були викликані раніше.

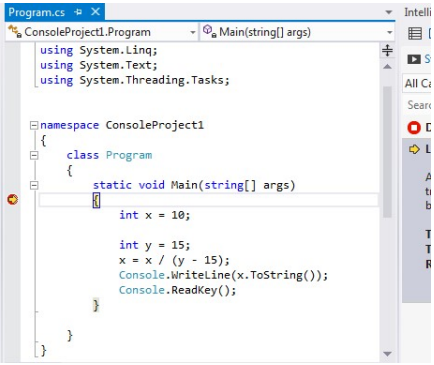

Рисунок 15. Покрокове виконання

Курсор покрокового виконання зупинився в нашій точці зупинки. Тепер потрібно почати тестування коду. На панелі, де з'явилася нова панель налагодження (див. Рис. 16), де є необхідні команди, також з’явилися коменди в меню Debug (див. рис. 17).

Рисунок 16. Панель покрокового виконання

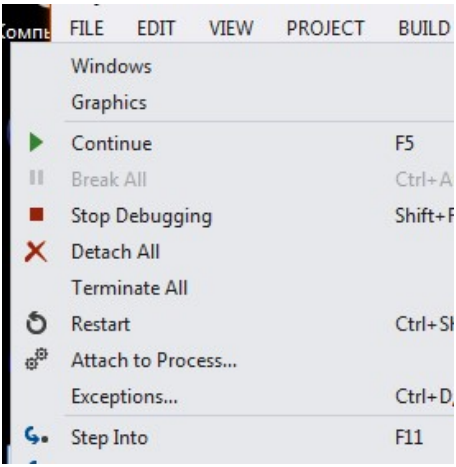

Рисунок 17. Меню Debug

Команди для покрокового виконання:
Continue (F5) - продовжити виконання програми;

Stop debugging (Shift+F5) - зупинити налагодження. При цьому виконання програми перерветься на тій точці, на якій зараз і знаходитися;

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

Step Into (F11) - виконати черговий оператор. Якщо це метод, то перейти в початок цього методу, щоб почати налагодження. Наприклад, якщо ви знаходитесь на рядку: `x = Method(x)` то, курсор покрокового виконання перейде на початок методу `Method` 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**.

Завдання для опрацювання матеріалу:

Завдання 1. Створіть проекти, які виводять текст «Hello World!» за шаблонами Console Application, Windows Forms Application та WPF Application.

Завдання 2. Використовуючи Visual Studio, створіть проект за шаблоном Console Application. Створіть клас, який представляє навчальну групу `Group`.

Створіть клас студент `Student`. В тілі класу створіть параметр ім'я та методи `void Study()`, `void Read()`, `void Write()`, `void Relax()`.

Створіть два похідних класи `GoodStudent`, `BadStudent` від класу базового класу `Student` і перевизначте кожен з методів, в залежності від успішності студента.

Конструктор класу `Group` приймає 3 аргументи типу `Student`. Група складається з 3-ох студентів.

Виведіть інформацію про імена студентів та про те, як всі студенти екземпляру класу `Group` уміють вчитися, читати, писати, відпочивати.

Завдання 3. Використовуючи Visual Studio, створіть проект за шаблоном Console Application. Створіть клас `Converter`. В тілі класу створити користувацький конструктор, який приймає два дійсні аргументи, і ініціалізує поля, які відповідають курсу долара та євро по відношенню до гривні – `public Converter(double usd, double eur)`.

Напишіть програму, яка буде виконувати конвертацію з гривні в одну з зазначених валют та навпаки.

Завдання 4. Використовуючи Visual Studio, створіть проект за шаблоном Console Application. Створіть клас «Трикутник» із полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметра. Створіть клас-нащадок «Рівносторонній трикутник», який має поле площі. Визначити метод обчислення площі.

Завдання 5. Використовуючи Visual Studio, створіть проект за шаблоном Console Application. Напишіть програму для роботи з геометричними фігурами (трикутник, круг, прямокутник, квадрат, ромб). У програмі створіть абстрактний

базовий клас «Фігура» з віртуальними методами обчислення площі й периметра та його нащадків – «Трикутник», «Круг», «Прямокутник», «Квадрат», «Ромб».

Завдання 6. Використовуючи Visual Studio, створіть проект за шаблоном Console Application. 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.

Запитання до теми:

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

Завдання для самостійної роботи. Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 1.

Аналіз задачі та розробка проекту.

Оформлення та захист звіту.

ТЕМА 2. ОСНОВНІ ЕТАПИ ЖИТТЄВОГО ЦИКЛУ (ЖЦ) ПС. ПРИЗНАЧЕННЯ ТА ВИКОРИСТАННЯ UML. ВИДИ ДІАГРАМ UML ТА ЇХ ВИКОРИСТАННЯ

Огляд, мета і призначення теми: 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. Призначення та використання UML

Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 1.

Завдання для опрацювання матеріалу:

Завдання 1. Використовуючи Visual Studio, створіть проект за шаблоном Modeling Projects.

Потрібно: створити діаграми прецедентів (Use Case), класів (Class) та послідовності (Sequence) для наступної задачі: інтернет-магазин.

Здійснює обробку замовлень, контроль доставки, складський облік, управління закупівлями, робота з клієнтською базою, контроль фінансів і аналіз продажів.

Для діаграми послідовності прецедент: замовлення клієнтом Івановим І.І. трьох телефонів №111 з доставкою на Глушкова 4Д.

Завдання 2. Потрібно: створити діаграми прецедентів (Use Case), класів (Class) та послідовності (Sequence) для наступної задачі: продаж квитків в кінотеатрі.

Завдання 3. Розробити щонайменше 8 UML-діаграм для лабораторної роботи № 1:

- діаграми варіантів використання (діаграми прецедентів) - 2..* (щонайменше до трьох прецедентів описати потоки подій);
- діаграми класів - 1..*;
- діаграми діяльності - 0..*;.
- діаграми взаємодії - 3..*;
- діаграми станів - 0..*;
- діаграми компонентів - 1..*;
- діаграми розгортання - 1..*.

Запитання до теми:

1. Які моделі життєвого циклу Ви знаєте?
2. Що таке сутності й відношення UML?

3. Перелічити й охарактеризувати основні види діаграм UML.
4. Що таке прикордонні й керівні класи, класи-сутності, абстрактні й конкретні класи?
5. Охарактеризувати відношення узагальнення й залежності між класами.
6. Охарактеризувати відношення асоціації, агрегації й композиції між класами.
7. Що таке діаграми станів і діяльності та яке їхнє призначення?
8. Що таке діаграми компонентів і розгортання та яке їхнє призначення?
9. Опишіть спрощену стратегію використання UML-діаграм при моделюванні ПС.
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. Практичне застосування UML для об'єктно-орієнтованого моделювання ПС. Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 1.

Аналіз задачі та розробка проекту.

Оформлення та захист звіту.

ТЕМА 3. ШАБЛОНИ ПРОЕКТУВАННЯ. ШАБЛОНИ СТВОРЕННЯ, СТРУКТУРНІ ТА ШАБЛОНИ ПОВЕДІНКИ

Огляд, мета і призначення теми: огляд основних етапів та моделей життєвого циклу програмних систем (ПС), розуміння місця мови UML в життєвому циклі ПС, огляд основних видів діаграм UML.

Лабораторні практикуми 3-4. Шаблони проектування

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 для самостійної роботи. Практичне застосування UML для об'єктно-орієнтованого моделювання ПС. Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 1.
Підготувати доповідь про один із шаблонів:

- Прототип (Prototype).
- 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. Діаграми класів та шаблони проектування. Структура шаблонів. Класифікація шаблонів: породжуючі, структурні, поведінки.
2. Різновиди шаблонів проектування.
3. Шаблон «Абстрактна фабрика».
4. Шаблон «Прототип».
5. Шаблон «Одиночний об'єкт».
6. Шаблон «Міст».
7. Шаблон «Ланцюг обов'язків».
8. Шаблон «Декоратор».
9. Шаблон «Ітератор».

Завдання для самостійної роботи.

Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 1.

1. Аналіз задачі та розробка проекту
2. Оформлення та захист звіту

ТЕМА 4. АНТИШАБЛОНИ: «ІНВЕРСІЯ АБСТРАКЦІЇ», «БОЖЕСТВЕННИЙ ОБ'ЄКТ», «МАГІЧНА КНОПКА», «МАГІЧНІ ЧИСЛА»

Огляд, мета і призначення теми: ознайомлення з антишаблонами в розробці ПС, типові антишаблони, типові випадки некоректних підходів до розробки ПС.

Вивчивши матеріал даної теми, студент зможе: охарактеризувати основні антишаблони.

Лабораторний практикум 5. Рефакторинг

Все більше спостерігається тенденція до збільшення тривалості ЖЦ програмних систем, зростає обсяг успадкованого коду. Тому надзвичайно важливими є задачі, пов'язані з полегшенням модифікації й розвитком існуючого програмного коду.

При внесенні в систему змін часто втрачається її структурованість, розібратися в ній стає важче і, як наслідок, її програмний код стає погано придатним для супроводження. Крім того, недбало спроектований код зазвичай займає занадто багато місця, у ньому часто трапляються повтори. Для розв'язання цих проблем застосовують рефакторинг.

Рефакторинг – це процес переписування програмного коду з метою поліпшення його внутрішньої структури та стилю. При цьому зовнішня поведінка програми не змінюється.

Рефакторинг здійснює зміни, пов'язані з методами, переміщеннями функцій між об'єктами, організацією даних, спрощенням умовних виразів і викликів методів, задачами узагальнення (виділення інтерфейсу, підкласу, батьківського класу тощо). Перелічимо коротко деякі з елементів рефакторингу.

Зміна сигнатури методу полягає в додаванні, зміні або видаленні параметра методу. Змінивши сигнатуру методу, необхідно скорегувати звернення до нього в коді всіх клієнтів. Ця зміна може торкнутися зовнішнього інтерфейсу програми. Якщо програмісту не доступні всі клієнти інтерфейсу, то може знадобитися та чи інша форма реєстрації змін інтерфейсу.

Інкапсуляція поля – якщо в класі є відкрите поле, то необхідно зробити його закритим і забезпечити методи доступу.

Виділення інтерфейсу – якщо кілька клієнтів використовують одну підмножину інтерфейсу класу або в кількох класах певна частина інтерфейсу спільна, то рефакторинг забезпечує виділення цієї підмножини в окремий інтерфейс.

Виділення класу – іноді клас реалізує функцію, яку слід розподілити між кількома класами. Тоді рефакторинг дозволяє створити новий клас, перемістити відповідні атрибути й методи зі старого класу в новий.

Виділення підкласу – якщо клас виконує різні види робіт, то доцільно створити підкласи, які йому спадкують і виконують кожен свій вид роботи.

Виділення надкласу – якщо маємо кілька класів зі схожими методами й полями, то можна перемістити їх у надклас (батьківський для всіх даних підкласів).

Виділення класу – іноді клас реалізує функцію, яку слід розподілити між кількома класами. Тоді рефакторинг дозволяє створити новий клас, перемістити відповідні атрибути й методи зі старого класу в новий.

Виділення підкласу – якщо клас виконує різні види робіт, то доцільно створити підкласи, які йому спадкують і виконують кожен свій вид роботи.

Виділення надкласу – якщо маємо кілька класів зі схожими методами й полями, то можна перемістити їх у надклас (батьківський для всіх даних підкласів).

Виділення інтерфейсу – якщо кілька клієнтів використовують одну підмножину інтерфейсу класу або в кількох класах певна частина інтерфейсу спільна, то рефакторинг забезпечує виділення цієї підмножини в окремий інтерфейс.

Виділення локальної змінної – якщо певний вираз обчислюється в тілі методу в кількох місцях, то його значення слід обчислити один раз і запам'ятати в новій локальній змінній. Наприклад, код

```
func(a+b, (a+b) / 2, a+b-c);
```

слід рефакторизувати в

```
t=a+b;
func(t, t/2, t-c);
```

попередньо оголосивши `t` локальною змінною відповідного типу.

Вбудовування змінної – підстановка єдиний раз присвоєного значення існуючої локальної змінної замість неї самої. Можна використовувати для економії (стекової) пам'яті, що корисно в рекурсивних функціях. Наприклад, код

```
double y=f(x);
if (y<0) {…}
```

слід рефакторизувати в

```
if (f(x)<0) {…}
```

Виділення методу полягає у виділенні з довгого чи важкого для сприйняття коду окремих його фрагментів і перетворенні їх в окремі методи (функції).

Підйом поля (методу) – якщо у двох підкласах є однакове поле чи метод, то їх слід підняти в батьківський клас.

Спуск поля (методу) – якщо в батьківському класі є поля чи методи, що належать лише до деяких його підкласів, то їх потрібно перемістити в ці підкласи.

Вбудовування методу протилежне виділенню методу. Коли тіло методу (функції) так само зрозуміле, як і його (її) назва, тоді виклик методу замінюють на код. Крім того, такий рефакторинг застосовують при невдалому розбитті на методи – тоді кілька методів об'єднують, а потім знову виділяють окремі методи.

Заміна умовного оператора поліморфізмом – умовний оператор із кількома розгалуженнями замінюється викликом поліморфного методу деякого базового класу, що має підкласи для кожної гілки вихідного оператора. Вибір гілки здійснюється неявно.

Переміщення методу застосовується для методу, що частіше звертається до іншого класу, ніж до того, у якому він розташований.

```
class library {
      book[1000] books;
}
class book {
      int id;
      int InLibrary(library Lib) {
            for(int i=0; i<1000; i++)
            if (id==Lib.books[i].id) return 1;
            return 0;
      }
}
```

При застосуванні останнього рефакторингу цей код перетвориться на наступний:

```
 class library {
      book[1000] books;
      int InLibrary(book Book) {
            for(int i=0; i<1000; i++)
            if (Book.id==books[i].id) return 1;
            return 0;
      }
}
class book {
      int id;
}
```

Рефакторинг в Visual Studio 2013

Переіменування. При виконанні операції Переіменування підсистема оптимізації виконує операцію переіменування, що призначена конкретно для кожного символа коду.

Переіменовувати можна назви полів, локальних змінних, методів, просторів імен, властивостей, типів, ідентифікаторів, …

У випадку переіменування члена, який реалізує /перевизначає чи який реалізується /перевизначається членами інших типів, Visual Studio відкриває діалогове вікно, в якому говориться, про те, що виконання операції призведе до каскадного переіменування.

Для виконання операції переіменування стоячи на імені, яке слід переіменувати натискаємо праву кнопку миші (див. Рис. 18):

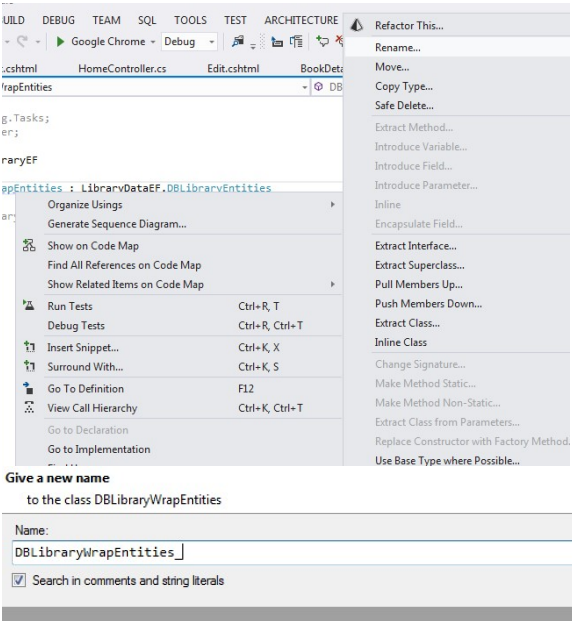

Рисунок 18. Рефакторинг. Переіменування

Переіменування методом смарт-тегів. Почніть змінювати ім'я (додавати чи видаляти символи) при цьому з'явиться підказка з пропозицією переіменування (див. рис. 19).

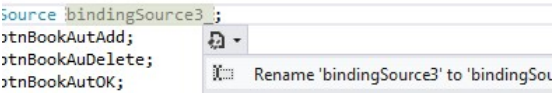

Рисунок 19. Рефакторинг. Переіменування

Виділення методу полягає у виділенні з довгого чи важкого для сприйняття коду окремих його фрагментів і перетворенні їх в окремі методи (функції).

Новий, виділений метод містить виділений код, а виділений код в існуючому елементі замінюється на виклик нового методу.

Для використання виділяємо потрібний рядок (рядки) та натискаємо праву кнопку миші (див. рис. 20-22).

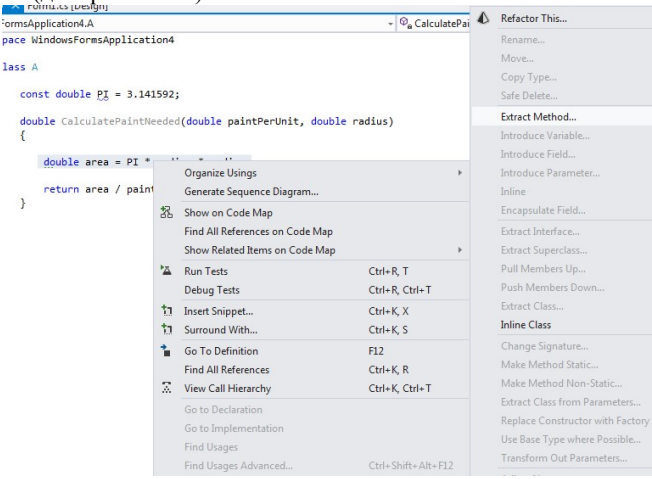

Рисунок 20. Рефакторинг. Виділення методу

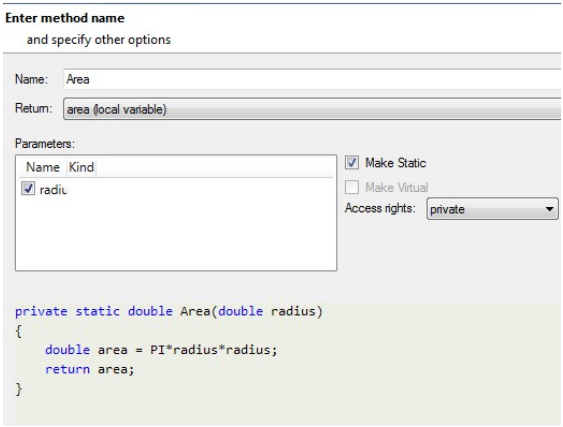

Рисунок 21. Рефакторинг. Виділення методу

```
class A
{
    const double PI = 3.141592;

    double CalculatePaintNeeded(double paintPerUnit, 
    {
        var area = Area(radius);

        return area / paintPerUnit;
    }

    private static double Area(double radius)
    {
        double area = PI*radius*radius;
```

Рисунок 22. Рефакторинг. Виділення методу

Інкапсуляція поля – дозволяє створювати з існуючого поля властивість, а потів послідовно оновити код, включивши до нього посилання на нову властивість. Якщо в класі є відкрите поле, то необхідно зробити його закритим і забезпечити методи доступу.

Якщо поле `public`, то при інкапсуляції поля модифікатор доступу замінюється на `private`, а потім генеруються методи досупу `get` та `set`. Стоячи на визначенні потрібного поля напишніть праву кнопку миші (див. рис. 23-26).

```
class Square
{
    public int width, height;
}
class MainClass
{
    public static void Main()
    {
        Square mySquare = new Square();
        mySquare.width = 110;
        mySquare.height = 150;
        // Output values for width and height.
        Console.WriteLine("width = {0}", mySquar
```

Рисунок 23. Рефакторинг. Інкапсуляція поля

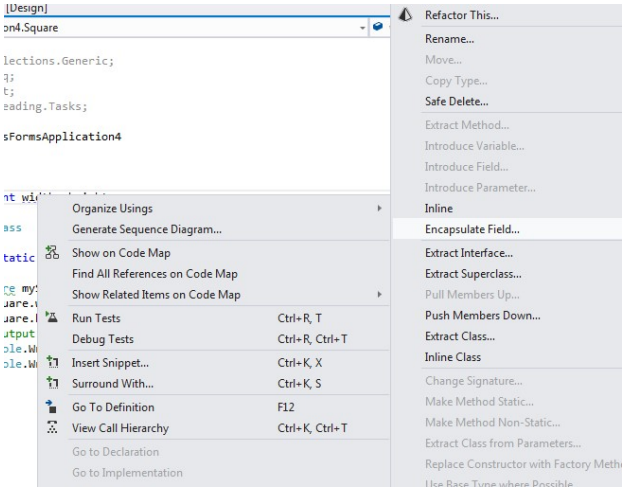

Рисунок 24. Рефакторинг. Інкапсуляція поля

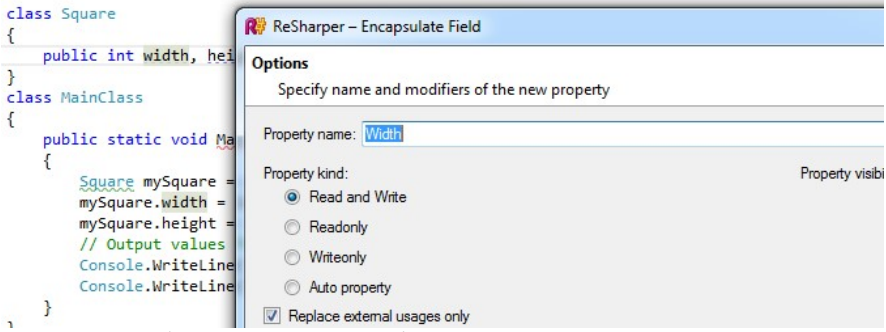

Рисунок 25. Рефакторинг. Інкапсуляція поля

```
class Square
{
    private int width;
    public int height;

    public int Width
    {
        get { return width; }
        set { width = value; }
    }
}
class MainClass
{
    public static void Main()
    {
        Square mySquare = new Square();
        mySquare.Width = 110;
        mySquare.height = 150;
        // Output values for width and height
```

Рисунок 26. Рефакторинг. Інкапсуляція поля

Виділення інтерфейсу – якщо декілька клієнтів використовують одну підмножину інтерфейсу класу або в кількох класах певна частина інтерфейсу спільна, то рефакторинг забезпечує виділення цієї підмножини в окремий інтерфейс.

При виконанні цієї операції в новому файлі генерується інтерфейс і курсор поміщається на початок цього файлу (див. Рис. 27-30).

```
class A
{
    public void Method1(string s
    public void Method2(string s
    public void Method3(string s
}

class B
{
    public void Method1(string s
    public void Method2(string s
```

Рисунок 27. Рефакторинг. Виділення інтерфейсу

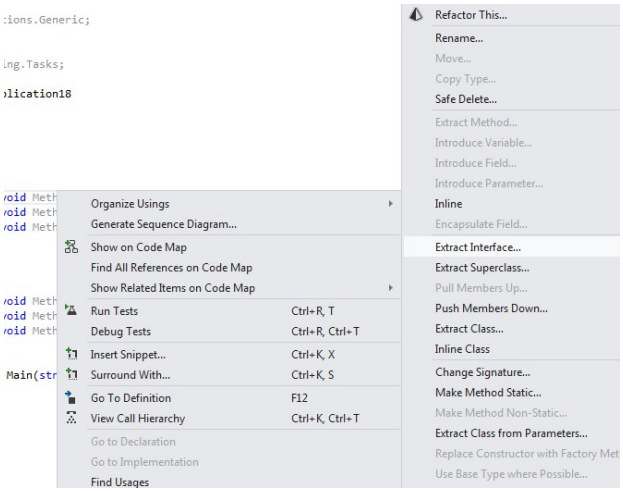

Рисунок 28. Рефакторинг. Виділення інтерфейсу

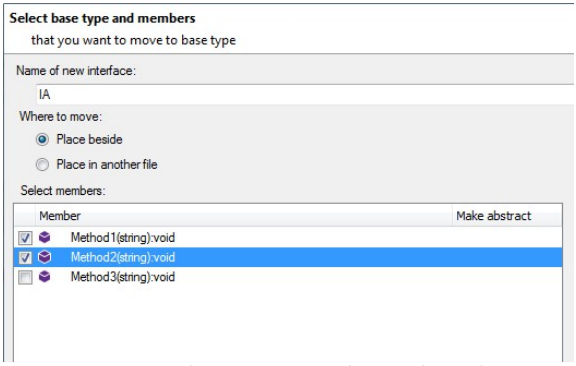

Рисунок 29. Рефакторинг. Виділення інтерфейсу

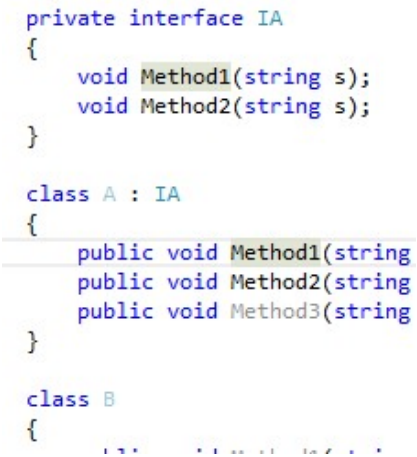

Рисунок 30. Рефакторинг. Виділення інтерфейсу

Зміна сигнатури — забезпечує простий спосіб видалення параметрів з методів. Ця операція змінює оголошення; в довільних місцях, де викликається член, параметр видаляється (див. Рис. 31-35). Розміщуємо курсор, в межах методу (див. Рис. 31).

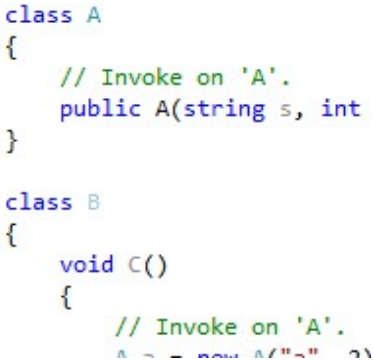

Рисунок 31. Рефакторинг. Зміна сигнатури

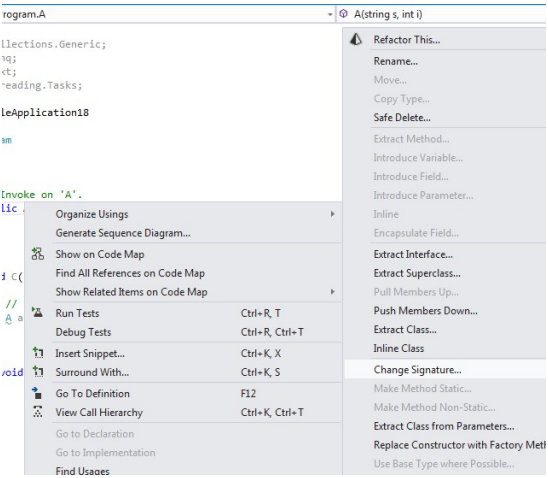

Рисунок 32. Рефакторинг. Зміна сигнатури

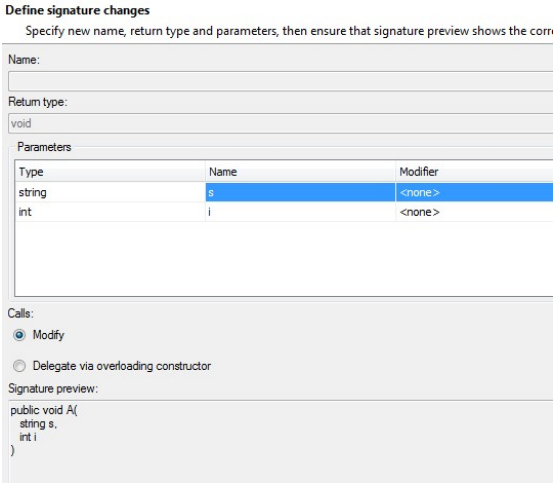

Рисунок 33. Рефакторинг. Зміна сигнатури

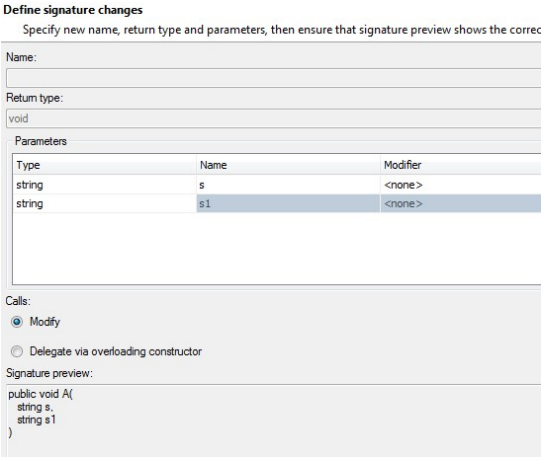

Рисунок 34. Рефакторинг. Зміна сигнатури

```
class A
{
    // Invoke on 'A'.
    public A(string s, string
}

class B
{
    void C()
    {
        // Invoke on 'A'.
        A a = new A("a", TODO)
```

Рисунок 35. Рефакторинг. Зміна сигнатури

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 проекту лабораторного практикуму № 1.

1. Аналіз задачі та розробка проекту.
2. Оформлення та захист звіту.

ТЕМА 5. ПРИНЦИПИ ПРОЕКТУВАННЯ «S.O.L.I.D.»

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 проектування S.O.L.I.D. та уміти їх застосовувати на практиці.

Лабораторний практикум 6. Принципи проектування класів (S.O.L.I.D.)
Опитування та обговорення лекційного матеріалу.

Виконання лабораторної роботи № 2.

Завдання для самостійної роботи.
Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 2.

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 лекційного матеріалу та власного проекту лабораторного практикуму № 2.

Аналіз задачі та розробка проекту.

Оформлення та захист звіту.

Продовження роботи над шаблонами (див. тема 3).

ЗМІСТОВИЙ МОДУЛЬ 2.

ОГЛЯД ТЕХНОЛОГІЇ .NET FRAMEWORK. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ C#. РОБОТА З XML-ДОКУМЕНТАМИ

ТЕМА 6. ОГЛЯД ТЕХНОЛОГІЇ .NET FRAMEWORK. СЕРЕДОВИЩЕ CLR. БІБЛІОТЕКА КЛАСІВ FRAMEWORK CLASS LIBRARY (FCL)

ТЕМА 7. ТИПИ І ЗМІННІ

Огляд, мета і призначення тем: огляд технології .NET Framework, середовище CLR, бібліотека класів Framework Class Library (FCL), загальна структура C#-програми.

Типи і змінні. Типи значень (вбудовані типи значень, порядковий тип (`enum`),структури (`struct`)). Типи посилань (вбудовані типи посилань (`object`, `string`), масиви, делегати, інтерфейси, класи).

Робота з рядками. Класи. Конструктори та деструктори. Константи. Методи. Властивість. Індексатори. Події. Делегати.

Вивчивши матеріал даної теми, студент зможе:

Охарактеризувати технології .NET Framework, середовище CLR, бібліотека класів Framework Class Library (FCL), загальна структура C#-програми.

Лабораторні практикуми 7-8. Типи і змінні. Типи значень. Порядковий тип, структури. Типи посилань. Клас як тип даних. Рядки. Наслідування класів. Наслідування як модель відношення “загальне-конкретне”. Втілення в мовах програмування. Перетворення типів.
Опитування та обговорення лекційного матеріалу.

Виконання лабораторної роботи № 2.

Завдання для опрацювання матеріалу:

Завдання 1. Використовуючи Visual Studio, створіть проект за шаблоном Windows Forms Application (див. рис. 36). В ньому створіть клас `ClassA` з полями `_a` і `_b` та властивістю `C`. Властивість – значення виразу над полями `_a` і `_b` (вираз і типи полів - див. варіант в таблиці 1). Конструктор залишити за замовчуванням.

Простежити, щоб поля `_a` і `_b` безпосередньо в інших класах були недоступні. В методі `get` поля `C` задати обчислення функції відповідно до варіанту.

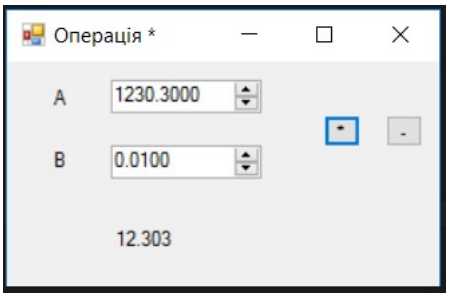

Рисунок 36. Зовнішній вигляд форми для завдання 1

Таблиця 4. Варіанти завдань до завдання №1

Варіант	Тип полів a та b	Вирази
1.	`float`	`/, -`
2.	`float`	`/ =, +`
3.	`int`	`*, -`
4.	`int`	`* =, +`
5.	`int`	(постфіксний) `++, * =, -`
6.	`float`	`/ =,` (постфіксний) `--, *`
7.	`float`	`* =, ++` (префіксний), `/`
8.	`int`	`* =,` (постфіксний) `++, -`
9.	`int`	`+ =, -` (префіксний), `-`
10.	`int`	`- =,` (постфіксний) `++, -`
11.	`float`	`/ =, *`
12.	`float`	`* =, /`
13.	`int`	`-` (префіксний), `++` (префіксний), `+`
14.	`byte`	`-,` (постфіксний) `++, -`
15.	`byte`	`* =, +`
16.	`byte`	`* =, -`

Завдання 2. Використовуючи Visual Studio, створіть проект за шаблоном Windows Forms Application (див. рис. 37). В ньому створіть клас `ClassB` – успадкований від класу `ClassA` (з попереднього завдання) з полем `_d` і властивістю `C2`. Властивість `C2` – результат обчислення виразу над полями `_a`, `_b`, `_d` (відповідно до варіанту табл. 2). У тілі властивості використовувати оператор (див. варіант в таблиці 2). У класу `ClassA` створити конструктор. Для класу `ClassB` визначити 2 конструктора: один – успадковується від конструктора класу `ClassA`, другий – власний. В конструкторі форми після ініціалізації створити об'єкти класів `ClassA` і `ClassB`, продемонструвавши роботу всіх конструкторів. Вивести значення властивості на екран.

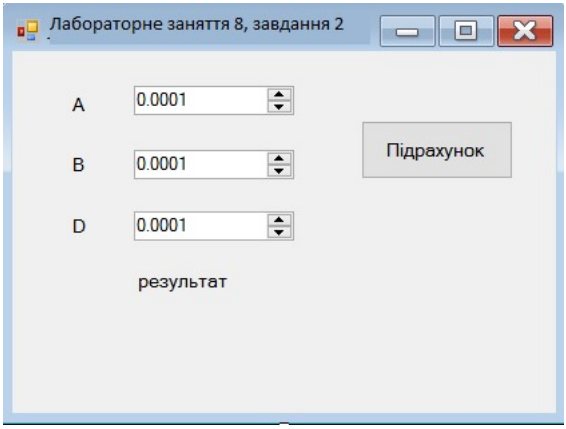

Рисунок 37. Зовнішній вигляд форми для завдання 2.

Таблиця 5. Варіанти завдань до завдання №2

Варіант	Керуючий оператор
1.	`If`
2.	`while`
3.	`switch`
4.	`for`
5.	`while`

Завдання 3. Використовуючи Visual Studio, створіть проект за шаблоном Windows Forms Application. Створіть клас `Student`, який містить наступні закриті поля:

• прізвище студента;

• номер студентського квитка;

• дата народження.

Створити клас `Group`, який містить закритий масив елементів типу `Student`. Забезпечити наступні можливості:

• виведення інформації про студента за номером студентського;

• виведення інформації про студента за номером за допомогою індексу;

• виведення на екран інформації про студента, прізвище якого введено з клавіатури, якщо таких студентів немає, видати відповідне повідомлення;

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 індексу першого елемента в масиві?
15. Що таке делегат?

Завдання для самостійної роботи

Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 2.

ТЕМА 8. СТРУКТУРИ ДАНИХ ЇХ ПОРІВНЯЛЬНИЙ АНАЛІЗ. СПИСКИ, СТЕКИ, ЧЕРГИ. МАСИВИ ТА ВИДИ КОЛЕКЦІЙ В МОВІ C#. ІТЕРАТОРИ. КЛАСИ. КОНСТРУКТОРИ ТА ДЕСТРУКТОРИ. КОНСТАНТИ. МЕТОДИ. ВЛАСТИВІСТЬ. ІНДЕКСАТОРИ. ПОДІЇ. ДЕЛЕГАТИ

Огляд, мета і призначення теми: Структури даних їх порівняльний аналіз. Списки, стеки, черги. Ітератори.

Лабораторний практикум 9. Структури даних їх порівняльний аналіз. Списки, стеки, черги. Масиви та види колекцій в мові C#. Ітератори. Класи. Конструктори та деструктори. Константи. Методи. Властивість. Індексатори. Події. Делегати.
Опитування та обговорення лекційного матеріалу.

Виконання лабораторної роботи № 2.

Завдання для опрацювання матеріалу:

Завдання 1. Використовуючи Visual Studio, створіть проект за шаблоном Console Application. Створіть колекцію `MyList<T>`. Реалізуйте в найпростішому наближенні можливість використання її екземпляра аналогічно екземпляру класу `List<T>`. Реалізуйте, як мінімум, наступну взаємодію з екземпляром: метод додавання елемента, індексатор для отримання значення елемента за індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізуйте можливість перебору елементів колекції в циклі `foreach`.

Завдання 2. Використовуючи Visual Studio, створіть проект за шаблоном Console Application. Створіть колекцію `MyDictionary<TKey,TValue>`. Реалізуйте в найпростішому наближенні можливість використання її екземпляра аналогічно екземпляру класу `Dictionary<TKey,TValue>`. Реалізуйте, як мінімум, наступну взаємодію з екземпляром: метод додавання елемента, індексатор для отримання значення елемента за індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізуйте можливість перебору елементів колекції в циклі `foreach`.

Завдання 3. Оголосити делегат `private delegate double MathAction(double num)`, описати методи `private static double DoubleAction(double input)` (подвоєння аргументу), `private static double SquareAction(double input)` (піднесення аргументу до квадрату). Створити два

екземпляри делегатів, прив'язавши їх до створених вище методів. Створити ще один екземпляр делегата з анонімним методом, який підносить значення параметру до кубу. Результати викликів усіх цих методів з параметром 5 вивести на екран.

Запитання до теми:

1. Що таке колекція?
2. Що таке оператор `yield`?
3. Опишіть алгоритм роботи циклу `foreach`.

Завдання для самостійної роботи

Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 2.

ТЕМА 9. ПОШУК В НЕВПОРЯДКОВАНОМУ ТА ВПОРЯДКОВАНОМУ МАСИВАХ. ХЕШ-ТАБЛИЦІ. АЛГОРИТМИ СОРТУВАННЯ. ЇХ РЕАЛІЗАЦІЯ В C#

ТЕМА 10. ПЕРЕТВОРЕННЯ ТИПІВ. ОПЕРАТОРИ ТА ВИРАЗИ. ОПЕРАТОРИ ОБРОБКИ ВИКЛЮЧЕНЬ. НЕЯВНІ ТА АНОНІМНІ ТИПИ

Огляд, мета і призначення теми: пошук в невпорядкованому та впорядкованому масивах. Хеш-таблиці. Алгоритми сортування. Їх реалізація в C#. Масиви та види колекцій в мові C#.

Перетворення типів:

- Неявні перетворення.
- Явні перетворення (приведення).
- Користувацькі перетворення.
- Перетворення за допомогою допоміжних класів.
- Операції `as` та `is`.

Оператори та вирази (відмінні від операторів та виразів C++). Директива using. Оператор `foreach`. Оператори обробки виключень (`throw`, `try-finally`, `try-catch`, `try-catch-finally`, оператор `using`).Неявні типи, анонімні типи (тип `var`) та тип `dynamic`.

Лабораторний практикум 10. Пошук в невпорядкованому та впорядкованому масивах. Хеш-таблиці. Алгоритми сортування. Їх реалізація в C#. Перетворення типів. Оператори та вирази. Оператори обробки виключень. Неявні та анонімні типи.
Опитування та обговорення лекційного матеріалу.

Виконання лабораторної роботи № 2.

Завдання для опрацювання матеріалу:

Завдання 1. Використовуючи Visual Studio, створіть проект за шаблоном Console Application.

Обчисліть середнє арифметичне трьох цілочисельних значень і виведіть його на екран.

З якою проблемою ви зіштовхнулися? Який тип змінних краще використовувати для коректного відображення результату?

Завдання 2. Опишіть клас з іменем `Student`, який містить наступні поля: П.І.П.; номер студентського квитка; рік вступу.

Написати програму, яка виконує наступні дії:

- введення з клавіатури даних в масив, який складається з десяти елементів типу `Student` (записи повинні бути впорядковані за алфавітом (за номером студентського));
- якщо значення року введено не у відповідному форматі видати виключення.
- виведення на екран прізвище студента (-ів), який (-і) навчаються довше за введене значення.

Запитання до теми:

1. Що таке перетворення значень типів?
2. В чому різниця явного і неявного перетворення значення типу?
3. Що робить оператор `sizeof()`?
4. Що таке виключення?
5. Що таке конструкція `try - catch`?
6. Що таке конструкція `try - catch - finally`?
7. Як створити користувацьке виключення?

8. Що таке анонімні типи?
9. Що таке динамічний тип?
10. Що робить оператор `dynamic`?

Завдання для самостійної роботи:
Виконання лабораторної роботи № 2.

ТЕМА 11. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ C#. ЛЯМБДА-ВИРАЗИ. ЗАСТОСУВАННЯ ЛЯМБДА-ВИРАЗІВ

Лабораторний практикум 11. Функціональні можливості C#. Лямбда-вирази. Застосування лямбда-виразів.
Опитування та обговорення лекційного матеріалу

Практичне застосування лямбда-виразів.

Виконання лабораторної роботи № 3.

Завдання для опрацювання матеріалу:

Завдання 1. Опрацюйте усі приклади з лекції.

Завдання 2. Дано цілочисельну послідовність, яка містить як додатні, так і від'ємні числа. Вивести її перший додатній елемент і останній від'ємний.

Завдання 3. Дано ціле число `L (> 0)` і рядкова послідовність `A`. Вивести останній рядок з `A`, який починається з цифри і має довжину `L`. Якщо потрібних рядків в послідовності `A` немає, то вивести рядок «Not found».

Завдання 4. Дано цілочисельну послідовність. Знайти кількість її від'ємних елементів, а також їх суму.

Завдання 5. Дано целі числа `K1` і `K2` і цілочисельні послідовності `A` і `B`. Отримати послідовність, яка містить всі числа з `A`, що більші `K1`, і всі числа з `B`, меньші за `K2`. Відсортувати отриману послідовність за зростанням.

Запитання до теми:

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

Завдання для самостійної роботи:

Функціональне програмування. Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 3.

1. Аналіз задачі та розробка проекту.
2. Кодування модулів.
3. Тестування.

ТЕМА 12. ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ C#. LINQ: МОВА ІНТЕРПРЕТОВАНИХ ЗАПИТІВ. ОПЕРАЦІЇ ЗАПИТІВ LINQ. ВИКОРИСТАННЯ LINQ ДЛЯ ДОСТУПУ ДО СТРУКТУР ДАНИХ В ПАМ'ЯТІ (LINQ TO OBJECTS))

Огляд, мета і призначення теми: LINQ (Language Integrated Query – мова інтегрованих запитів). Призначення та синтаксис.

- LINQ to Objects.
- LINQ to XML.
- LINQ to DataSet та SQL.
- LINQ to Entities.

Лабораторний практикум 12. Функціональні можливості C#. LINQ: мова інтерпретованих запитів. Операції запитів LINQ. Використання LINQ для доступу до структур даних в пам'яті (LINQ to Objects).

1. Опитування та обговорення лекційного матеріалу.
2. Практичне застосування LINQ to Objects.
3. Виконання лабораторної роботи № 3.

Завдання для опрацювання матеріалу:

Завдання 1. Опрацюйте усі приклади з лекції.

Завдання 2. В наступному прикладі, використовуючи LINQ to Object знайти усі 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 для самостійної роботи:

Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 3.

ТЕМА 13. РОБОТА З ФАЙЛАМИ ТА ДИРЕКТОРІЯМИ

Огляд, мета і призначення теми: Робота з файлами та директоріями. (папками, каталогами) Робота з файлами в директорії. Створення підкаталогів. Робота з файлами (читання, запис, створення, видалення). Робота з різними кодуваннями.

Лабораторний практикум 13. Робота з файлами та директоріями.

Опитування та обговорення лекційного матеріалу.

Практичне застосування LINQ to Objects.

Виконання лабораторної роботи № 3.

Завдання для опрацювання матеріалу:

Завдання 1. Використовуючи Visual Studio, створіть проект за шаблоном Windows Forms Application. В `TreeView` відобразіть диски. Надайте можливість відкривати диски та директорії (навігація). Створіть кнопку, які дозволяють створювати текстовий файл в поточній директорії, та кнопку видалення виліленого файлу. Створіть меню, де передбачте пункти: Файл (створити, видалити), Допомога (Про програму, Інструкція користувача).

Запитання до теми:

1. Робота з файлами в директорії.
2. Створення підкаталогів.
3. Робота з файлами (читання, запис, створення, видалення).

Завдання для самостійної роботи:

Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму.

ТЕМА 14. ОСНОВНІ КОМПОНЕНТИ XML ДОКУМЕНТА. ЗАСТОСУВАННЯ XML ДОКУМЕНТА. МОДЕЛІ DOM ТА SAX ДЛЯ ОБРОБКИ XML. ЗАСТОСУВАННЯ LINQ ТО XML

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. System.Xml. Застосування XmlDocument. Застосування XmlReader. LINQ to XML. Порівняння LINQ to XML та DOM. Класи XObject, XAttribute, XNode, XContainer, XElement, XDocument. Застосування LINQ to XML.

Лабораторний практикум 14. Основні компоненти XML документа. Застосування XML документа. Моделі DOM та SAX для обробки XML. Застосування LINQ to XML.

Опитування та обговорення лекційного матеріалу.

Практичне застосування LINQ to Objects.

Виконання лабораторної роботи № 3.

Завдання для опрацювання матеріалу:

Завдання 1. 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 2. Виконайте попереднє завдання з використанням SAX.

Завдання 3. Виконайте попереднє завдання з використанням LINQ to XML.

Запитання до теми:

1. Основні компоненти XML-документа.
2. Моделі DOM та SAX для обробки XML.
3. Застосування XmlDocument. Застосування XmlReader.
6. LINQ to XML.
7. Порівняння LINQ to XML та DOM.

Завдання для самостійної роботи:

Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 3.

ТЕМА 15. МОВА ЗАПИТІВ XPATH. ТРАНСФОРМАЦІЯ XML

Огляд, мета і призначення теми: Мова запитів XPath. Трансформація XML. Трансформація XML з використанням XSLT. Шаблон XSL. Клас `XslCompiledTransform`. Приклад трансформації XML в HTML.

Лабораторний практикум 15. Мова запитів XPath. Трансформація XML.

Опитування та обговорення лекційного матеріалу.

Практичне застосування LINQ to Objects.

Виконання лабораторної роботи № 3.

Завдання для опрацювання матеріалу:

Завдання 1. Опрацювати усі приклади з лекції.

Завдання 2. Трансформувати створений на попередньому занятті файл до формату HTML.

Запитання до теми:

1. Мова запитів XPath.
2. Трансформація XML.
3. Трансформація XML з використанням XSLT.
4. Шаблон XSL.

Завдання для самостійної роботи. Опрацювання лекційного матеріалу та власного проекту лабораторного практикуму № 3.

КОНТРОЛЬ ЗНАНЬ І РОЗПОДІЛ БАЛІВ, ЯКІ ОТРИМУЮТЬ СТУДЕНТИ

Передбачається, що до кінця семестру кожен студент:
• виконає та захистить три лабораторні роботи;
• успішно напише дві модульних контрольних роботи.
В кінці семестру передбачено іспит за всіма темами семестру.

У випадку відсутності студента з поважних причин відпрацювання та перездачі модульних контрольних робіт здійснюються у відповідності до «Положення про порядок оцінювання знань студентів при кредитно-модульній системі організації навчального процесу» від 1 жовтня 2010 року.

Порядок розрахунку загальної оцінки. Модульні лабораторні роботи (проекти) – 30 = 10+10+10 балів. За невчасне чи неякісне виконання роботи знімаються штрафні бали. Семестрові контрольні роботи – 20 = 10+10 балів. За активну роботу студент може отримати до 10 балів (зокрема відвідування до 3 балів, створення звіту з виконаної роботи до 3 балів, до 4 балів за unit-тести (по 1 балу за тест)).

Студент має право на одне перескладання контрольної роботи із можливістю отримання 7 балів. Термін перескладання визначається викладачем.

Терміни здачі лабораторних робіт першого семестру:

- 1 лабораторна робота – до 6 тижня семестру;
- 2 лабораторна робота – до 10 тижня семестру;
- 3 лабораторна робота – до 14 тижня семестру.

У разі неякісного виконання лабораторної роботи, викладач має право не зарахувати лабораторну роботу, або знизити за неї бали.

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

Наприклад, якщо лабораторну роботу № 1 студент буде здавати в перший день сьомого тижня семестру, то за першу лабораторну роботу він може отримати максимум 8 балів.

Розрахунок балів по модулях:

Модуль 1: 25 балів (виконання завдань на лабораторних заняттях (у випадку обрання студентом форми роботи № 1), або модульна лабораторна робота (проект) № 1 (у випадку обрання форми роботи №2) – 10, модульна контрольна робота –10 балів, за активну роботу – 5 балів).

Модуль 2: 35 балів (модульна лабораторна робота (проект) № 2 – 10, модульна лабораторна робота (проект) № 3 – 10, модульна контрольна робота –10 балів, за активну роботу – 5 балів).

Семестрова оцінка – 60 балів; із них модульні лабораторні роботи (проекти) – 30 балів, за активну роботу – 10 балів, семестрові контрольні роботи – 20 бал. Підсумкова екзаменаційна робота – 40 балів. Максимальна підсумкова оцінка –100 балів.

Студент допускається до складання екзамену, якщо кількість набраних за семестр балів не менше 21 балу.

Підсумкова екзаменаційна робота – 40 балів.

При цьому, кількість балів:

- **1-34** відповідає оцінці «незадовільно» з обов'язковим повторним вивченням дисципліни;
- **35-59** відповідає оцінці «незадовільно» з можливістю повторного складання;
- **60-64** відповідає оцінці «задовільно» («достатньо»);
- **65-74** відповідає оцінці «задовільно»;
- **75 - 84** відповідає оцінці «добре»;
- **85 - 89** відповідає оцінці «добре» («дуже добре»);
- **90 - 100** відповідає оцінці «відмінно».

Таблиця 6. Шкала відповідності

За 100 – бальною шкалою	За національною шкалою	
90 – 100	5	Відмінно
85 – 89	4	Добре
75 – 84		
65 – 74	3	Задовільно
60 – 64		
35 – 59	2	не задовільно
1 – 34		

ПЕРЕЛІК ВЖИВАНИХ СКОРОЧЕНЬ

CLR	Common Language Runtime – вихідний файл виконання мов
FCL	Framework Class Library – стандартна бібліотека класів платформи .NET Framework
HTML	HyperText Markup Language – мова розмітки гіпертекстових документів
IDE	Integrated development environment – інтегроване середовище розробки
LINQ	Language Integrated Query – мова інтегрованих запитів
UML	Unified Modeling Language – уніфікована мова моделювання
XML	Extensible Markup Language – розширювана мова розмітки
АРМ	автоматизоване робоче місце
БД	база даних
БНФ	нотація Бекуса-Наура
ЖЦ	життєвий цикл
ООП	об’єктно-орієнтоване програмування
П.І.П.	прізвище, ім’я, по-батькові
ПС	програмна система

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. – М.: Издательский дом «Вильямс», 2000. – 384 с.
2. Боггс У., Боггс М. UML и Rational Rose 2002. – Издательство: ЛОРИ, 2004. – 582 с.
3. Буч Г. Объектно-ориентированный анализ и проектирование. С примерами приложений на C++ : Пер. с англ. – 2-е изд. – СПб.: Невский диалект, 2001. – 368 с.
4. Буч Г., Якобсон А., Рамбо Дж. UML. Классика CS. 2-е изд. / Пер. с англ.; Под общей редакцией проф. С. Орлова — СПб.: Питер, 2006. — 736 с.
5. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования /Питер, 2010. – 366 с.
6. Зубенко В.В., Омельчук Л.Л. Програмування : навчальний посібник (гриф МОН України) / - К. : ВПЦ "Київський університет", 2011. - 623 с.
7. Нейгел К., Ивьен Б. C# 5.0 и платформа .NET 4.5 для профессионалов/ М.: Вильямс, 2014. – 1440 с.
8. Лавріщева К. М. Програмна інженерія.- К.: BHV. 2007. - 413 с.
9. Объектно-ориентированный анализ и проектирование. С примерами приложений. Буч Гради, Максимчук Роберт А., Энгел Майкл У., Янг Бобби Дж., Коналлен Джим, Хьюстон Келли А.: Пер. с англ. – 3-е изд. – М.: ООО «И.Д.Вильямс», 2008. – 720 с.
10. Ставровский А.Б, Карнаух Т.О. Перші кроки програмування.- К.:Диалектика. 2005. – 389 с.
11. Фримен А., Раттц Д.С. LINQ. Язык интегрированных запросов в C# 2010 для профессионалов / М.: Вильямс, 2011. – 656с.

Навчальне видання

Бойко Борис Іванович
Омельчук Людмила Леонідівна
Русіна Наталія Геннадіївна

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Лабораторний практикум

Навчальний посібник

Кафедра теорії та технології програмування
факультету комп'ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка

Формат 60×84 1/16. Гарнітура Times New Roman.
Набір комп'ютерний. Ум. друк. арк 5,23.
Наклад 300 прим. Підписано до друку 15.12.2016.

Виготовлювач: Типографія «Айс Принт»
Тел: +38 (099) 192-00-33, +38 (048) 706-92-82
Site: www.ice-print.com.ua
E-mail:info@ice-print.com.ua