МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Методичні рекомендації до лабораторних робіт

для студентів спеціальностей
122 "Комп'ютерні науки",
126 "Інформаційні системи та технології"
першого (бакалаврського) рівня

Харків
ХНЕУ ім. С. Кузнеця
2021

УДК 004.42(07.034)
О-29

Укладач Ю. Е. Парфьонов

Затверджено на засіданні кафедри інформаційних систем.
Протокол № 6 від 05.01.2021 р.

Самостійне електронне текстове мережеве видання

Об'єктно-орієнтоване програмування [Електронний ресурс] : О-29 методичні рекомендації до лабораторних робіт для студентів спеціальностей 122 "Комп'ютерні науки", 126 "Інформаційні системи та технології" першого (бакалаврського) рівня / уклад. Ю. Е. Парфьонов. – Харків : ХНЕУ ім. С. Кузнеця, 2021. – 57 с.

Наведено методичні рекомендації, що сприяють поглибленню й закріпленню знань з теоретичних питань об'єктно-орієнтованого програмування (ООП) та надбанню практичних навичок з розроблення об'єктно-орієнтованих застосунків.

Рекомендовано для студентів спеціальностей 122 "Комп'ютерні науки", 126 "Інформаційні системи та технології" першого (бакалаврського) рівня.

УДК 004.42(07.034)

Вступ

Відомо, що використання потужних комп'ютеризованих засобів неможливе без програмного забезпечення. Галузь розроблення програмного забезпечення набуває все більшого значення, оскільки складність та функціональні можливості комп'ютерної техніки, що швидко зростають, потребують більш досконалих програмних засобів для задоволення потреб користувачів.

Істотною рисою таких програмних систем є рівень складності: для одного розробника практично неможливо охопити всі їхні аспекти. Причому ця складність є неминучою: з нею можливо справитися, але позбавитися від неї неможливо.

У теперішній час найбільш розповсюдженим методом боротьби зі складністю є об'єктно-орієнтований підхід до розроблення програмного забезпечення. З використанням цього підходу створюється більша частина програм у всьому світі. Отже, відповідні фахівці мають добре володіти концепціями об'єктно-орієнтованої парадигми. Це дає можливість їхнього практичного використання під час розроблення застосунків будь-якою об'єктно-орієнтованою мовою програмування.

Базовими мовами програмування для виконання лабораторних робіт є C# та Java, які інтенсивне використовуються компаніями-розробниками програмного забезпечення. В цих мовах також реалізовано більшість концепцій об'єктно-орієнтованого підходу в "класичному" вигляді. Їх також досить легко вивчати, тому що C# та Java є C-подібними мовами програмування, синтаксис яких має багато спільних рис.

Запропонований курс лабораторних робіт спрямовано на формування стійких практичних навичок щодо розроблення застосунків з використанням об'єктно-орієнтованого підходу.

Для розроблення програм студент має можливість вибрати мову програмування C# або Java. Також рекомендується використовувати інтегровані середовища розробки, такі як: Microsoft Visual Studio, JetBrains Rider, IntelliJ IDEA та Eclipse.

Загальні рекомендації до виконання лабораторних робіт

Необхідним елементом успішного засвоєння матеріалу навчальної дисципліни "Об'єктно-орієнтоване програмування" є самостійна робота студентів із технічною літературою. Тому в процесі виконання лабораторних робіт, поряд з використанням теоретичних відомостей щодо тематики кожної роботи, наведених у цих методичних рекомендаціях, також доцільно переглянути певні джерела зі списку рекомендованої літератури, зокрема навчальний посібник "Основи об'єктно-орієнтованого програмування" [4].

Звіт з будь-якої лабораторної роботи має містити:

1. Титульний лист (відомості про назву навчальної дисципліни; тему лабораторної роботи; ПІБ студента, курс, спеціальність, шифр навчальної групи; ПІБ та посаду викладача).

2. Лист змісту (нумерований перелік назв структурних елементів роботи (див. пункт 3) із зазначенням номерів сторінок).

3. Опис виконаних завдань:

а) умова завдання;

б) опис архітектури програми (структура класів, зв'язки між ними, алгоритми тощо);

в) вихідний код програми;

г) приклади результатів роботи програми на тестових вихідних даних.

4. Висновки з лабораторної роботі з урахуванням усіх завдань, що були виконані: аналіз результатів, які були одержані за кожним завданням; ступінь відповідності програм, що були розроблені, формулюванню завдання; інша інформація.

Приклад оформленого звіту наведено в додатку А.

Оцінювання результатів виконання лабораторної роботи

Результати виконання лабораторної роботи оцінюються за такими параметрами:

1. Якість розробленого програмного продукту.

Оцінюється за ступенем його оригінальності, відповідності вимогам певного завдання роботи та якістю розробленого програмного коду.

2. Виконання календарного плану роботи відповідно до технологічної карти дисципліни.

3. Якість представлення результатів роботи на її захисті.

4. Якість виконання звіту.

Оцінюється за ступенем оригінальності його контенту та відповідності вимогам до змісту й оформлення, викладених у цих методичних рекомендаціях.

Встановлюються таки рівні оцінювання результатів роботи (відповідно до технологічної карти навчальної дисципліни):

Рівень1. До 70 % від максимальної оцінки.

Рівень 2. До 80 % від максимальної оцінки.

Рівень 3. До 90 % від максимальної оцінки.

Рівень 4. До 100 % від максимальної оцінки.

Лабораторна робота 1
Розроблення застосунків з використанням базових елементів ООП

Цілі лабораторної роботи:

1. Набуття практичних навичок використання основних елементів об'єктно-орієнтованого програмування (ООП) під час розроблення програм мовами Java і C#.

2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами розробки.

Перед виконанням лабораторної роботи студент має знати:

1. Поняття класу та об'єкта, співвідношення між ними.

2. Принципи об'єктно-орієнтованого підходу.

3. Синтаксис опису класу.

4. Життєвий цикл об'єкта в програмі.

5. Порядок використання конструкторів.

Після виконання лабораторної роботи студент має вміти самостійно описувати класи мовами Java і C#.

Реалізація базових елементів ООП в мовах Java і C#

Парадигма програмування – це набір теорій, методів, стандартів, які використовуються під час розроблення та реалізації програм на комп'ютері. ООП – одна з парадигм, яка розглядає програму як безліч об'єктів,

що взаємодіють. Кожен об'єкт здатний отримувати повідомлення, обробляти дані та передавати повідомлення іншим об'єктам. Кожен об'єкт можна розглядати як незалежний автомат з окремим призначенням або відповідальністю.

ООП базується на трьох основних принципах:

1. Інкапсуляція – об'єднання в єдине ціле (клас) даних і алгоритмів оброблення цих даних. В ООП дані називаються полями, а алгоритми – методами.

2. Успадкування – можливість створення ієрархії класів, коли класи-нащадки отримують від класу-предка поля і методи.

3. Поліморфізм – властивість класів однієї ієрархії вирішувати схожі за змістом завдання за допомогою різних алгоритмів.

Класи

Класи – це основний спосіб організації даних у Java і C#. Будь-яка програма, яка розробляється з їхнім використанням, має складатися не менше ніж з одного класу.

Функції в Java і C#

Будь-яка функція має бути членом класу (методом). Методи можуть приймати чи не приймати параметри, повертати чи не повертати значення. Параметри можуть передаватися в методи за значенням або за посиланням. Методи можуть бути статичними або методами примірників класу (об'єктів).

Модифікатори

Одним з важливих принципів об'єктно-орієнтованого підходу до програмування є інкапсуляція даних. Для вираження різних рівнів доступності елементів класу в Java і C# існує ряд модифікаторів. Наприклад, поля та методи можуть мати такі основні модифікатори доступу:

public (цей елемент класу доступний усім зовнішнім споживачам);

private (елемент недоступний за межами опису класу).

Крім того, існують модифікатори, що змінюють поведінку елементів класу, наприклад:

const (тільки в C# – означає, що це поле є константою);

final (тільки в Java – означає, що це поле є константою).

static (вказує, що цей елемент належить класу, а не його конкретного екземпляру).

Ключове слово this:

є посиланням на поточний екземпляр об'єкта;

є прихованим покажчиком, передається в кожен нестатичний метод класу;

будь-який метод може використовувати this для доступу до інших нестатичних методів і полях цього об'єкта.

Конструктори:

конструктор – метод, який ініціалізує стан об'єкта після його створення;

конструктор має те саме ім'я, що і клас, у якому він використовується;

конструктор не повертають значення (навіть типу void);

конструктор викликається автоматично;

якщо конструктор не заданий у програмі, то він буде автоматично згенерований компілятором для побудови відповідних об'єктів;

конструктор можна перевантажувати.

Приклад програми на Java

У цій програмі є клас Apple з конструктором, а також клас Program є методом main, у якому створюється об'єкт класу Apple, а потім викликаються його методи (аналогічна програма на C# відрізняється, головним чином, операторами введення-виведення даних):

```
class Apple {
   // Поля
   private String sort, color;
   private double diameter;
   // Конструктор
   public Apple(String sort, String color, double diameter) {
     this.sort = sort;
     this.color = color;
     this.diameter = diameter;
   }
   // ---------- Методи start ----------
   public String getSort() {
     return sort;
   }

   public String getColor() {
      return color;
   }
```

```
    public double getDiameter() {
       return diameter;
     }

    public double calculateVolume() {
      return (Math.PI * Math.pow (diameter, 3) / 6);
    }
      // -------------------- Методи end ---------------------
}

public class Program {
    public static void main (String [ ] args) {
      Apple apl = new Apple ( "Антонівка", "Жовтий", 8.5);
      System.out.println ( "Сорт:" + apl.getSort());
      System.out.println ( "Колір:" + apl.getColor());
      System.out.println ( "Діаметр:" + apl.getDiameter() + '\ n');
      System.out.format ( "Обсяг:% 1 $ 5.3f куб. см.", apl.calculateVolume());
    }
}
```

Масиви об'єктів в Java (C#)

Далі наведено фрагмент програми на C#, у якому створюється масив для зберігання двох об'єктів класу MyClass і два об'єкти цього класу додаються у масив:

```
MyClass [ ] ArrayOfObject = new MyClass [2];
for (int i = 0; i <ArrayOfObject.Length; i ++)
ArrayOfObject [i] = new MyClass();
```

Тут **Length** – це властивість, яка містить довжину масиву (в Java необхідно використовувати властивість **length**).

Порядок виконання лабораторної роботи

Завдання. Розробіть застосунок мовою Java або C#, призначений для автоматизації предметної області відповідно до варіанта завдання.

Для цього необхідно створити два класи: Клас 1 і Клас 2 (конкретні імена Класу 1 і Класу 2 залежать від предметної області за варіантом завдання).

Клас 1 призначений для опису будь-якої з записів відомості. Поля цього класу мають відповідати полям відомості, які призначені для збері-

гання початкових даних. Отримання поточних значень полів, розрахунки за формулами мають виконуватися за допомогою відповідних нестатичних методів цього класу. Для установки значень полів повинен використовуватися конструктор з параметрами.

Клас 2 має містити статичні методи:

метод main() – це точка входу в програму;

необхідну кількість методів для підрахунку підсумкових даних відомості (рівень 4);

метод для виведення всіх даних відомості на консоль.

У методі main() має бути визначений масив об'єктів типу Клас 1 для зберігання відповідних об'єктів.

Застосунок має забезпечувати:

1. Можливість використання текстового меню для вибору варіантів дій користувача (рівні 2 – 4).

2. Введення з консолі кількості записів, що будуть міститися у відомості.

3. У циклі – введення з консолі значень початкових полів кожного запису відомості; створення об'єктів Класу 1, кількість яких відповідає кількості записів відомості; додавання об'єктів Класу 1 в масив.

4. Виведення на консоль початкових і розрахункових даних кожного запису відомості у вигляді:

• необхідної кількості колонок із заголовками (рівні 1, 2);

• справжньої таблиці з горизонтальними і вертикальними лініями сітки (числові значення мають виводитися з певною кількістю знаків після коми з використанням засобів форматного виведення) (рівні 3, 4).

5. Обчислення і виведення на консоль підсумкових даних стовпців відомості (рівень 4).

6. Запобігання появи помилок під час виконання програми (рівень 4).

Варіант 1

Відомість нарахування зарплати співробітникам підприємства:

Прізвище	Зарплата, грн	Утримано, грн	Видано, грн
F	Z	P	S = Z – P
…	…	…	…
Разом	∑	∑	∑

Варіант 2

Відомість витрати палива на автобазах міста:

Автобаза	Витрачено палива, кг	Кількість автомашин, шт.	Середня витрата палива, кг
A	T	K	C = T / K
…	…	…	…
Разом	∑	∑	∑ / n

Варіант 3

Відомість використання машинного часу на обчислювальному центрі:

Кафедра	Використання машинного часу, годин		Відхилення від плану	
	за планом	фактично	у годинах	у %
K	P	F	$O_1 = P - F$	$O_2 = O_1 \times 100 / P$
…	…	…	…	…
Разом	∑	∑		

Варіант 4

Відомість споживання електроенергії на заводах міста:

Завод	Споживання електроенергії, кВт·год		Відхилення від плану	
	за планом	фактично	у кВт·год	у %
Z	P	F	$O_1 = P - F$	$O_2 = O_1 \times 100 / P$
…	…	…	…	…
Разом	∑	∑		

Варіант 5

Відомість руху матеріалів на складі підприємства за звітний період:

Склад	Рух матеріалів за період, грн			Залишок на кінець періоду
	залишок на початок періоду	отримано	видано	
C	Oc	P	V	$R = O_c + P - V$
…	…	…	…	…
Разом	∑	∑	∑	∑

Варіант 6

Відомість прибутку підприємства за звітний період за видами продукції:

Продукція	Кількість, шт.	Оптова ціна, грн	Собівартість, грн	Прибуток, грн
Pr	K	Z	C	P = K (Z – C)
…	…	…	…	…
Разом	∑			∑

Варіант 7

Відомість відвідування занять студентами:

Прізвище	Пропущена, год		Пропуски	
	усього	виправдано	у годинах	у %
F	V	O	P1 = V – O	P2 = P1 × 100 / V
…	…	…	…	…
Разом:	∑	∑	∑	

Варіант 8

Відомість обсягу поставок продукції в натуральному та вартісному вираженні:

Продукція	Шифр	Обсяг поставки, шт.	Оптова ціна, грн	Обсяг поставки, грн
P	H	V	Z	O = V × Z
…	…	…	…	…
Разом		∑	∑	∑

Варіант 9

Відомість розрахунку середньої вартості перевезення авіапасажирів:

Тип літака	Рейс	Витрати на рейс, грн	Кількість пасажирів	Середня вартість перевезення, грн
T	R	Z	K	S = Z / K
…	…	…	…	…
Разом		∑	∑	∑/n

Варіант 10

Відомість обліку часу роботи верстатів підприємства:

Тип верстата	Час роботи, год		Відхилення від плану	
	за планом	фактично	у годинах	у %
Z	P	F	O1 = P – F	O2 = O1 × 100 / P
…	…	…	…	…
Разом	∑	∑		

Варіант 11

Відомість випуску деталей робочими цеху:

Прізвище	Кількість деталей, шт.		Брак	
	виготовлено	прийнято	шт.	%
Z	P	F	O1 = P – F	O2 = O1 × 100 / P
…	…	…	…	…
Разом	∑	∑	∑	

Варіант 12

Відомість наявності та руху основних фондів підприємства:

Назва фонду	Наявність на початок року, шт.	Надійшло, шт.	Вибуло, шт.	Наявність на кінець року, шт.
F	N1	P	V	N2 = N1 + P – V
…	…	…	…	…
Разом	∑	∑	∑	∑

Варіант 13

Відомість обліку запчастин на складі підприємства:

Марка автомобіля	Назва запчастини	Кількість одиниць товару, шт.	Ціна без ПДВ, грн	Ціна з ПДВ, грн
M	N	Q	P	AVP = P + 0,2 × P
…	…	…	…	…
Разом		∑	∑	∑

Варіант 14

Відомість обліку успішності студентів:

Прізвище	Ім'я	Оцінка з першої дисципліні	Оцінка з другої дисципліні	Середня оцінка студента
LN	FN	M1	M2	AM = (M1 + M2) / 2
…	…	…	…	…
Разом		∑ / n	∑ / n	

Варіант 15

Відомість обліку оплати за телефонні розмови:

Прізвище	Нараховано за міські розмови	Нараховано за міжміські розмови	Нараховано за міжнародні розмови	Нараховано всього з ПДВ
LN	C	IC	F	T = (C + IC + F) × 1,2
…	…	…	…	…
Разом	∑	∑	∑	

Контрольні запитання

1. Які принципи об'єктно-орієнтованого програмування є найбільш важливими?
2. Як можна охарактеризувати клас і об'єкт, а також співвідношення між ними?
3. Яким є життєвий цикл об'єкта в програмі?
4. Для чого призначено посилання this та які основні варіанти його використання?
5. Що означає перевантаження методу?
6. Яким є призначення конструктора?
7. Які основні особливості конструктора?

Лабораторна робота 2
Розроблення застосунків з використанням агрегації та композиції

Цілі лабораторної роботи:

1. Набуття практичних навичок використання відносин агрегації та композиції під час розроблення програм мовами Java і C#.

2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами розробки.

Перед виконанням лабораторної роботи студент має знати:

1. Синтаксис агрегації.
2. Синтаксис композиції.

Після виконання лабораторної роботи студент має вміти застосовувати відношення агрегації під час розроблення програм.

Реалізація відносин агрегації та композиції в Java і C#

Між класами або об'єктами, можуть існувати різні логічні відносини.

Часто зустрічається логічним відношенням між об'єктами є асоціація. Асоціація показує, що об'єкти одного класу пов'язані з об'єктами іншого класу. Якщо між двома класами встановлено зв'язок, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Асоціація, що зв'язує два класи, називається бінарною. Часто у процесі моделювання буває важливо вказати, скільки об'єктів може бути пов'язано за допомогою одного екземпляра асоціації. Це число називається кратністю ролі асоціації. Кратність, зазначена на одному з кінців асоціації, позначає, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати дорівнює одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *) і т. д. Асоціації може бути присвоєно ім'я, яке описує природу відносини. Приклад відносини асоціації показаний на рис. 2.1.

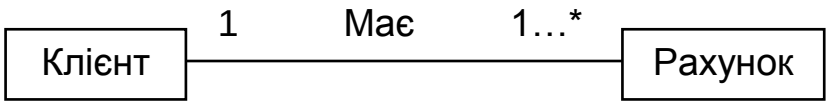


Рис. 2.1. **Відношення асоціації**

Проста асоціація відображає структурне відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і жоден з них не є більш важливим, ніж інший.

Однак дуже часто доводиться моделювати відношення типу "частина/ціле", в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин). Відношення такого типу

називають агрегацією. Воно є окремим випадком асоціації та зображується у вигляді простої асоціації з незафарбованим ромбом з боку "цілого".

Агрегація може означати фізичне входження одного об'єкта в інший, коли об'єкт-ціле не може існувати без об'єктів-частин. Наприклад, літак складається з крил, двигунів, шасі та інших частин. Тоді це сильніший варіант відносини агрегації, який називається композицією. Композиція зображується у вигляді простої асоціації з зафарбовані ромбом з боку "цілого".

Відмінності між композицією і агрегацією

Наприклад, відносини між об'єктами класів "Університет", "Студент" і "Факультет" злегка відрізняються один від одного, хоча обидва є відносинами агрегування. Відношення між об'єктами класів "Університет" і "Факультет" – це композиція, так як у разі знищення об'єкта "Університет" об'єкти факультетів, що належать цьому університету, також мають бути знищені. Об'єкти класів "Студент" і "Університет" перебувають у відношенні агрегації тому, що об'єкт класу "Студент" може існувати після знищення цього університету, так як студент може вчитися в декількох навчальних закладах одночасно.

Агрегація (композиція) в Java і C#

Об'єкт, який є атрибутом іншого об'єкта (агрегату), має зв'язок зі своїм агрегатом. Через цей зв'язок агрегат може посилати йому повідомлення.

Синтаксично агрегація (композиція) – це використання посилання на об'єкт одного класу в якості поля другого класу.

Приклад реалізації відносини агрегації (один об'єкт класу AppleList може містити кілька об'єктів класу Apple) наведено далі (Java). У цьому прикладі об'єкти класу Apple створюються за межами класу AppleList, тобто об'єкти класу Apple можуть існувати і після знищення об'єкта класу AppleList.

```
class AppleList
{
    private Apple[ ] lst;

    //Конструктор
    public AppleList(int n)
    {
```

```
        lst = new Apple[n];
    }

    public void addApple(Apple apl, int index)
    {
        lst[index] = apl;
    }
    private double calculateAverageVolume()
    {
        double sum = 0;
        for (int i = 0; i < lst.length; i++)
            sum += lst[i].calculateVolume();
        return (sum/lst.length);
    }

    public void printTable()
    {
        System.out.println("Цвет\tСорт\tДіаметр\tОб'єм");
        foreach (Apple A in lst)
        {
                System.out.format("%1$\t%2$\t%3$\t%4$10.4f}", A.getSort(),
      A.getColor(), A.getDiameter(), A.calculateVolume());
        }
       System.out.format("\nРазом:\t\t\t%1$6.2f}",  calculateAverageVolume());
    }
}
class Program
{
    public static void main(String[ ] args)
    {
       java.util.Scanner sc = new java.util.Scanner(System.in);
        System.out.print("Введіть кількість яблук: ");
        int n = sc.nextInt();
        AppleList vedomost = new AppleList(n);
        String sort, color;
        double diameter;
       Apple apl;
        for (int i = 0; i < n; i++)
        {
            System.out.print("Введіть сорт яблука: ");
            sort = sc.next();
            System.out.print ("Введіть колір яблука: ");
            color = sc.next();
            System.out.print("Введіть діаметр яблука: ");
            diameter = sc.nextDouble();
            System.out.println();
           apl = new Apple(sort, color, diameter);
            vedomost.addApple(apl, i);
        }
       sc.close();
        System.out.println();
```

```
        vedomost.printTable();
        System.out.println();
    }
}
```

Приклад реалізації відношення композиції (один об'єкт класу AppleList може містити кілька об'єктів класу Apple) наведено далі (C#). У цьому прикладі об'єкти класу Apple створюються всередині класу AppleList, тобто об'єкти класу Apple можуть існувати лише доти, доки існує об'єкт класу AppleList.

```
class AppleList
{
    private Apple [ ] lst;

    // Конструктор
    public AppleList (int n)
    {
        lst = new Apple [n];
    }

    public void AddApple (string sort, string color, double diameter, int index)
    {
        lst [index] = new Apple (sort, color, diameter);
    }
    public double CalculateAverageVolume ()
    {
        double sum = 0;
        for (int i = 0; i <lst.Length; i ++)
            sum + = lst [i] .CalculateVolume ();
        return (sum / lst.Length);
    }

    public void PrintTable ()
    {
        Console.WriteLine ( "Колір\tСорт\tДіаметр\tОб'єм");
        foreach (Apple A in lst)
        {
            Console.WriteLine ( "{0}\t{1}\t{2}\t{3,10: f4}", A.GetSort (), A.GetColor (),
A.GetDiameter (), A.CalculateVolume ());
        }
        Console.WriteLine ( "\nРазом:\t\t\t{0,6: f2}", CalculateAverageVolume ());
```

```
    }
}

class Program
{
    static void Main (string [ ] args)
    {
        Console.Write ( "Введіть кількість яблук:");
        int n = int.Parse (Console.ReadLine ());
        AppleList Vedomost = new AppleList (n);
        string sort, color;
        double diameter;
        for (int i = 0; i <n; i ++)
        {
            Console.Write ( "Введіть сорт яблука:");
            sort = Console.ReadLine ();
            Console.Write ( "Введіть колір яблука:");
            color = Console.ReadLine ();
            Console.Write ( "Введіть діаметр яблука:");
            diameter = double.Parse (Console.ReadLine ());
            Console.WriteLine ();
            Vedomost.AddApple (sort, color, diameter, i);
        }
        Console.WriteLine ();
        Vedomost.PrintTable ();
        Console.WriteLine ();
    }
}
```

Порядок виконання лабораторної роботи

Завдання. Перетворіть застосунок для обробки відомості (див. завдання лабораторної роботи 1 та відповідні варіанти) так, щоб в ньому використовувалося відношення агрегації або композиції, а його функціональні можливості залишилися тими ж.

Застосунок має бути розроблено мовою Java або мовою C#.

Він має містити три класи:

1. Клас "ціле", що описує відомість з безліччю записів.

2. Клас "частина", який описує будь-який запис відомості.

3. Клас, що містить головний метод програми, в якому має створюватися один об'єкт класу "ціле" і з використанням посилання на цей об'єкт викликатися методи класу "ціле".

Контрольні запитання

1. Що означає відношення агрегації?
2. Що означає відношення композиції?
3. Як синтаксично виразити відношення агрегації мовою C# або Java?
4. Як синтаксично виразити відношення композиції мовою C# або Java?

Лабораторна робота 3
Застосування успадкування та поліморфізму

Цілі лабораторної роботи:

1. Удосконалення практичних навичок з розроблення класів мовами C# та Java.
2. Набуття практичних навичок застосування успадкування та поліморфізму в мовах C# і Java.
3. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами розробки.

Перед виконанням лабораторної роботи студент має знати:

1. Синтаксис опису класу.
2. Принципи перевизначення методів.
3. Синтаксис успадкування в C# і Java.
4. Порядок виклику конструкторів при успадкуванні.
5. Принципи реалізації поліморфізму в C# і Java.
6. Основи використання абстрактних класів й інтерфейсів у C# і Java.

Після виконання лабораторної роботи студент повинен вміти застосовувати механізми успадкування та поліморфізму під час розроблення програм мовами C# і Java.

Відношення успадкування

Успадкування – це одно з основних понять об'єктно-орієнтованого програмування. Воно дозволяє одному класу використовувати елементи іншого "спорідненого" класу, не визначаючи їх знову. У такий спосіб програма буде містити менше коду.

Для опису класів, які зв'язані відношенням успадкування, існують певні варіанти термінології:

1. Предок і нащадок.
2. Суперклас і підклас.
3. Базовий клас і похідний клас.

Відношення успадкування має деякі особливості:

1. Похідні класи автоматично успадковують від базового класу все відкриті (public) поля та методи.

2. Через об'єкт похідного класу не можна безпосередньо звернутися до закритих (private) полів, визначених у базовому класі, за їхніми іменами.

3. Конструктори не успадковуються, тому похідний клас має визначати власні конструктори.

4. Після створення об'єкта похідного класу з його конструктора спочатку викликається конструктор базового класу, а тільки після цього виконується код конструктора цього (похідного) класу.

На UML-діаграмі класів відношення успадкування зображується лінією зі стрілкою, яка завжди спрямована на базовий клас. Це показує, що завжди можна стверджувати, що похідний клас, як мінімум, такий самий, як базовий клас. Наприклад, автобус (похідне поняття) – це вид транспортного засобу (базове поняття).

Головні варіанти використання успадкування

Це питання розглянемо з використанням фрагментів програм мовою Java. Реалізація мовою C# є аналогічною.

Варіант 1. B – це A.

```
class A {
  private int field;
  public void f() { }
}
class B extends A { }
```

Цей варіант може використовуватися для клонування базового класу.

Варіант 2. B схожий на A.

```
class A {
```

```
  private int field1;
  public void f() { … }
}
class B extends A {
  private int field2;
  public void g() { … }
}
```

Цей варіант є основним. У даному прикладі об'єкт класу B має два метода: власний метод g та успадкований – f.

Варіант 3. Перевантаження метода базового класу.

```
class A {
  private int field1;
  public void f() { … }
}
class B extends A {
  private int field2;
  public void f(int argument) { … }
}
```

Тут реалізація методу f, яка викликається для об'єкта похідного класу B, залежить від наявності його аргумента. Тобто, метод з одним параметром викликається з класу B, а метод без параметрів – з базового класу A.

Варіант 4. Перевизначення метода базового класу.

```
class A
   private int field1;
   public void f() { //Алгоритм 1 }
}
class B extends A {
   private int field2;
   public void f() { //Алгоритм 2  }
}
```

Цей варіант є основою реалізації принципу поліморфізму. Використовується для динамічного вибору варіанта методу, який буде викликано під час виконання. Рішення залежить від типу посилання та типу об'єкта, яким було ініціалізоване це посилання.

Реалізація принципу поліморфізму

Поліморфізм (polymorphism) (від грецького polymorphos) – це властивість, яка дозволяє одне і те саме ім'я використовувати для вирішення двох або більше схожих, але технічно різних завдань. Метою поліморфізму стосовно об'єктно-орієнтованого програмування є використання одного імені для завдання загальних для класу дій. Виконання кожної конкретної дії визначатиметься типом даних.

Ключовим у розумінні поліморфізму є те, що він дозволяє вам маніпулювати об'єктами різного ступеня складності шляхом створення загального для них стандартного інтерфейсу для реалізації схожих дій.

Поліморфізм, згідно з Г. Бучем, полягає в тому, що імена можуть відповідати різним класам, що є спадкоємцями одного і того ж базового класу. Отже, об'єкт, відмічений таким ім'ям, може по-різному виконувати теж саму дію.

Проілюструємо це на такому прикладі мовою C#:

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

namespace Sample1_1
{
   class Starter
   {
      static void Main(string[ ] args)
      {
         Writer w2 = new TwoWordsWriter();
         Writer w3 = new ThreeWordsWriter();
         // вивод Two words
        w2.Write();
         // вывод Three words
        w3.Write();
      }
   }
   class Writer
   {
      public virtual void Write() { }
   }

   class TwoWordsWriter : Writer
```

```
    {
        public override void Write()
        {
            Console.WriteLine("Two words");
        }
    }

    class ThreeWordsWriter : Writer
    {
        public override void Write()
        {
            Console.WriteLine("Three words");
        }
    }
}
```

У цьому прикладі є три класи: базовий клас OneWordWriter і підкласи TwoWordsWriter та ThreeWordsWriter, кожний з яких має метод Write(). З використанням посилання базового типу Writer кожен з похідних класів викликає свою версію методу Write. Це можливо, тому що цей метод було визначено в базовому класі Writer.

Порядок виконання лабораторної роботи

Необхідно виконати:

для рівнів оцінювання 1 і 2 – завдання 1 мовою Java або C#;

для рівнів оцінювання 3 і 4 – завдання 2 мовою Java або C#.

Також треба розробити UML-діаграму класів, яка відповідає варіанту виконаного завдання (рівні 2 – 4).

Завдання 1. Використовуючи принцип успадкування розробіть програму, яка надає користувачеві функціональність калькулятора, що має два режими роботи: "стандартний" та "інженерний".

У стандартному режимі калькулятор може виконувати будь-яку з чотирьох арифметичних операцій: "+", "-", "*", "/" з двома дійсними числами. У інженерному режимі крім арифметичних операцій він також може виконувати додаткову операцію, яка визначається варіантом завдання (див. табл. нижче). Поточний режим роботи програми вибирає користувач програми.

Програма має забезпечувати:

1. Створення по одному об'єкту класів 1 і 2.

2. Одноразове введення вихідних даних, отримання результату обчислень у вибраному режимі роботи, виведення його на екран (рівень 1).

3. Багаторазове повторення процесу введення початкових даних, отримання результату обчислень у вибраному режимі роботи та виведення його на екран поки користувач не введе певний рядок символів (рівні 2 – 4).

4. Виконання оброблення таких помилкових ситуацій: "поділ на нуль", "вихідні дані для виконання додаткової операції не відповідають обмеженням варіанта завдання" (рівень 4).

Функціональність програми має бути реалізована в трьох класах: **Клас 1**, **Клас 2**, **Клас 3**, які мають знаходитися в різних файлах проєкту.

Рекомендована структура класів програми (метод Pow() призначений для обчислення функції піднесення до степеня; # – protected) наведена на рис. 3.1.

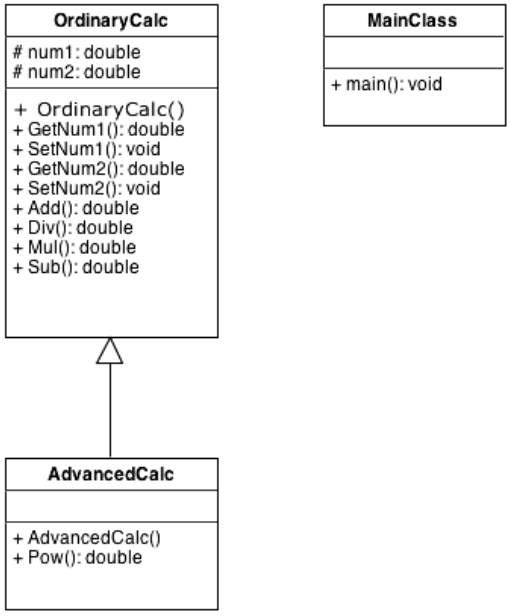


Рис. 3.1. **Рекомендована структура класів (завдання 1)**

Клас 1 є базовим і призначений для опису обчислювального модуля "стандартного" режиму. Поля цього класу мають відповідати початковим даним для виконання арифметичних операцій. Для встановлення значень полів і отримання їхніх поточних значень має використовуватися відповідні властивості (C#) або методи set ... і get ... (Java). Цей клас має

конструктор без параметрів. Виконання кожної з арифметичних операцій має виконуватися за допомогою відповідного нестатичного методу цього класу.

Клас 2 є спадкоємцем **Класу 1** і призначений для опису обчислювального модуля "інженерного" режиму. Цей клас за необхідності має містити одне або кілька полів (для встановлення значень полів і отримання їхніх поточних значень мають використовуватися відповідні властивості (C#) або методи set ... і get ... (Java)). В цьому класі має бути конструктор без параметрів. Додаткова операція "інженерного калькулятора" має бути реалізована у вигляді нестатичного методу даного класу.

Клас 3 містить головний метод програми.

Завдання 2. Перетворіть програму з завдання 1 у такий спосіб, щоб у ній використовувалися абстрактний клас і інтерфейс. Нова програма має працювати також, як програма із завдання 1 за відповідним варіантом. Функціональність програми має бути реалізована в одному абстрактному класі, одному інтерфейсі та трьох звичайних класах.

Абстрактний клас AbstractCalc має поле – модель калькулятора і метод для отримання поточного значення цього поля, а також абстрактні методи для опису операцій обчислювального модуля стандартного режиму, тобто арифметичних операцій.

Клас OrdinaryCalc обчислювального модуля стандартного режиму є спадкоємцем абстрактного класу AbstractCalc і перевизначає його методи. В іншому його сенс той самий, що і в завданні 1. Методи обчислювального модуля стандартного режиму мають викликатися поліморфним способом, тобто через посилання на базовий клас AbstractCalc.

Інтерфейс IAdvanced призначений для опису додаткової операції обчислювального модуля інженерного режиму.

Клас AdvancedCalc обчислювального модуля інженерного режиму є спадкоємцем класу обчислювального модуля стандартного режиму і реалізує інтерфейс IAdvanced. Додаткова операція обчислювального модуля інженерного режиму має викликатися поліморфним способом, тобто через посилання на інтерфейс IAdvanced.

У застосунку також є клас MainClass, у якому знаходиться головний метод.

Рекомендовану структуру класів програми наведено на рис. 3.2.

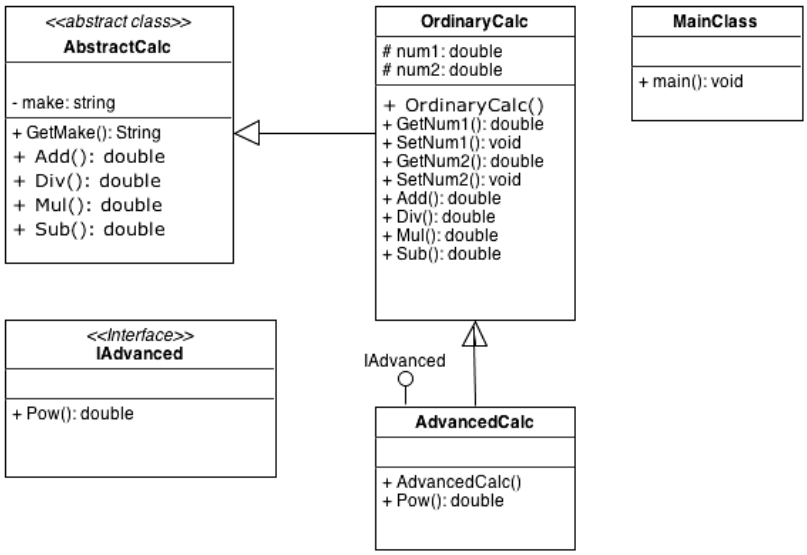


Рис. 3.2. **Рекомендована структура класів (завдання 2)**

Варіанти завдань

№ п/п	Додаткова операція калькулятора
1	$a^{1/2}$, де $1 \le a \le 10000$
2	Перетворення аргументу a(кілометр → миля), де $1 \le a \le 1\ 000$
3	exp(a), де $-5 \le a \le 5$
4	Перетворення аргументу a(Цельсій → Фаренгейт), де мінус $273° \le a \le 273°$
5	lg (a), де $1 \le a \le 1\ 000\ 000$
6	Перетворення аргументу a(байт → килобайт), де $1 \le a \le 10^9$
7	sin(a), де $0° \le a \le 360°$
8	cos(a), де $0° \le a \le 360°$
9	tg(a), де $-90° < a < 90°$
10	Перетворення аргументу a(гривня → євро), де $1 \le a \le 100\ 000$
11	ctg(a), де $-180° < x < 180°$
12	a^3, де $-100 \le a \le 100$
13	ln(a), де $1 \le a \le 100\ 000$
14	Перетворення аргументу a(радіани → градуси), де $0 < a < 6{,}28$
15	Перетворення аргументу a(градуси → радіани), де $0° \le a \le 360°$

Контрольні запитання

1. Які приклади відношення успадкування можна навести?
2. Який синтаксис успадкування в C# і Java?
3. Якою є послідовність виклику конструкторів у спадкуванні?
4. Як перевизначити базовий метод у C# і Java?
5. Яким чином реалізовано принцип поліморфізму в C# і Java?
6. Які є особливості використання абстрактних класів у C# і Java?
7. Які є особливості використання інтерфейсів у C# і Java.

Лабораторна робота 4
Використання шаблонів проєктування

Цілі лабораторної роботи:

1. Надбання практичних навичок застосування шаблонів проєктування під час розроблення програм.

2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами розробки.

Перед виконанням лабораторної роботи студент має знати:

1. Загальні відомості про шаблони проєктування.
2. Класифікацію шаблонів проєктування.
3. Основи використання шаблонів проєктування.

Після виконання лабораторної роботи студент має вміти застосовувати основні шаблони проєктування під час розроблення програм.

Шаблони проєктування

У розробленні програмного забезпечення, шаблон проєктування або патерн (design pattern) – повторювана архітектурна конструкція, що є рішенням проблеми проєктування в рамках деякого контексту, що часто виникає.

Зазвичай шаблон не є закінченим зразком, який може бути прямо перетворений у код; це лише типовий приклад вирішення певних завдань, який можна використовувати в різних ситуаціях. Об'єктно-орієнтовані шаблони показують відносини і взаємодії між класами або об'єктами, без визначення того, які кінцеві класи або об'єкти будуть використовуватися.

Класифікація шаблонів проєктування GoF [3]:

1. Твірні шаблони – відносяться до створення екземплярів класів.

2. Структурні шаблони – відносяться до створення з класів і об'єктів більших структур.

3. Поведінкові шаблони – відносяться до розподілу обов'язків між об'єктами і типовим способам їх взаємодії.

Твірні шаблони, які використовуються найчастіше:

1. Абстрактна фабрика (Abstract Factory) – створює екземпляр декількох сімейств класів.

2. Фабричний метод (Factory Method) – створює екземпляр декількох похідних класів.

3. Одинак (Singleton) – може існувати тільки один екземпляр такого класу.

Приклад використання шаблону "Одинак" наведено на рис. 4.1.

Класи-учасники: **Earth**

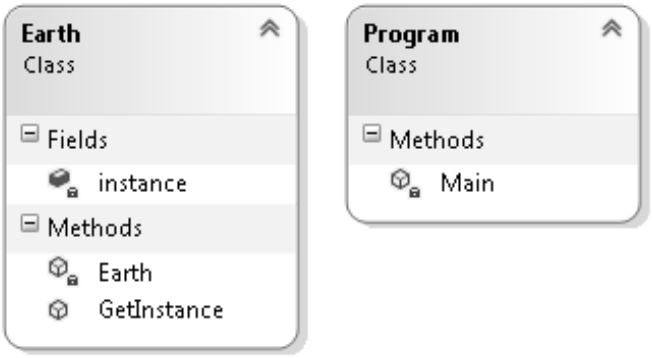


Рис. 4.1. **Діаграма класів програми (шаблон "Одинак")**

Код класів програми мовою C#:

```
class Earth
{
  private static Earth instance;
  private Earth()
  {
    Console.WriteLine("Об'єкт створено...");
  }
  public static Earth GetInstance()
  {
```

```
    if (instance == null)
    {
        instance = new Earth();
      }
      return instance;
  }
}

class Program
{
  static void Main(string[ ] args)
  {
    Earth e1 = Earth.GetInstance();
  }
}
```

Структурні шаблони, які використовуються найчастіше:

1. Адаптер (Adapter) – "стикує" інтерфейси різних класів.

2. Заступник (Proxy) – об'єкт, який контролює доступ до іншого об'єкта, перехоплюючи всі виклики до нього.

3. Компонувальник (Composite) – деревоподібна структура простих і складних об'єктів.

4. Фасад (Facade) – єдиний клас, який представляє всю підсистему.

Діаграму класів шаблону "Заступник" наведено на рис. 4.2.

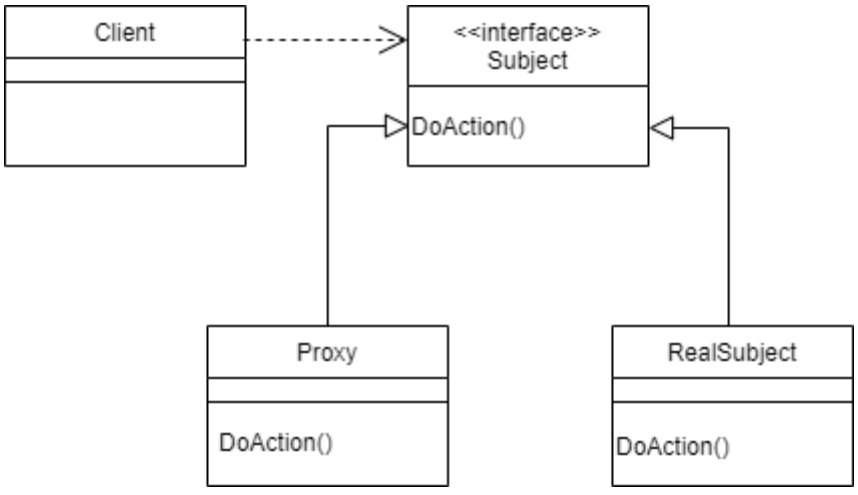


Рис. 4.2. **Шаблон "Заступник"**

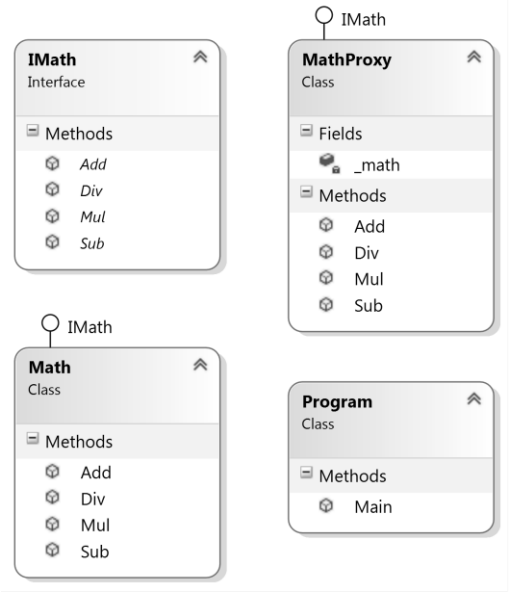
Тут Subject – інтерфейс, у якому є метод DoAction().

RealSubject – клас, який реалізує інтерфейс Subject і перевизначає метод DoAction() для виконання деякої конкретної дії.

Proxy – клас, який також реалізує інтерфейс Subject і перевизначає метод DoAction() у такий спосіб, що в ньому відбувається виклик методу DoAction() класу RealSubject.

Client – клас, який використовує посилання на інтерфейс Subject для створення об'єкта класу Proxy.

Приклад використання шаблону "Заступник" наведено на рис. 4.3.



Add – метод для додавання двох чисел.
Div – метод для ділення двох чисел.
Mul – метод для множення двох чисел.
Sub – метод для віднімання двох чисел.
Поле **_math** – посилання на інтерфейс **IMath**

```
public static void Main(string[ ] args)
{
    // Створення объекта класу-«заступника»
    IMath proxy = new MathProxy();

    Console.WriteLine("4 + 2 = " + proxy.Add(4, 2));
    Console.WriteLine("4 - 2 = " + proxy.Sub(4, 2));
    Console.WriteLine("4 * 2 = " + proxy.Mul(4, 2));
    Console.WriteLine("4 / 2 = " + proxy.Div(4, 2));
}
```

Рис. 4.3. **Використання шаблона "Заступник"**

Класи-учасники (програма "Калькулятор"):

1. Інтерфейс IMath – виконує роль Subject.
2. Клас Math – виконує роль RealSubject.
3. Клас MathProxy – виконує роль Proxy.
4. Клас Program – виконує роль Client.

Порядок виконання лабораторної роботи

Виконати одне з завдань мовою Java або C#. Варіанти індивідуальних завдань взяти із лабораторної роботи 3.

Завдання 1 (рівень 1). Розробіть програму, яка надає користувачеві функціональність калькулятора, що виконує будь-яку з 4 арифметичних операцій: "+", "–", "*", "/" з двома дійсними числами. Програма має забезпечувати багаторазове повторення процесу введення початкових

даних, отримання результату обчислень і виведення його на екран до введення користувачем певного рядка символів.

Структура класів застосунку має відповідати шаблону "Заступник" (див. рис. 4.3).

Завдання 2 (рівень 2). Розробіть програму, яка надає користувачеві функціональність калькулятора, що виконує будь-яку з 4 арифметичних операцій: "+", "–", "*", "/" з двома дійсними числами, а також додаткову операцію в відповідність до варіанта завдання. Програма має забезпечувати багаторазове повторення процесу введення вихідних даних, отримання результату обчислень і виведення його на екран до введення користувачем певного рядка символів.

Структура класів застосунку має відповідати шаблону "Заступник" (див. рис. 4.3). Клас MathProxy має бути "Одинаком" (див. рис. 4.1).

Завдання 3 (рівні 3 і 4). Розробити програму, яка надає користувачеві функціональність калькулятора, що виконує будь-яку з 4 арифметичних операцій: "+", "–", "*", "/" з двома дійсними числами, а також додаткову операцію в відповідність до варіанта завдання. Програма має забезпечувати багаторазове повторення процесу введення вихідних даних, отримання результату обчислень і виведення його на екран до введення користувачем певного рядка символів. У програмі необхідно передбачити оброблення таких помилкових ситуацій:

1. "Ділення на нуль" – з використанням відповідного бібліотечного класу винятку.

2. "Вихідні дані для виконання додаткової операції не відповідають обмеженням варіанта завдання" – з використанням користувацького класу винятку (рівень 4).

Структура класів застосунку має відповідати шаблону "Заступник". Вона в основному аналогічна структурі класів, яка наведена на рис. 4.3, за винятком того, що методи для обчислення арифметичних операцій і обчислення додаткової операції необхідно розподілити між двома інтерфейсами, які реалізуються в класах Math і MathProxy. Клас MathProxy має бути "Одинаком" (рис. 4.1).

Контрольні запитання

1. Що таке "шаблон проєктування"?
2. Які види шаблонів проєктування GoF існують?

3. Як застосувати деякий шаблон проєктування до власної програми?

4. Для чого використовують шаблон "Одинак"?

5. Скільки класів беруть участь у шаблоні "Одинак"?

6. Для чого використовують шаблон "Заступник"?

7. Скільки класів та інтерфейсів беруть участь у шаблоні "Заступник"?

Лабораторна робота 5
Використання основних бібліотек Microsoft .NET і Java SE

Цілі лабораторної роботи:

1. Удосконалення практичних навичок з розроблення класів.

2. Надбання практичних навичок використання класів винятків, введення-виведення, колекцій.

3. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами розробки.

Перед виконанням лабораторної роботи студент має знати організацію системи введення-виведення в Microsoft .NET і Java SE.

1. Принципи оброблення винятків у Microsoft .NET і Java SE

2. Основні структури даних бібліотеки колекцій платформи Microsoft .NET і Java SE та принципи їхнього використання.

Після виконання лабораторної роботи студент має вміти використовувати основні бібліотеки Microsoft .NET і Java SE під час розроблення програм.

Колекції в Microsoft .NET і Java SE

Під колекцією розуміють групу об'єктів. Колекції спрощують програмування, пропонуючи вже готові рішення для побудови структур даних, розроблення яких "з нуля" відрізняється великою трудомісткістю. Мова йде про убудовані колекції, які підтримують, наприклад, функціонування стеків, черг і хеш-таблиць.

Основна перевага колекцій полягає в тому, що вони стандартизують спосіб оброблення груп об'єктів у прикладних програмах. Усі колекції

реалізують певні інтерфейси, які задають модель поведінки тієї чи іншої колекції. Більшість убудованих реалізацій таких інтерфейсів, таких як класи ArrayList, Hashtable, Stack і Queue, можна використовувати "як є". У кожного програміста також є можливість реалізувати власну колекцію, але в більшості випадків вистачає вбудованих.

Колекції можуть бути нетипізованими або типізованими.

Нетипізовані колекції оперують даними базового типу System.Object у Microsoft .NET (java.lang.Object у Java SE). Отже, вони можуть зберігати дані будь-якого вбудованого або користувальницького типу, тому що всі вони є похідними від System.Object або java.lang.Object. В одній колекції припустимо мати дані різних типів, зокрема не зв'язаних між собою відношенням успадкування.

Типізовані колекції можуть зберігати тільки таки елементи, які сумісні за типом з певною колекцією. Це можуть бути елементи вбудованих типів Microsoft .NET або Java SE, а також будь-якого користувальницького типу, такого як MyClass (тут MyClass – клас власного розроблення). В одній типізованій колекції припустимі дані тільки певного типу або його нащадків.

Колекції на платформі Microsoft .NET

Головними елементами бібліотеки колекцій Microsoft .NET є інтерфейси та класи, що їх реалізують.

Інтерфейси забезпечують уніфікацію поведінки колекцій.

Класи – це переважно структури даних, а саме списки, черги, стеки, множини, словники (асоціативні масиви) тощо.

Простір імен System.Collections містить класи й інтерфейси, які визначають різні нетипізовані колекції об'єктів. Основні з них наведено в табл. 5.1.

Таблиця 5.1

Основні класи та інтерфейси колекцій

Клас або інтерфейс	Опис
1	2
ICollection	Інтерфейс. Визначає розмір, перелічувачі та методи синхронізації для всіх нетипізованих колекцій

Закінчення табл. 5.1

1	2
IComparer	Інтерфейс. Надає метод для порівняння двох об'єктів
IDictionary	Інтерфейс. Визначає поведінку нетипізованих колекцій пар "ключ – значення"
IEnumerable	Інтерфейс. Надає перелічувач, яке підтримує просту ітерацію у нетипізованих колекціях
IList	Інтерфейс. Представляє нетипізовану колекцію об'єктів, до яких можна отримати індивідуальний доступ за допомогою індексу
ArrayList	Реалізує інтерфейс IList, використовуючи масив, розмір якого за потреби динамічна збільшується
Hashtable	Представляє набір пар ключ/значення, які організовані на основі хеш-коду ключа
Queue	Надає нетипізовану колекцію об'єктів, що обслуговується за принципом "першим прийшов – першим вийшов" (FIFO)
SortedList	Представляє набір пар ключ/значення, які відсортовані за ключами та доступні за ключем і за індексом
Stack	Надає нетипізовану колекцію об'єктів, що обслуговується за принципом "останнім прийшов – першим вийшов" (LIFO)

Простір імен System.Collections.Generic містить інтерфейси та класи, що визначають узагальнені колекції. Вони дозволяють створювати типізовані колекції, що забезпечують підвищену продуктивність і безпеку типів.

Багато типізованих колекцій є прямими аналогами нетипізованих, а деякі з них мають однакові назви. Наприклад, колекція Dictionary <TKey, TValue> – це узагальнена версія Hashtable. List <T> – це типізована версія ArrayList. Класи Queue <T> і Stack <T> є аналогами нетипізованих колекцій Queue та Stack відповідно.

Також є декілька колекції, які не мають аналогів та існують тільки в типізованому варіанті (табл. 5.2).

Таблиця 5.2

Колекції, що існують тільки в типізованому варіанті

Клас колекції	Опис
HashSet<T>	Представляє множину унікальних елементів на базі хеш-таблиці
LinkedList<T>	Представляє двозв'язний список
SortedDictionary<K, V>	Представляє набір пар ключ / значення, на базі червоно-чорного дерева, які відсортовані за ключами
SortedSet<T>	Представляє множину унікальних елементів на базі червоно-чорного дерева

Колекції на платформі Java SE

Java-колекції також, як і в C#, можуть бути нетипізованими або типізованими. Однак один і той же клас може представляти як нетипізовану так і типізовану колекцію. Якщо під час створення колекції вказується тип даних, що будуть у ній зберігатися, це типізована колекція. Інакше – нетипізована. Елементи бібліотеки колекцій містяться в пакеті java.util. Її головні компоненти – це інтерфейси та класи. Інтерфейси визначають поведінку, притаманну всім класам колекцій, які реалізують ці інтерфейси. Класи бібліотеки колекцій, головним чином, є конкретними реалізаціями таких структур даних, як динамічний масив, список, словник тощо. Ієрархію інтерфейсів бібліотеки колекцій наведено на рис. 5.1.

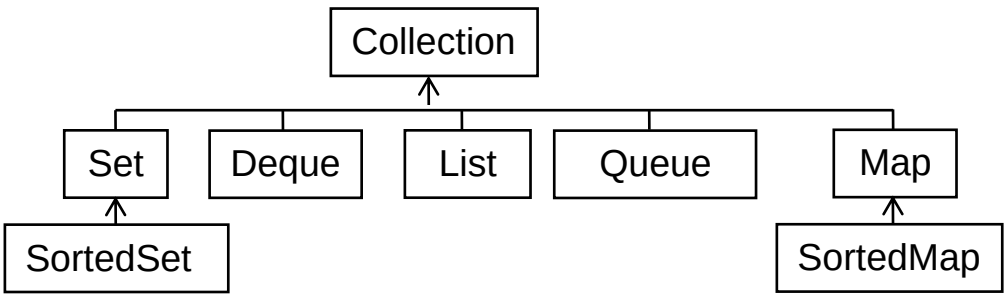


Рис. 5.1. **Інтерфейси бібліотеки колекцій Java SE**

Коротка характеристика інтерфейсів бібліотеки колекцій Java SE:
Collection – кореневий інтерфейс;
Set – множина (однакові елементи неприпустимі);
SortedSet – множина, елементи якої впорядковані за зростанням;

List – список;
Queue, Dequeue – черги;
Map – словник або асоціативний масив;
SortedMap – словник, ключі якого впорядковані за зростанням.

Основні класи реалізації наведено в табл. 5.3.

Таблиця 5.3

Основні класи бібліотеки колекцій Java SE

Ім'я класу	Опис
ArrayList	Динамічний масив, що реалізує інтерфейс List
ArrayDeque	Динамічний масив, що реалізує інтерфейси Queue та Dequeue. Клас можна використовувати в режимі черги або стеку
LinkedList	Двозв'язний список
HashSet	Множина на базі хеш-таблиці
TreeSet	Множина на базі червоно-чорного дерева
HashMap	Словник на базі хеш-таблиці
TreeMap	Словник на базі червоно-чорного дерева

Введення-виведення даних у Microsoft .NET і Java SE

Програми мовами C# і Java виконують операції введення-виведення за допомогою потоків, які побудовані на певній ієрархії класів. Потік (stream) – це абстракція, яка генерує та приймає дані. За допомогою потоку можна читати дані з різних джерел (клавіатура, файл) і записувати в різні джерела (принтер, екран, файл). Попри те, що потоки зв'язуються з різними фізичними пристроями, характер поведінки всіх потоків однаковий. Тому класи та методи введення-виведення можна застосувати до багатьох типів пристроїв.

На найнижчому рівні ієрархії потоків введення-виведення знаходяться потоки, які оперують байтами. Це пояснюється тим, що багато пристроїв у виконанні операцій введення-виведення орієнтовані на байти.

Проте для людини більш звично оперувати символами. Тому розроблені символьні потоки, які фактично є оболонками, що виконують перетворення байтових потоків у символьні, та навпаки.

Потоки працюють з джерелами та споживачами даних, роль яких можуть виконувати консоль, файл, оперативна пам'ять і мережеве з'єднання. Центральну частину потокової системи Microsoft .NET займає клас Stream простору імен System.IO. Клас Stream – це байтовий потік, який є базовим для решти всіх потокових класів. Від класу Stream успадковують такі байтові класи потоків, як: *FileStream* – байтовий потік, розроблений для файлового введення-виведення; *BufferedStream* – укладає в оболонку байтовий потік і додає буферизацію, яка у багатьох випадках збільшує продуктивність програми; *MemoryStream* – байтовий потік, який використовує пам'ять для зберігання даних.

Шляхом успадкування програміст може створити власні потокові класи. Проте, для переважної більшості застосунків достатньо вбудованих потоків. Часто використовуються клас FileStream, класи StreamWriter і StreamReader, що є оболонками для класу FileStream і дозволяють перетворювати байтові потоки в символьні, а також класи BinaryWriter і BinaryReader, що є оболонками для класу FileStream і дозволяють перетворювати байтові потоки в двійкові для роботи з int-, double-, short- й іншими подібними типами даних.

Основну інфраструктуру вводу-виводу в Java SE визначено в пакеті java.io. Java також має два типи потоків: байтові та символьні. Байтові потоки забезпечують зручні засоби для обробки вхідних і вихідних даних у вигляді байтів. Символьні потоки призначено для оброблення вхідних і вихідних даних у вигляді символів. Вони використовують Unicode і тому можуть бути інтернаціоналізовані.

Байтові потоки визначаються за допомогою двох базових абстрактних класів: InputStream і OutputStream. Нащадки InputStream – це потоки вводу даних, а нащадки OutputStream – потоки виводу даних. Важливі класи байтових потоків наведено в табл. 5.4.

Таблиця 5.4

Класи байтових потоків у Java SE

Клас	Опис
1	2
InputStream	Абстрактний клас для опису потока вводу
OutputStream	Абстрактний клас для опису потока виводу
BufferedInputStream	Потік вводу з підтримкою буферизації

Закінчення табл. 5.4

1	2
BufferedOutputStream	Потік виводу з підтримкою буферизації
DataInputStream	Призначено для читання даних "примітивних" типів
DataOutputStream	Призначено для запису даних "примітивних" типів
FileInputStream	Потік вводу для читання даних із файла
FileOutputStream	Потік виводу для запису даних у файл
PrintStream	Додає функціональність до іншого вихідного потоку, а саме можливість зручного друку значень даних різних типів з використанням методів print, println, printf та format

Ці класи мають низку методів. Найбільш важливими з них є read і write, які призначено, відповідно, для читання або запису байтів.

Символьні потоки визначають за допомогою двох базових абстрактних класів: Reader і Writer. Нащадки Reader – це потоки вводу даних, а нащадки Writer – потоки виводу даних. Важливі класи символьних потоків наведено в табл. 5.5.

Таблиця 5.5

Класи символьних потоків у Java SE

Класи	Опис
Reader	Абстрактний клас для опису потока вводу
Writer	Абстрактний клас для опису потока виводу
BufferedReader	Потік вводу з підтримкою буферизації
BufferedWriter	Потік виводу з підтримкою буферизації
FileReader	Потік вводу для читання даних із файла
FileWriter	Потік виводу для запису даних у файл
InputStreamReader	Потік вводу, який транслює байти в символи
OutputStreamWriter	Потік виводу, який транслює символи в байти
PrintWriter	Потік виводу, що друкує відформатовані подання об'єктів до потоку текстового виводу

Порядок виконання лабораторної роботи

Завдання. *За основу рекомендується взяти програму зі свого варіанту з лабораторної роботи 2, у якій використовується відношення агрегації або композиції.*

Розробіть програму мовою Java або C# для оброблення відомості, дані якої зберігаються в текстовому файлі, де кожен запис має перебувати на новому рядку. Поля кожного запису розділені роздільником відповідно до варіанта завдання, наприклад, комою:

Іван, Петренко, 8,7,11.

Для виконання завдання необхідно створити:

"Клас 1", призначений для опису будь-якого з записів відомості. В цьому класі мають бути поля, відповідні полям відомості, які призначені для зберігання початкових даних; конструктор з параметрами; необхідні нестатичні методи і властивості;

"Клас 2" містить певну колекцію для зберігання даних відомості, а також статичні методи, призначені для додавання даних в колекцію, виведення даних відомості з колекції в текстовий файл, введення їх у колекцію з цього файлу (рівні 3 і 4), виведення даних відомості з колекції на консоль;

"Клас 3" – клас, що містить головний метод програми.

Кожен з класів має знаходитися в окремому файлі (рівні 3 і 4).

Програма має забезпечувати:

1. Вибір варіантів дій користувача за допомогою текстового меню (рівні 2, 3 і 4).

2. Введення з консолі початкових даних кожного із записів відомості в поточному сеансі роботи.

3. Створення для кожного запису відомості об'єкта "Класу 1"; подальше додавання кожного такого об'єкта в колекцію, яка відповідає варіанту завдання.

4. Виведення даних з колекції, відповідної варіанту завдання, в текстовий файл, ім'я якого вводиться з консолі, з використанням відповідного методу "Класу 2".

5. Введення даних відомості з текстового файлу в колекцію об'єктів "Класу 1", тип якої визначається варіантом завдання, з використанням відповідного методу "Класу 2" (рівні 3 і 4)

6. Виведення початкових і розрахункових даних кожного запису відомості з колекції, а також підсумкової інформації відомості на консоль у вигляді необхідної кількості колонок із заголовками (рівні 1 і 2) або у вигляді справжньої таблиці даних, з горизонтальними і вертикальними лініями сітки з використанням засобів форматного виведення (рівні 3 і 4).

7. Обробку винятків введення-виведення IOException (рівень 4) під час виконання операцій з файлом.

Далі наведено рекомендовану діаграму класів програми на прикладі відомості обліку яблук (рис. 5.2).

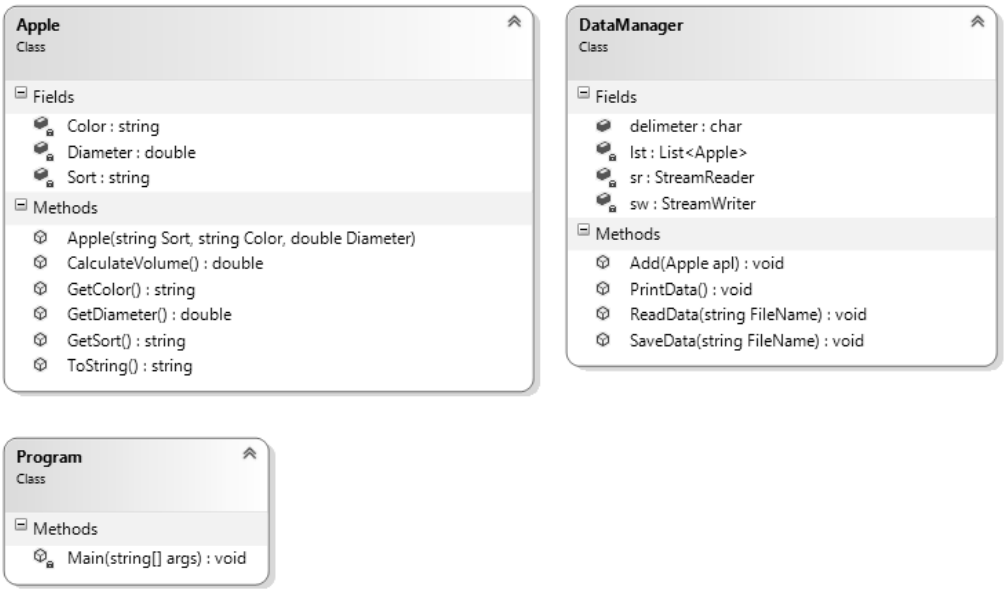


Рис. 5.2. **Рекомендована діаграма класів**

Варіанти завдань:

а) варіанти класів колекцій і символів-роздільників:

№ п/п	Символ – роздільник полів записів файлу	Клас колекції Microsoft. NET	Клас колекції Java SE
1	;	Hashtable	HashSet<T>
2	:	ArrayList	HashMap<K, V>
3	|	SortedList	ArrayList<T>
4	/	List<T>	TreeSet<T>
5	\	LinkedList<T>	TreeMap<K, V>
6	$	HashSet<T>	LinkedList<T>
7	&	Dictionary<K, V>	LinkedHashSet<T>
8	!	SortedDictionary <K, V >	ArrayList
9	#	SortedList<K, V>	HashSet<T>
10	%	Hashtable	HashMap<K, V>
11	+	SortedSet<T>	LinkedHashMap <K, V>
12	пробіл	SortedList	TreeSet<T>
13	-	HashSet<T>	TreeMap<K, V>
14	*	Dictionary<K, V>	LinkedList<T>
15	@	SortedList<K, V>	LinkedHashSet<T>

б) варіанти предметних областей (див. варіанти завдань з лабораторної роботи 1).

Контрольні запитання

1. Яким є склад бібліотеки контейнерів Java SE або Microsoft. NET?
2. Які основні інтерфейси існують у бібліотеці контейнерів Java SE або Microsoft .NET?
3. Які реалізації структури даних "список" є в Java SE або Microsoft .NET?
4. Які реалізації структур даних "черга" та "стек" є в Java SE або Microsoft .NET?
5. Які реалізації структури даних "множина" є в Java SE або Microsoft .NET?
6. Які реалізації структури даних "словник" існують в Java SE або Microsoft .NET?
7. Що таке "хеш-таблиця" і "хеш-функція"?
8. Які базові класи байтових і символьних потоків введення-виведення є в Java SE або Microsoft .NET?
9. Для чого призначено символьні потоки введення-виведення в Java SE або Microsoft .NET?
10. Для чого призначено байтові потоки введення-виведення в Java SE та Microsoft .NET?

Лабораторна робота 6
Використання регулярних виразів

Цілі лабораторної роботи:

1. Придбання практичних навичок використання регулярних виразів для оброблення текстової інформації в програмах мовами C# і Java.
2. Удосконалення навичок роботи з рекомендованими інтегрованими середовищами розробки.

Перед виконанням лабораторної роботи студент має знати принципи створення і використання регулярних виразів.

Після виконання лабораторної роботи студент повинен вміти самостійно використовувати механізм регулярних виразів під час розроблення програм розбору текстів і пошуку за зразком мовами C# і Java.

Регулярні вирази

Регулярні вирази – формальна мова для пошуку та здійснення маніпуляцій з підрядками в тексті, заснований на використанні метасимволів. По суті це рядок-зразок, що складається з символів і метасимволів та задає правило пошуку.

Оскільки з використанням регулярних виразів виконується не просто пошук буквальних фрагментів тексту, певні символи призначено для спеціального використання. Їх часто називають "метасимволами".

Метасимволи регулярних виразів наведені далі:

< ([{ \ ^ – = $! |] }) ? × + . >

Якщо ви хочете використовувати будь-який із цих символів як літерал у регулярному виразі, вам потрібно "екранувати" їх зворотнім слешем. Наприклад, якщо необхідно шукати текстові послідовності 1 + 1 = 2, правильним регулярним виразом для цього є 1 \ + 1 = 2.

Зверніть увагу, що 1 + 1 = 2 без зворотного слешу також є дійсним регулярним виразом. Тож ви не отримаєте повідомлення про помилку. Але, знайдені послідовності – це не 1 + 1 = 2. Це може бути 111 = 2 або 111111 = 2 і таке інше, тому що тут плюс є квантором (відповідає одному або більшої кількості входжень попереднього символу або групи). Якщо забути "екранувати" метасимвол там, де його використання заборонено, наприклад, в + 1, ви отримаєте повідомлення про помилку.

Основні символи та послідовності, використовувані в регулярних виразах, наведено в табл. 6.1.

Таблиця 6.1

Символи, використовувані в регулярних виразах

Символ	Опис
1	2
^	Відповідність має виявлятися на початку вхідного рядка
$	Відповідність має виявлятися в кінці вхідного рядка
*****	Квантор (відповідає 0 або більше входжень попереднього символу або групи)
+	Квантор (відповідає 1 або більше входжень попереднього символу або групи)

Закінчення табл. 6.1

1	2
?	Квантор (відповідає 0 або 1 входження попереднього символу або групи)
.	Клас символів (відповідає будь-якому символу, за винятком символу '\n')
[aeiou]	Клас символів (відповідає будь-якому символу з множини, заданої у квадратних дужках)
[^aiou]	Клас символів (відповідає будь-якому символу, за винятком символів, заданих у квадратних дужках)
[0-9] **[a-z]**	Клас символів (завдання одного символу з діапазону)
\w	Клас символів (один із символів, які використовуються під час завдання ідентифікаторів)
\s	Клас символів (відповідає символу пробілу)
\d	Клас символів (відповідає будь-якому символу з множини цифр)
\W	Клас символів (будь-який з символів, крім використовуваних під час завдання ідентифікаторів)
\S	Клас символів (будь-яке непорожнє місце)
\D	Клас символів (не цифра)
{n}	Квантор (попередній елемент повторюється рівно n раз)
{n,}	Квантор (попередній елемент повторюється мінімум n раз)
{n,m}	Квантор (Попередній елемент повторюється мінімум n раз, але не більше, ніж m разів)

Головний клас для оброблення регулярних виразів на платформі Microsoft. NET – це **System.Text.RegularExpressions.Regex**, на платформі Java SE – **java.util.regex.Pattern**.

Найпростішим прикладом використання класів Regex і Pattern є перевірка користувальницького вводу, яка просто повідомляє, чи задовольняє вхідна послідовність символів регулярному виразу.

Приклад 1. Фрагмент програми на C# для пошуку телефонних номерів формату **(0XX)XXX-XX-XX**, де X – будь-яка цифра:

```
Console.WriteLine("Введіть номер телефона:");
string str = Console.ReadLine();
 string pattern = @"^\(0\d{2}\)[1-9](\d{2}-){2}\d{2}$";
Regex reg = new Regex(pattern);
if (reg.IsMatch(str))
  Console.WriteLine("Номер записано правильно");
```

```
  else
  Console.WriteLine("Неправильний формат номера!!!");
```

Приклад 2. Фрагмент програми на Java для пошуку телефонних номерів формату **(0XX)XXX-XX-XX**, де X – будь-яка цифра:

```
Scanner sc = new Scanner(System.in);
System.out.println("Введите номер телефона");
String str = sc.next();
String pattern = "^\\(0\\d{2}\\)[1-9](\\d{2}-){2}\\d{2}$";
if (Pattern.matches(pattern, str)) {
   System.out.println("Номер записано правильно \n");
} else {
    System.out.println("Неправильний формат номера!!!\n");
  }
```

Загальні відомості про синтаксичні аналізатори

Синтаксичний аналіз – це процес аналізу вхідної послідовності символів з метою розбору її граматичної структури за заданою формальною граматикою. Зокрема, синтаксичний аналіз дозволяє визначити, чи належить вхідна послідовність символів до заданої граматики (наприклад, до граматики деякої мови).

Синтаксичний аналізатор – це програма, яка виконує синтаксичний аналіз.

Під час опису граматик будемо використовувати лінгвістичні формули (розширена форма Бекуса – Наура). Це формальна система визначення синтаксису, в якій одні синтаксичні категорії послідовно визначаються через інші.

Головні елементи, які використовуються для запису металінгвістичних формул у відповідності із розширеною формою Бекуса – Наура:

::= (означає "це");

{} (фігурні дужки використовуються для об'єднання декількох елементів);

{}* (вказує, що вкладений у фігурні дужки елемент формули може бути опущений або повторений довільне число раз);

{}+ (вказує, що вкладений у фігурні дужки елемент формули може бути повторений один або довільне число раз);

{}? (вказує, що вкладений у фігурні дужки елемент формули може бути повторений один раз або він може бути відсутнім);

| (означає "або").

Порядок виконання лабораторної роботи

Необхідно виконати:

- для рівня 1 – завдання 1 мовою Java або C#;
- для рівня 2 – завдання 2 мовою Java або C#;
- для рівня 3 і 4 – завдання 1 та завдання 2 мовою Java або C#.

Завдання 1. Розробіть консольну програму для виконання аналізу вхідної послідовності символів, що вводяться користувачем, з метою визначення її приналежності до граматики, яка описує деяке поняття (згідно з варіантом завдання).

Програма має містити блок синтаксичного аналізу вхідної послідовності символів на базі регулярних виразів.

Якщо вхідна послідовність припустима, то програма має видавати відповідне повідомлення користувачу, інакше – у аналогічний спосіб повідомляти його про неприпустимість вхідної послідовності.

У будь-якому випадку далі виконання програми має тривати до введення користувачем рядка-ознаки завершення програми.

Варіант 1

Розробіть синтаксичний аналізатор для поняття "**умовний_ оператор**":

умовний_оператор::= if (умова) оператор {else оператор}?
оператор::={змінна++;} | {змінна--;}
умова::= змінна {< | >} змінна
змінна::=літера$^{+}$цифра*
буква::a|b|c
літера ::= 0|1|2|3|4|5|6|7|8|9

Варіант 2

Розробіть синтаксичний аналізатор для поняття "**новий_ об'єкт**":

новий_ об'єкт::= {ім'я_класу змінна=new конструктор;}|{new конструктор;}
конструктор:: = ім'я_класу ({змінна {, змінна }*}*)
ім'я_класу::= літера$^{+}$
змінна::= літера цифра
літера::d|e|f
цифра ::= 1|2|3

Варіант 3

Розробіть синтаксичний аналізатор для поняття "**просте_логічне**":

просте_логічне ::= TRUE | FALSE | {простий_ідентифікатор
знак_операции простий_ідентифікатор }
простий_ідентифікатор::= літера
знак_операції ::= AND | OR
літера::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Варіант 4

Розробіть синтаксичний аналізатор для поняття "**відношення**":

відношення::= операнд знак_ відношення операнд
знак_ відношення::= $<|>|=$
операнд ::= константа $|$змінна
константа ::= цифра$\{$цифра$\}^*$
змінна ::= літера $\{$ літера $|$цифра$\}^*$
цифра ::= 0|1|2|3|4|5|6|7|8|9
літера::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Варіант 5

Розробіть синтаксичний аналізатор для поняття "**присвоювання**":

присвоювання::= {лева_частина=права_частина}
лева_частина ::= змінна
права_частина::= змінна |{ змінна знак_операції змінна }
змінна::= літера { літера | цифра}*
знак_операции ::= $+|-|*|/$
цифра ::= 0|1|2|3|4|5|6|7|8|9
літера::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Варіант 6

Розробіть синтаксичний аналізатор для поняття "**список_параметрів**":

список_параметрів ::= параметр {,параметр}*
параметр ::= {ім'я = цифра$^+$}| { ім'я = літера$^+$}
ім'я::= літера$^+$
цифра ::= 0|1|2|3|4|5|6|7|8|9
літера::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Варіант 7

Розробіть синтаксичний аналізатор для поняття "**скалярний_тип**":

скалярний_тип ::= {ім'я_константи {, ім'я_константи}*}
| {константа . . константа}
Ім'я_константи ::= літера {цифра | літера }*
константа ::= цифра {цифра}*
цифра ::= 0|1|2|3|4|5|6|7|8|9
літера::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Варіант 8

Розробіть синтаксичний аналізатор для поняття "**дійсне_число**":

дійсне _число ::= {ціле_число**.** ціле_без_знака}
| { ціле_число. ціле_без_знака E ціле _число}
| { ціле_число E ціле_число}
ціле_число ::= {+ | - }? ціле_без_знака
ціле_без_знака ::= цифра {цифра}*
цифра ::= 0|1|2|3|4|5|6|7|8|9

Варіант 9

Розробіть синтаксичний аналізатор для поняття "**опис_міток**":

опис_міток ::= LABEL метка {, метка}*;
метка ::= ідентифікатор | послідовність_цифр
послідовність_цифр ::= 0 . . 9999
ідентифікатор::= літера { літера | цифра}*
цифра ::= 0|1|2|3|4|5|6|7|8|9
літера::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Варіант 10

Розробіть синтаксичний аналізатор для поняття "**арифметична дія**":

арифметична дія ::= ціле {знак_операції ціле }+
ціле ::= цифра{цифра}*
знак_операції ::= * | / | + | -
цифра ::= 0|1|2|3|4|5|6|7|8|9

Варіант 11

Розробіть синтаксичний аналізатор для поняття **"оператор_циклу"**:

оператор_циклу ::= while (умова) оператор_присвоювання
умова::= переменная {< | >} цифра$^{+}$
оператор_ присвоювання::= {змінна=цифра$^{+}$;}
змінна := літера$^{+}$
цифра ::= 1|2|3|4|5
літера::= a|b|c|d

Варіант 12

Розробіть синтаксичний аналізатор для поняття "**функція**":

функція ::= тип_повертання ім'я список_параметрів тіло_функції
тип_ повертання::= void
ім'я ::= літера { літера | цифра}*
список_параметрів ::= {()} | {(параметр{, параметр}*)}
тіло_функції ::= ліва_дужка пустий_оператор права_дужка
пустий_оператор ::= ;
ліва_ дужка::= {
права_ дужка::= }
літера::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
цифра ::= 0|1|2|3|4|5|6|7|8|9

Варіант 13

Розробіть синтаксичний аналізатор для поняття "**текст**":

текст::=слово{,слово}*
слово::= корень{суфікс}*
корень::= сад | мед
суфікс::= ик | ок

Варіант 14

Розробіть синтаксичний аналізатор для поняття "**список_списків**":

список_списків ::= список {; список}*
список ::= елемент{, елемент }*
елемент ::= літера
літера::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

Варіант 15

Розробіть синтаксичний аналізатор для поняття "**послідовність_символов**":

послідовність_символов ::= елемент {роздільник елемент}*
елемент ::= {символ символ*}
символ ::= літера | цифра
літера ::= а|б|в|…|я
цифра ::= 0|1|2|3|4|5|6|7|8|9
роздільник::= {, | ;}

Завдання 2. Розробіть програму, яка знаходить усі номери телефонів припустимих форматів у довільній послідовності символів. Програма має містити блок синтаксичного аналізу вхідної послідовності на базі регулярних виразів. Послідовність символів для аналізу необхідно ввести з консолі. Виведіть на консоль початкову послідовність символів, всі знайдені номери телефонів та їхню загальна кількість. Якщо жоден номер телефону не знайдений, необхідно вивести на консоль відповідне повідомлення. Програма має виконуватися циклічно до введення користувачем рядка-ознаки завершення програми.

Варіанти цього завдання наведені далі.

Номер варіанта	Припустимі формати номерів, де x – цифра від 1 до 9	
1	(+81xx)xx-xxx-xxx	(xxx)xx-xxx-xxx
2	(+380xx)xxx-xx-xx	xxx-xx-xx
3	(0572)x-xx-xxx	x-xx-xxx
4	(+1xxx) xx-xxx-xx	(xxx) xx-xxx-xx
5	(+86) xxxx-xxxx-xxx	xxxx-xxxx-xxx
6	(xxx)xx-xx-xx	xx-xx-xx
7	(+55xx) xx-xxx-xxx	(xx) xx-xxx-xxx
8	(+244) xxx-xxx-xxx	xxx-xxx-xxx
9	(044)xxxxx-xx	xxxxx-xx
10	(+61) xx-xxx-xxxx	xx-xxx-xxxx
11	(+226) xx-xx-xx-xx	xx-xx-xx-xx
12	(061)xxxxxxx	xx-xxx-xx
13	(+850) xxxx-xxx-xxx	xxxx-xxx-xxxx
14	(+27) xx-xxx-xxxx	xxx-xxx-xxxx
15	(031)x-xxxx-xx	x-xxxx-xx

Контрольні запитання

1. Коли доцільно використовувати регулярні вирази?
2. Який простір імен або пакет необхідно підключити для можливості використання регулярних виразів у Microsoft .NET або Java SE відповідно?
3. Який клас Microsoft .NET або Java SE представляє компільований регулярний вираз?
4. Які основні операції підтримуються класом регулярних виразів?
5. Для чого використовуються символи ^ та $ під час визначення регулярного виразу?

Рекомендована література

1. Блинов И. Н. Java. Методы программирования : учеб.-метод. пособ. / И. Н. Блинов, В. С. Романчик. – Минск : Изд-во "Четыре четверти", 2013. – 768 с.
2. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р. Максимчук, М. Энгл и др. ; пер. с англ. – Москва : ИД "Вильямс", 2008. – 720 с.
3. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – Санкт-Петербург: Питер, 2007. – 366 с.
4. Щербаков О. В. Основи об'єктно-орієнтованого програмування : навч. посіб. / О. В. Щербаков, Ю. Е. Парфьонов, В. М. Федорченко. – Харків : ХНЕУ ім. С. Кузнеця, 2019. – 237 с.
5. Michaelis M. Essential C# 6.0 / M. Michaelis, E. Lippert. – Boston : Addison-Wesly, 2016. – 1004 p.

Інформаційні ресурси

6. Парфьонов Ю. Е. Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Основи об'єктно-орієнтованого програмування" [Електронний ресурс] / Ю. Е. Парфьонов. – Режим доступу : http://www.ikt.hneu.edu.ua/course/view.php?id=879.

7. Парфьонов Ю. Е. Основи об'єктно-орієнтованого програмування : презентації лекцій [Електронний ресурс] / Ю. Е. Парфьонов. – Режим доступу : http://www.ikt.hneu.edu.ua/course/view.php?id=879.

8. Уроки Java для начинающих [Электронный ресурс]. – Режим доступа : http://cybern.ru/category/java/begin-java.

9. Programming Tutorials and Source Code Examples [Electronic resource]. – Access mode : http://www.java2s.com.

10. Design Patterns [Electronic resource]. – Access mode : www.oodesign.com.

11. Design Patterns [Electronic resource]. – Access mode : http://sourcemaking.com/design_patterns.

12. .NET Design Patterns [Electronic resource]. – Access mode : https://www.dofactory.com/net/design-patterns.

13. The premier Website about regular expressions [Electronic resource]. – Access mode : http://www.regular-expressions.info/.

Додатки

Додаток А

Приклад оформленого звіту з лабораторної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНИВЕРСИТЕТ
ІМЕНІ СЕМЕНА КУЗНЕЦА

КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ

"Об'єктно-орієнтоване програмування"

Звіт

з лабораторної роботи № 1

на тему: "Основи використання мов C# та Java"

Виконав:
студент 2 курсу
групи 6.04.122.010.19.1
факультету ІТ Пет-
ренко І. С.

Перевірив:
доцент кафедри ІС
к.т.н., с.н.с. Новіков П. М.

м. Харків – 2021 рік

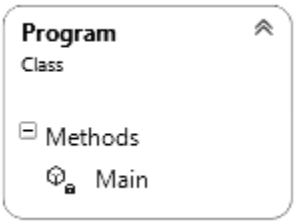
Зміст

Завдання 1

Умова завдання

Розробити програму мовою C# для знаходження суми двох цілих чисел.

Опис архітектури програми

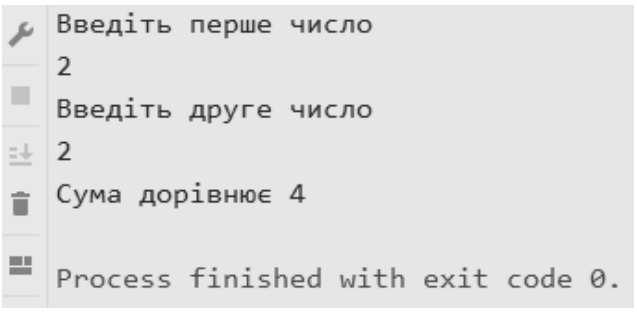


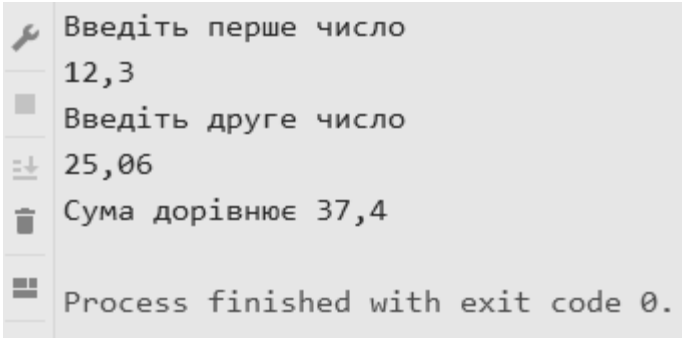
Program – головний клас

Вихідний код програми

```
using System;
namespace Summa
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Введіть перше число");
         double a = double.Parse(Console.ReadLine());
         Console.WriteLine("Введіть друге число");
         double b = double.Parse(Console.ReadLine());
         double c = a + b;
         Console.WriteLine("Сума дорівнює {0:###.#}", c);
       }
   }
}
```

Приклади результатів роботи програми





Висновки

У ході виконання лабораторної роботи 1 я отримав практичні навички зі створення простих консольних програм з використанням основних елементів мови C#, а також середовища розробки JetBrains Rider. Розроблені програми головним чином відповідають постановці завдання. Результати роботи програми на тестових вихідних даних відповідають очікуваним.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Методичні рекомендації до лабораторних робіт

для студентів спеціальностей 122 "Комп'ютерні науки", 126 "Інформаційні системи та технології" першого (бакалаврського) рівня

Самостійне електронне текстове мережеве видання

Укладач **Парфьонов** Юрій Едуардович

Відповідальний за видання *І. О. Ушакова*

Редактор *А. С. Ширініна*

Коректор *В. Ю. Труш*

План 2021 р. Поз. № 101 ЕВ. Обсяг 57 с.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***