**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**Кафедра інтелектуальних технологій**

**ЛАБОРАТОРНА РОБОТА №3**

з дисципліни «Об'єктно-орієнтоване програмування»

Тема роботи: «Поліморфізм. Абстрактні класи.»

Варіант №10

Виконала студентка

групи АнД-21

Радоманова С.П.

Перевірила:

Москаленко Н.В.

**Київ-2024**

Варіант 10:Базовий клас Series (прогресія) з методом для визначення і-го члену прогресії та суми елементів. Похідні класи для арифметичної та

геометричної прогресії з власними методами для визначення і-го члену

прогресії та суми елементів.

Оскільки під час виклику в функції main ми звертались безпосередньо до об’єктів “підкласів” Arithmetic та Geometric на виводі отримуємо інформацію щодо екземпляру базового класу (текстові строчки) та щодо екземплярів класів-нащадків. Працює просте перекриття (статичний поліморфізм).

Змінюємо в базовому класі розрахункові методи на віртуальні. Це дозволяє дочірнім класам заміщати ці методи. Описуємо три покажчики на базовий клас, для них створюємо екземпляри, викликаємо GetInfo(). В результаті бачимо, що завдяки віртуальним методам при виклику цих методів через вказівник на базовий клас викликається версія методу з дочірнього класу.

Заміщення (динамічний поліморфізм) є істинним поліморфізмом, оскільки він дозволяє в момент виконання визначити, який метод викликати, на основі типу об'єкта, а не на основі типу вказівника або змінної. Воно необхідне, коли ви використовуєте вказівники або змінні базового типу і хочете, щоб методи дочірніх класів викликалися динамічно під час виконання (в залежності від того, який об'єкт фактично створений).

Просте перекриття (статичний поліморфізм) підходить, коли тип об'єкта відомий на етапі компіляції, і немає потреби в гнучкості.

Описуємо функцію void foo в якості параметра якої передається базовий клас (передаємо параметр за значенням). Як відомо, в такому випадку створюється копія об'єкта. Коли ми передаємо об'єкт похідного класу через копіювання в параметрі функції використовується копія об'єкта базового класу Series, що означає, що методи похідних класів не викликаються, замість цього викликані методи базового класу.

Спробуємо передати в функцію foo посилання/покажчик на базовий клас. Бачимо, що в такому разі копія об’єкту не створюється. Такий підхід дозволяє динамічно викликати методи похідних класів.

Принцип підстановки Лисковговорить, що об'єкти похідного класу повинні бути взаємозамінними з об'єктами базового класу без порушення коректності програми. Тобто, де б у коді не використовувався об'єкт базового класу, ви повинні мати можливість замінити його об'єктом похідного класу, і програма повинна працювати правильно, без зміни її поведінки, розроблені в цій програмі класи відповідають цьому принципу.

Чистий віртуальний метод — це метод, який оголошується, але не визначається в базовому класі. Відредагуємо програму зазначивши розрахункові методи в базовому класі як чисті віртуальні. Тепер базовий клас стає абстрактним, що унеможливлює створення об’єктів типу Series базового класу, тож відкоментуємо всі помилкові фрагменти.

В головній програмі створюємо масив покажчиків на базовий клас.

Коли викликаємо метод getInfo() для кожного з елементів масиву через покажчик на базовий клас, фактично викликаються реалізації цього методу з класів-нащадків завдяки динамічному поліморфізму, що підтверджує правильність реалізації поліморфізму.

Після додавання звичайних деструкторів до кожного з класів помітно, що видаляється тільки частина об'єкта, що належить базовому класу, а частина класу-нащадка не видаляється.

Віртуальний деструктор у базовому класі гарантує, що коли ми видаляємо об'єкт через покажчик на базовий клас, викликаються також деструктори класів-нащадків.

Під час виконання завдання 6 (виконання на мові java) враховуємо такі особливості:

**Чисті віртуальні методи** у C++ стають **абстрактними методами** в Java. **Деструктори** у Java не використовуються, краще покладатися на автоматичне управління пам'яттю.

Для випадкового створення екземплярів класів-нащадків використовується **клас Random**.

Поліморфізм працює через базовий клас Series, що дозволяє викликати перевизначені методи calculateTerm() та calculateSum() з об'єктів класів-нащадків.

### **Опис структури класу Series**

**Клас:** Series  
**Опис:** Абстрактний базовий клас для представлення математичних прогресій.

**Поля класу:**

* protected double startNumber — перший член прогресії.
* protected double increase — різниця для арифметичної прогресії або знаменник для геометричної прогресії.

**Методи класу:**

* Series(double startNumber, double increase) — конструктор з параметрами, що ініціалізує перший член прогресії та різницю або знаменник.
* abstract void calculateTerm(int i) — абстрактний метод для обчислення i-го члена прогресії, який буде перевизначений в класах-нащадках.
* abstract void calculateSum(int n) — абстрактний метод для обчислення суми перших n членів прогресії.
* void getInfo(int i, int n) — метод, що викликає calculateTerm() та calculateSum() для виведення інформації про прогресію.

### **Опис структури класу Arithmetic**

**Клас:** Arithmetic  
**Опис:** Клас для представлення арифметичної прогресії, що наслідує базовий клас Series.

**Поля класу:**Наслідує поля від класу Series

**Методи класу:**

* Arithmetic(double startNumber, double increase) — конструктор з параметрами, що ініціалізує перший член та різницю арифметичної прогресії.
* void calculateTerm(int i) — перевизначений метод для обчислення i-го члена арифметичної прогресії за формулою: startNumber + (i - 1) \* increase.
* void calculateSum(int n) — перевизначений метод для обчислення суми перших n членів арифметичної прогресії за формулою: (n / 2.0) \* (2 \* startNumber + (n - 1) \* increase).

### **Опис структури класу Geometric**

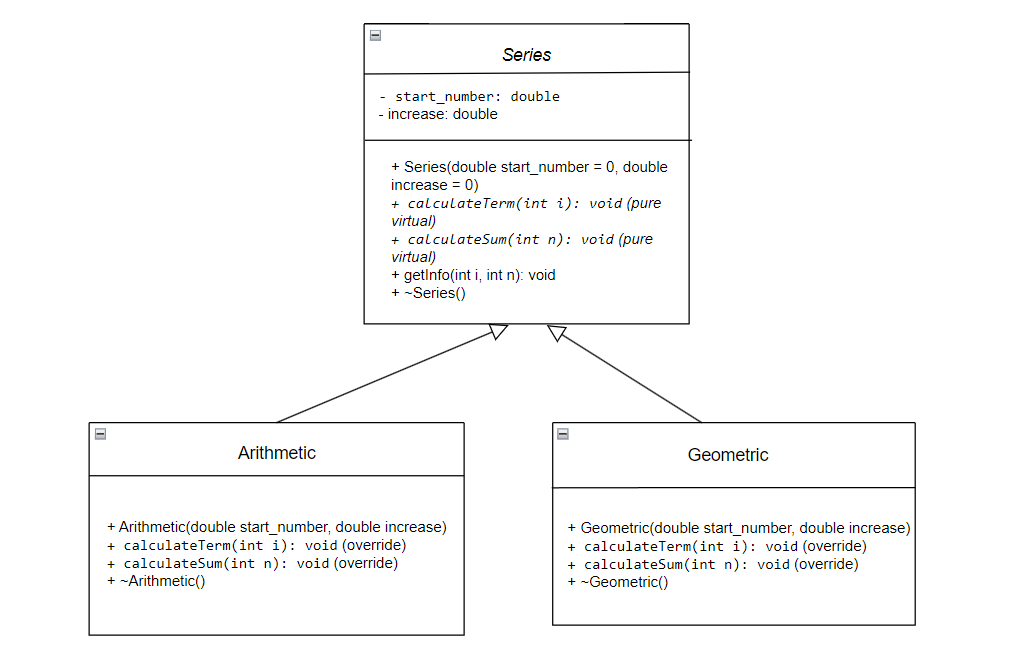
**Клас:** Geometric  
**Опис:** Клас для представлення геометричної прогресії, що наслідує базовий клас Series.

**Поля класу:**Наслідує поля від класу Series

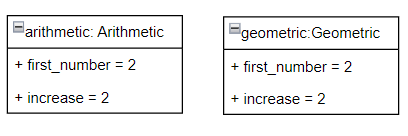
**Методи класу:**

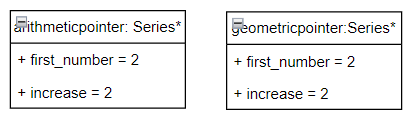
* Geometric(double startNumber, double increase) — конструктор з параметрами, що ініціалізує перший член та знаменник геометричної прогресії.
* void calculateTerm(int i) — перевизначений метод для обчислення i-го члена геометричної прогресії за формулою: startNumber \* Math.pow(increase, i - 1).
* void calculateSum(int n) — перевизначений метод для обчислення суми перших n членів геометричної прогресії. Якщо знаменник прогресії дорівнює 1, сума обчислюється як: startNumber \* n. Якщо не дорівнює 1, сума обчислюється за формулою: startNumber \* (1 - Math.pow(increase, n)) / (1 - increase).

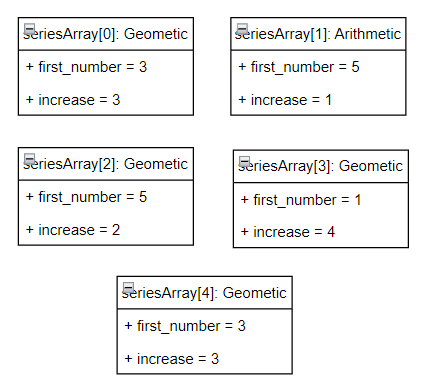
UML діаграма класів:



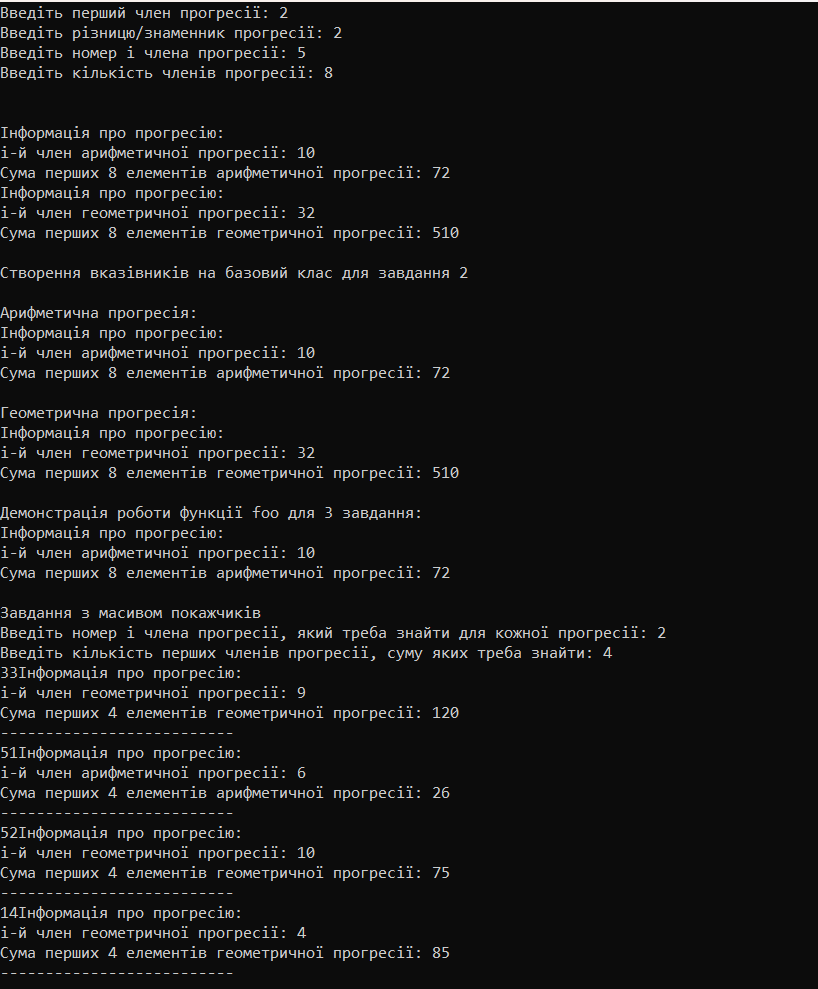
опис екземплярів, які було створено під час тестування:

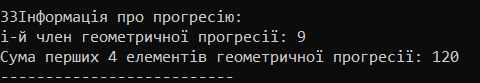




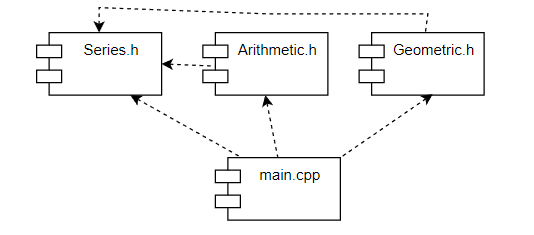


Тестовий приклад:

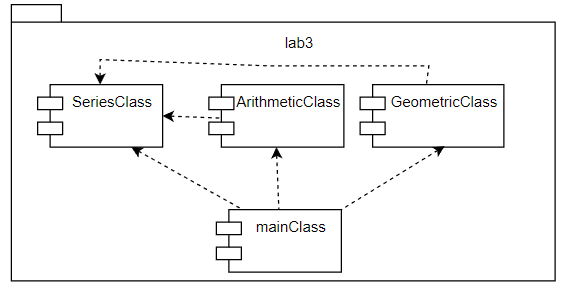




Опис структури програми у вигляді UML-діаграми компонентів мовою С++:



мовою Java:



**Текст програми мовою С++:**

Series.h:

#pragma once

class Series {

public:

double start\_number; // Перший член прогресії

double increase; // Різниця або знаменник прогресії

public:

// Конструктор з параметрами

Series(double start\_number = 0, double increase = 0) {

this->start\_number = start\_number;

this->increase = increase;

}

// Чистий віртуальний метод для обчислення i-го члена прогресії

virtual void calculateTerm(int i) = 0;

/\*{

std::cout << "Метод визначення i-го члена в базовому класі Series.\n";

}\*/

// Чистий віртуальний метод для обчислення суми прогресії

virtual void calculateSum(int n) = 0;

/\* {

std::cout << "Метод визначення суми в базовому класі Series.\n";

}\*/

// Метод GetInfo для виведення інформації

void getInfo(int i, int n) {

std::cout << "Інформація про прогресію:\n";

calculateTerm(i);

calculateSum(n);

}

virtual ~Series() {

std::cout << "Спрацював деструктор для базового класу \n";

};

};

Arithmetic.h:

#pragma once

class Arithmetic : public Series {

public:

// Конструктор з параметрами

Arithmetic(double start\_number, double increase) : Series(start\_number, increase) {}

// Перекриття методу для визначення i-го члена

void calculateTerm(int i) {

double new\_number = start\_number + (i - 1) \* increase;

std::cout << "i-й член арифметичної прогресії: " << new\_number << "\n";

}

// Перекриття методу для визначення суми прогресії

void calculateSum(int n) {

double sum = (n / 2.0) \* (2 \* start\_number + (n - 1) \* increase);

std::cout << "Сума перших " << n << " елементів арифметичної прогресії: " << sum << "\n";

}

~Arithmetic()override {

std::cout << "Спрацював деструктор для арифметичної прогресії \n";

};

};

Geometric.h:

#pragma once

class Geometric : public Series {

public:

// Конструктор з параметрами

Geometric(double start\_number, double increase) : Series(start\_number, increase) {}

// Перекриття методу для визначення i-го члена

void calculateTerm(int i) override {

double new\_number = start\_number \* pow(increase, i - 1);

std::cout << "i-й член геометричної прогресії: " << new\_number << "\n";

}

// Перекриття методу для визначення суми прогресії

void calculateSum(int n) override {

double sum;

if (increase == 1) {

sum = start\_number \* n;

}

else {

sum = start\_number \* (1 - pow(increase, n)) / (1 - increase);

}

std::cout << "Сума перших " << n << " елементів геометричної прогресії: " << sum << "\n";

}

~Geometric()override {

std::cout << "Спрацював деструктор для геометричної прогресії \n";

};

};

main.cpp:

#include <iostream>

#include <cmath>

#include <cstdlib> // для rand() і srand()

#include <ctime> // для time()

#include "Series.h"

#include "Arithmetic.h"

#include "Geometric.h"

void foo(Series& obj, int i, int n)

{

obj.getInfo(i, n);

}

int main()

{

double first\_number;

std::cout << "Введіть перший член прогресії: ";

std::cin >> first\_number;

double increase;

std::cout << "Введіть різницю/знаменник прогресії: ";

std::cin >> increase;

int i;

std::cout << "Введіть номер i члена прогресії: ";

std::cin >> i;

int n;

std::cout << "Введіть кількість членів прогресії: ";

std::cin >> n;

std::cout << '\n';

// Series defaultSeries(first\_number, increase);

Arithmetic arithmetic(first\_number, increase);

std::cout << '\n';

Geometric geometric(first\_number, increase);

// defaultSeries.getInfo(i, n);

arithmetic.getInfo(i, n);

geometric.getInfo(i, n);

std::cout << '\n';

std::cout << "Створення вказівників на базовий клас для завдання 2\n";

// Створюємо вказівники на базовий клас і екземпляри дочірніх класів

// Series\* defaultSeriespointer = new Series(first\_number, increase);

Series\* arithmeticpointer = new Arithmetic(first\_number, increase);

Series\* geometricpointer = new Geometric(first\_number, increase);

// Викликаємо GetInfo для кожного екземпляра

// std::cout << "\nБазовий клас Series:\n";

// defaultSeriespointer->getInfo(i, n);

std::cout << "\nАрифметична прогресія:\n";

arithmeticpointer->getInfo(i, n);

std::cout << "\nГеометрична прогресія:\n";

geometricpointer->getInfo(i, n);

std::cout << '\n';

//Task 3

std::cout << "Демонстрація роботи функції foo для 3 завдання:\n";

// foo(defaultSeries, i, n);

foo(arithmetic, i, n);

// Task 4

std::cout << "\nЗавдання з масивом покажчиків\n";

srand(time(0)); // Ініціалізація генератора випадкових чисел

const int SIZE = 5; // Розмір масиву

Series\* seriesArray[SIZE]; // Масив покажчиків на базовий клас

// Заповнення масиву випадковими екземплярами класів-нащадків

for (int i = 0; i < SIZE; i++) {

double first = rand() % 10 + 1; // Генеруємо перший член прогресії

double step = rand() % 5 + 1; // Генеруємо різницю для арифметичної або знаменник для геометричної прогресії

if (rand() % 2 == 0) {

seriesArray[i] = new Arithmetic(first, step); // Створення арифметичної прогресії

}

else {

seriesArray[i] = new Geometric(first, step); // Створення геометричної прогресії

}

}

// Виклик методу GetInfo() для кожного елемента масиву

int i\_1;

std::cout << "Введіть номер i члена прогресії, який треба знайти для кожної прогресії: ";

std::cin >> i\_1;

int n\_1;

std::cout << "Введіть кількість перших членів прогресії, суму яких треба знайти: ";

std::cin >> n\_1;

for (int j = 0; j < SIZE; j++) {

std::cout << seriesArray[j]->start\_number;

std::cout << seriesArray[j]->increase;

seriesArray[j]->getInfo(i\_1, n\_1);

std::cout << "--------------------------" << '\n';

}

// Видалення динамічно створених об'єктів

for (int j = 0; j < SIZE; j++) {

delete seriesArray[j];

}

//delete defaultSeriespointer;

delete arithmeticpointer;

delete geometricpointer;

return 0;

}

Текст програми мовою Java:

Series.java:

package lab3;

public abstract class Series {

protected double startNumber; // Перший член прогресії

protected double increase; // Різниця або знаменник прогресії

// Конструктор

public Series(double startNumber, double increase) {

this.startNumber = startNumber;

this.increase = increase;

}

// Чистий віртуальний метод для обчислення i-го члена прогресії

public abstract void calculateTerm(int i);

// Чистий віртуальний метод для обчислення суми прогресії

public abstract void calculateSum(int n);

// Метод для виведення інформації про прогресію

public void getInfo(int i, int n) {

System.***out***.println("Інформація про прогресію:");

calculateTerm(i);

calculateSum(n);

}

}

Arithmetic.java:

package lab3;

public class Arithmetic extends Series {

// Конструктор

public Arithmetic(double startNumber, double increase) {

super(startNumber, increase);

}

// Перевизначення методу для обчислення i-го члена

*@Override*

public void calculateTerm(int i) {

double term = startNumber + (i - 1) \* increase;

System.***out***.println("i-й член арифметичної прогресії: " + term);

}

// Перевизначення методу для обчислення суми прогресії

*@Override*

public void calculateSum(int n) {

double sum = (n / 2.0) \* (2 \* startNumber + (n - 1) \* increase);

System.***out***.println("Сума перших " + n + " елементів арифметичної прогресії: " + sum);

}

// finalize метод для симуляції деструктора

*@Override*

protected void finalize() throws Throwable {

System.***out***.println("Видалення об'єкта класу Arithmetic");

}

}

Geometric.java:

package lab3;

public class Geometric extends Series {

// Конструктор

public Geometric(double startNumber, double increase) {

super(startNumber, increase);

}

// Перевизначення методу для обчислення i-го члена

*@Override*

public void calculateTerm(int i) {

double term = startNumber \* Math.*pow*(increase, i - 1);

System.***out***.println("i-й член геометричної прогресії: " + term);

}

// Перевизначення методу для обчислення суми прогресії

*@Override*

public void calculateSum(int n) {

double sum;

if (increase == 1) {

sum = startNumber \* n;

} else {

sum = startNumber \* (1 - Math.*pow*(increase, n)) / (1 - increase);

}

System.***out***.println("Сума перших " + n + " елементів геометричної прогресії: " + sum);

}

}

main.java

package lab3;

import java.util.Random;

import java.util.Scanner;

public class main {

public static void foo(Series obj, int i, int n) {

obj.getInfo(i, n);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.print("Введіть перший член прогресії: ");

double firstNumber = scanner.nextDouble();

System.***out***.print("Введіть різницю/знаменник прогресії: ");

double increase = scanner.nextDouble();

System.***out***.print("Введіть номер i члена прогресії: ");

int i = scanner.nextInt();

System.***out***.print("Введіть кількість членів прогресії: ");

int n = scanner.nextInt();

System.***out***.println();

// Створення екземплярів класів

Arithmetic arithmetic = new Arithmetic(firstNumber, increase);

Geometric geometric = new Geometric(firstNumber, increase);

// Виклик методів

arithmetic.getInfo(i, n);

geometric.getInfo(i, n);

System.***out***.println("\nСтворення масиву класів-нащадків:");

// Створення масиву з випадковими прогресіями

Random random = new Random();

Series[] seriesArray = new Series[5];

for (int j = 0; j < seriesArray.length; j++) {

double first = random.nextInt(10) + 1;

double step = random.nextInt(5) + 1;

if (random.nextInt(2) == 0) {

seriesArray[j] = new Arithmetic(first, step);

} else {

seriesArray[j] = new Geometric(first, step);

}

}

// Виклик методів getInfo для кожного елемента масиву

System.***out***.print("Введіть номер i члена прогресії для всіх прогресій: ");

int i\_1 = scanner.nextInt();

System.***out***.print("Введіть кількість перших членів прогресії для обчислення суми: ");

int n\_1 = scanner.nextInt();

for (Series series : seriesArray) {

series.getInfo(i\_1, n\_1);

System.***out***.println("--------------------------");

}

scanner.close();

}

}