### Кам’янець-Подільський національний університет імені Івана Огієнка

### КАФЕДРА КОМП’ЮТЕРНИХ НАУК

### Навчальна дисципліна «Об’єктно-орієнтоване програмування»

### ЛАБОРАТОРНА РОБОТА #0104

## Тема:

**Поліморфізм**

Варіант №1

**Виконав**:  
студент 1-го курсу  
групи KNms1-B24  
Білик Я. Ю.

**Прийняв**:  
старший викладач,  
Слободянюк О. В.

### Кам’янець-Подільський – 2025

1. **Короткі теоретичні відомості.**

Поліморфізм – наступний з трьох принципів ООП. Його застосування дозволяє створювати у похідних класах методи, які мають той самий інтерфейс, що й методи базового класу, проте відмінну функціональність.

Придивимось більш уважно до віртуальних функцій. З цією метою розглянемо досить загальний клас Base з віртуальним методом DoSomething(), який просто надсилає повідомляє про себе. У похідному класі SubBase визначимо відповідний метод із модифікатором override. Створимо також похідний клас наступного рівня спадкування SubSubBase. Крім того створимо зовнішню функцію fun(Base sb), яка приймає аргумент типу Base. Всередині цієї функції міститься звертання до методу DoSomething(). Оскільки функція fun може бути викликана як для екземпляру класу Base, так і для екземплярів класів SubBase та SubSubBase, рішення про те, який саме метод DoSomething() (базового чи одного з похідних класів) буде викликаний, неможливо прийняти у момент компіляції програми, а лише під час виконання. Цей механізм називається *динамічним поліморфізмом* або *динамічною диспетчеризацією методів*. (Відповідний механізм в мові С++ називають *пізнім зв’язуванням*.)

Можлива ситуація, коли у базовому класі неможливо реалізувати деякий метод. Проте зрозуміло, що подібний метод зі своєю особливою функціональністю буде присутній у похідних класах. В такому випадку є сенс визначити «порожній» віртуальний метод у базовому класі (свого роду «заглушку»), визначивши його інтерфейс, а у похідних класах слушним чином визначити коди відповідних методів, що його заміщають. Засіб, який примушує всі похідні класи обов’язково замістити такий «невизначений» метод, полягає у використанні модифікатору abstract:

Абстрактний метод, таким чином, взагалі не має тіла. Абстрактними можуть бути не лише методи, але й властивості. Клас, який містить принаймні один абстрактний член класу, теж має бути визначений як abstract, адже від нього не можна утворювати екземпляри – такі класи використовуються лише для спадкування. Класичним прикладом демонстрації абстрактного класу є клас геометричних форм – не існує загальної формули для визначення площі абстрактної геометричної фігури. Проте всім відомо як визначити площі, наприклад, квадрата або прямокутника чи кола.

1. **Повні умови завдань.**

Задача 1

Створити додаток, який задовольняє вимогам, наведеним в завданні. Успадкування застосовувати тільки в тих завданнях, в яких це логічно обґрунтоване. Аргументувати приналежність класу кожного створюваного методу і коректно перевизначити для кожного класу методи equals(), hashCode(), toString().

Створити об'єкт класу Текст, використовуючи класи Речення, Слово. Методи: доповнити текст, вивести на консоль текст, заголовок тексту.

Задача 2

У кожному варіанті завдання при описі класів самостійно визначити необхідні поля, властивості та методи вводу/виводу. Деякі методи класу-предка повинні бути віртуальними і абстрактними. У програмі-клієнті для збереження сукупності об’єктів використати масив.

Створити клас TPrism, який представляє правильну призму і містить методи для знаходження площі поверхні та об’єму. На основі цього класу створити класи- нащадки TPrism3 та TPrism4, які представляють правильну трикутну та чотирикутну призми. З клавіатури вводиться дані для створення правильної трикутної та чотирикутної призми. На їх основі поступово створити *m* правильних призм (трикутних та чотирикутних), об’єм кожної з яких на 5 більше попередньої. Для трикутних призм знайти сумарний об’єм, а для чотирикутних – суму площ поверхні.

1. **Лістинги програм.**

Задача 1

using System;

using System.Collections.Generic;

using System.Linq;

class Word

{

public string Value { get; set; }

public Word(string value)

{

Value = value;

}

public override bool Equals(object obj)

{

if (obj is Word other)

{

return this.Value.Equals(other.Value, StringComparison.OrdinalIgnoreCase);

}

return false;

}

public override int GetHashCode()

{

return Value.ToLower().GetHashCode();

}

public override string ToString()

{

return Value;

}

}

class Sentence

{

public List<Word> Words { get; set; }

public Sentence()

{

Words = new List<Word>();

}

public void AddWord(Word word)

{

Words.Add(word);

}

public override bool Equals(object obj)

{

if (obj is Sentence other)

{

return Words.SequenceEqual(other.Words);

}

return false;

}

public override int GetHashCode()

{

return string.Join(" ", Words).GetHashCode();

}

public override string ToString()

{

return string.Join(" ", Words);

}

}

class Text

{

public string Title { get; set; }

public List<Sentence> Sentences { get; set; }

public Text(string title)

{

Title = title;

Sentences = new List<Sentence>();

}

public void AddSentence(Sentence sentence)

{

Sentences.Add(sentence);

}

public void PrintText()

{

Console.WriteLine($"Заголовок: {Title}");

foreach (var sentence in Sentences)

{

Console.WriteLine(sentence.ToString());

}

}

public override bool Equals(object obj)

{

if (obj is Text other)

{

return this.Title.Equals(other.Title, StringComparison.OrdinalIgnoreCase) &&

Sentences.SequenceEqual(other.Sentences);

}

return false;

}

public override int GetHashCode()

{

return Title.GetHashCode() ^ Sentences.Aggregate(0, (acc, sentence) => acc ^ sentence.GetHashCode());

}

public override string ToString()

{

return $"{Title}\n" + string.Join("\n", Sentences);

}

}

class Program

{

static void Main()

{

// Створюємо об'єкти класу Word

var word1 = new Word("Це");

var word2 = new Word("приклад");

var word3 = new Word("тексту");

// Створюємо речення і додаємо до нього слова

var sentence = new Sentence();

sentence.AddWord(word1);

sentence.AddWord(word2);

sentence.AddWord(word3);

// Створюємо текст, додаємо речення

var text = new Text("Заголовок тексту");

text.AddSentence(sentence);

// Друкуємо текст

text.PrintText();

// Перевірка методів equals() і hashCode()

var word4 = new Word("Це");

Console.WriteLine(word1.Equals(word4)); // true

Console.WriteLine(word1.GetHashCode() == word4.GetHashCode()); // true

var text2 = new Text("Заголовок тексту");

text2.AddSentence(sentence);

Console.WriteLine(text.Equals(text2)); // true

Console.WriteLine(text.GetHashCode() == text2.GetHashCode()); // true

}

}

**Задача 2**

using System;

abstract class TPrism

{

protected double height;

public double Height

{

get => height;

set => height = value;

}

public TPrism(double height)

{

this.height = height;

}

public abstract double GetSurfaceArea();

public abstract double GetVolume();

}

class TPrism3 : TPrism

{

private double side;

public double Side => side;

public TPrism3(double side, double height) : base(height)

{

this.side = side;

}

public override double GetSurfaceArea()

{

double baseArea = (Math.Sqrt(3) / 4) \* side \* side;

double perimeter = 3 \* side;

return 2 \* baseArea + perimeter \* height;

}

public override double GetVolume()

{

double baseArea = (Math.Sqrt(3) / 4) \* side \* side;

return baseArea \* height;

}

}

class TPrism4 : TPrism

{

private double side;

public double Side => side;

public TPrism4(double side, double height) : base(height)

{

this.side = side;

}

public override double GetSurfaceArea()

{

double baseArea = side \* side;

double perimeter = 4 \* side;

return 2 \* baseArea + perimeter \* height;

}

public override double GetVolume()

{

double baseArea = side \* side;

return baseArea \* height;

}

}

class Program

{

static void Main()

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

Console.InputEncoding = System.Text.Encoding.UTF8;

Console.Write("Введіть сторону та висоту для трикутної призми (через пробіл): ");

string[] input1 = Console.ReadLine().Split();

double side3 = double.Parse(input1[0]);

double height3 = double.Parse(input1[1]);

TPrism3 baseTriPrism = new TPrism3(side3, height3);

Console.Write("Введіть сторону та висоту для чотирикутної призми (через пробіл): ");

string[] input2 = Console.ReadLine().Split();

double side4 = double.Parse(input2[0]);

double height4 = double.Parse(input2[1]);

TPrism4 baseQuadPrism = new TPrism4(side4, height4);

Console.Write("Скільки призм створити? ");

int m = int.Parse(Console.ReadLine());

TPrism[] prisms = new TPrism[m];

double volume = Math.Min(baseTriPrism.GetVolume(), baseQuadPrism.GetVolume());

for (int i = 0; i < m; i++)

{

if (i % 2 == 0)

{

prisms[i] = AdjustHeight(baseTriPrism, volume);

}

else

{

prisms[i] = AdjustHeight(baseQuadPrism, volume);

}

volume += 5;

}

double totalVolumeTri = 0;

double totalAreaQuad = 0;

foreach (var prism in prisms)

{

Console.WriteLine($"{prism} -> Об'єм = {prism.GetVolume():F2}, Площа поверхні = {prism.GetSurfaceArea():F2}");

if (prism is TPrism3)

totalVolumeTri += prism.GetVolume();

else if (prism is TPrism4)

totalAreaQuad += prism.GetSurfaceArea();

}

Console.WriteLine($"\nСумарний об'єм трикутних призм: {totalVolumeTri:F2}");

Console.WriteLine($"Сумарна площа поверхні чотирикутних призм: {totalAreaQuad:F2}");

}

static TPrism AdjustHeight(TPrism prism, double targetVolume)

{

if (prism is TPrism3 p3)

{

double baseArea = (Math.Sqrt(3) / 4) \* p3.Side \* p3.Side;

double newHeight = targetVolume / baseArea;

return new TPrism3(p3.Side, newHeight);

}

else if (prism is TPrism4 p4)

{

double baseArea = p4.Side \* p4.Side;

double newHeight = targetVolume / baseArea;

return new TPrism4(p4.Side, newHeight);

}

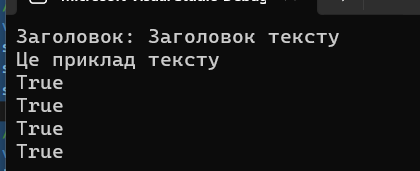
return prism;

}

}

1. **Результати роботи програм.**

Задача 1



Задача 2

