Дніпровський національний університет

імені ОЛеся Гончара

Факультет прикладної математики

Кафедра математичного забезпечення ЕОМ

Звіт до лабораторної роботи 6

за курсом «Програмування»

Студента 3 курсу групи ПЗ-17-2

Спеціальності 121 Інженерія програмного забезпечення

Виконав:

Мурашов М. О.

м. Дніпро

2020 р.

Зміст

[1. Постановка задачі 3](#_Toc40370569)

[2. Опис розв’язку 3](#_Toc40370570)

[3. Вихідний текст програми 5](#_Toc40370571)

[4. Опис інтерфейсу 8](#_Toc40370572)

[5. Опис тестових прикладів 10](#_Toc40370573)

[6. Аналіз помилок 10](#_Toc40370574)

# Постановка задачі

Завдання лабораторної роботи полягає в реалізації двох патернів проектування та написанні реферату щодо використання патернів. Оскільки патерни присвячені двом різним задачам, тому реалізуються у вигляді двох окремих програм. Не обов’язково вигадувати один приклад, в якому застосовувались би обидва патерна з варіанту одночасно.

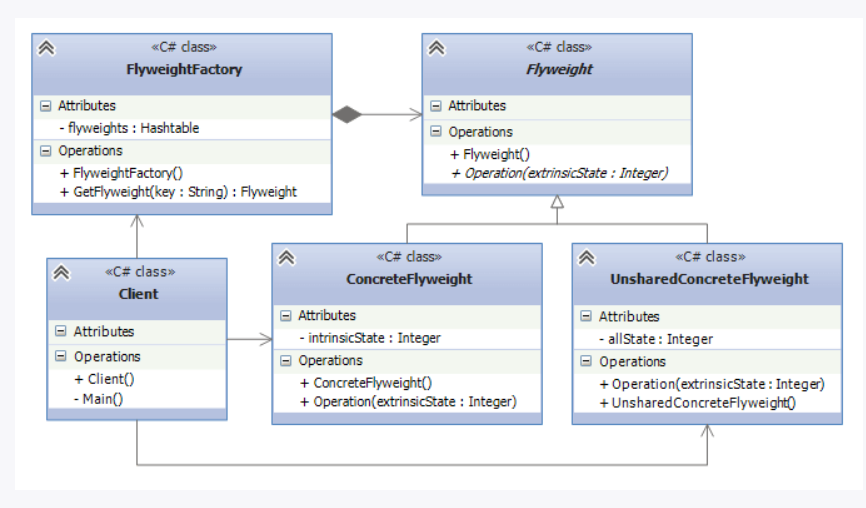
Перший з патернів кожного варіанту описаний в літературному джерелі [1]. Рекомендована тематика для прикладу використання: ієрархія графічних примітивів, яка вже реалізовувалася в лабораторній роботі 4 та курсовій роботі. В разі вибору такого прикладу зберігання повного функціоналу курсової роботи не є обов’язковим. Достатньо залишити прості фігури (без агрегатів), засоби їх графічного відображення та руху. Уникайте прикладів з беззмістовними класами на зразок «myClass1, myClass2», це не є демонстрацією практичного використання патернів.

Індивідуальне завдання (варіант 11):

Паттерн Flyweight

# Опис розв’язку

За основу було взято курсову роботу з цього курсу. Після чого було розглянуто саму структуру патерна:



Учасники:

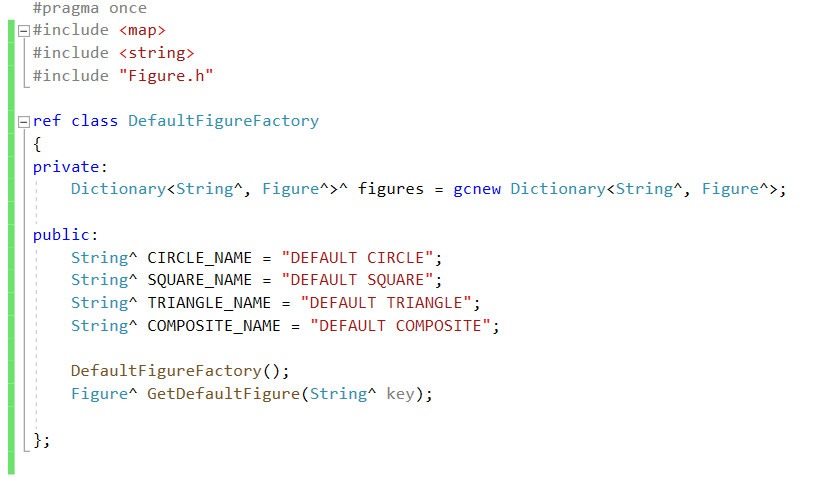
* *Flyweight:* визначає інтерфейс, через який пристосуванці-колективні об'єкти можуть отримувати зовнішнє стан або впливати на нього
* *ConcreteFlyweight:* конкретний клас розділяється пристосуванця. Реалізує інтерфейс, оголошений в типі Flyweight, і при необхідності додає внутрішній стан. Причому будь-яке зберігається їм стан має бути внутрішнім, не залежних від контексту
* *UnsharedConcreteFlyweight*: ще одна конкретна реалізація інтерфейсу, визначеного в типі Flyweight, тільки тепер об'єкти цього класу є неподільні
* *FlyweightFactory*: фабрика пристосуванців - створює об'єкти поділюваних пристосуванців. Так як пристосуванці поділяються, то клієнт не повинен створювати їх безпосередньо. Всі створені об'єкти зберігаються в пулі. В наведеному вище прикладі для визначення пулу використовується об'єкт Hashtable, але це не обов'язково. Можна застосовувати й інші класи колекцій. Однак в залежності від складності структури, що зберігає колективні об'єкти, особливо якщо у нас велика кількість пристосуванців, то може збільшуватися час на пошук потрібного пристосуванця - напевно це один з небагатьох недоліків даного патерну. Якщо запитаного пристосуванця не виявилося в пулі, то фабрика створює його.
* *Client*: використовує об'єкти пристосуванців. Може зберігати зовнішній стан і передавати його в якості аргументів в методи пристосуванців

Після чого згідно з патерном були створенні нові класи та використані вже існуючі. До таких класів увійшли:

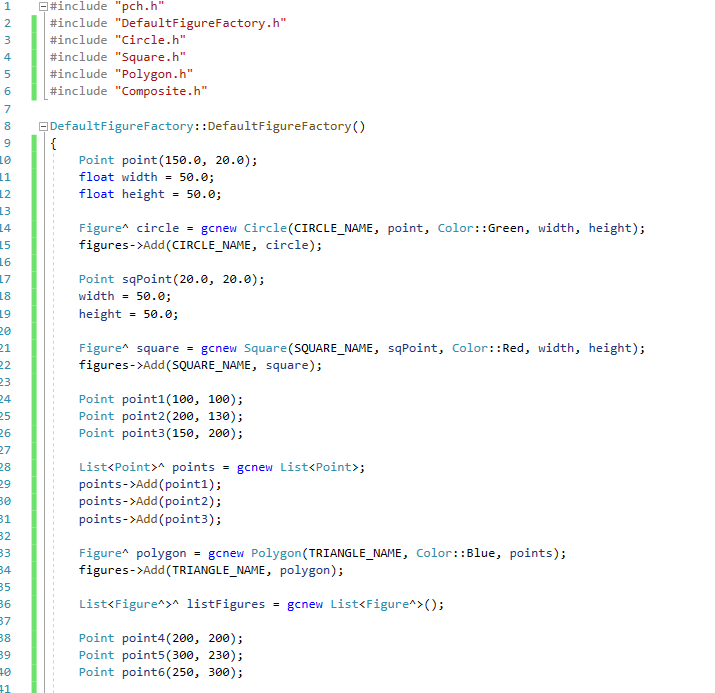
* Figure => *Flyweight*
* Square => *ConcreteFlyweight*
* Circle => *ConcreteFlyweight*
* Polygon => *ConcreteFlyweight*
* Composite => *UnsharedConcreteFlyweight*
* MainForm => *Client*
* DefaultFigureFactory => *FlyweightFactory*

# Вихідний текст програми

Був доданий новий клас DefaultFigureFactory:



Реалізація:

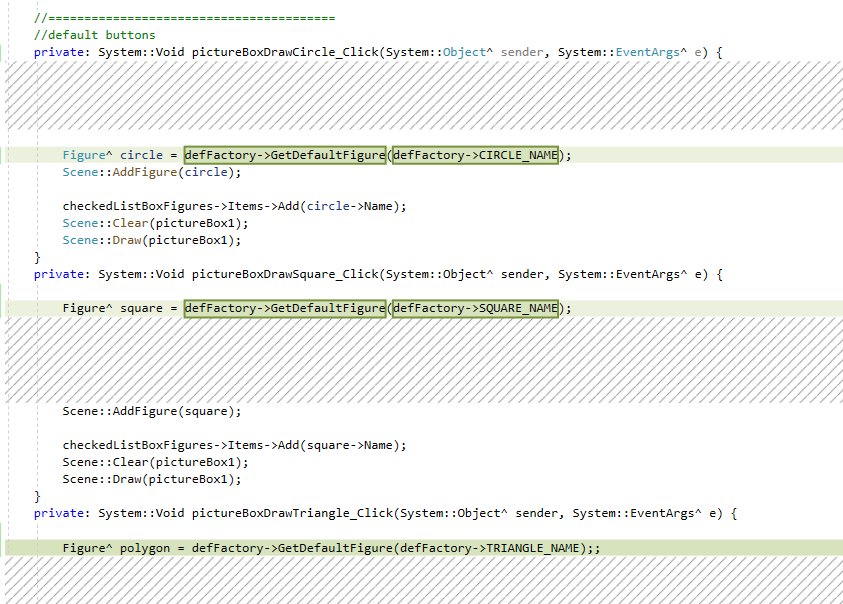


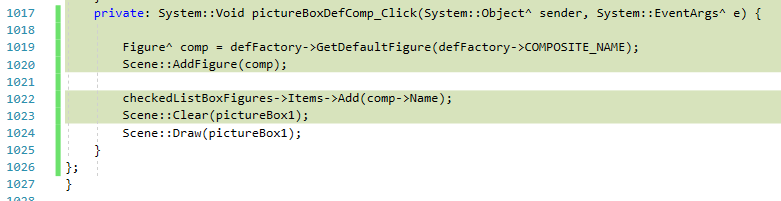


Та змінений клас головної форми. Було:



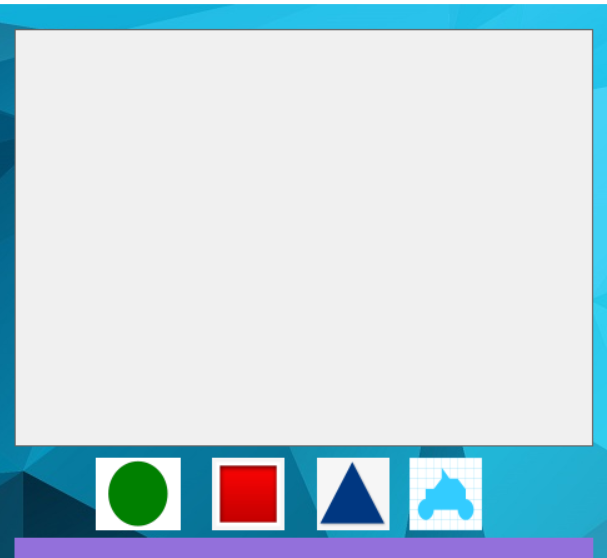
Стало:



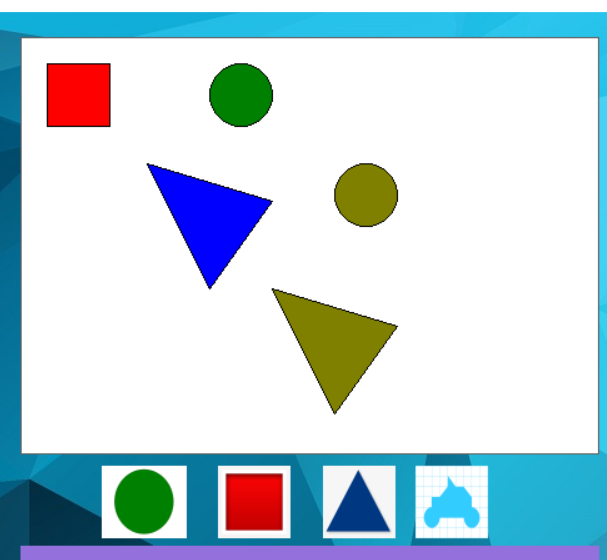


# Опис інтерфейсу

Щоб продемонструвати роботу патерну потрібно натискати на кнопки, що створюють фігури за замовчуванням (коло, квадрат, трикутник та композит):



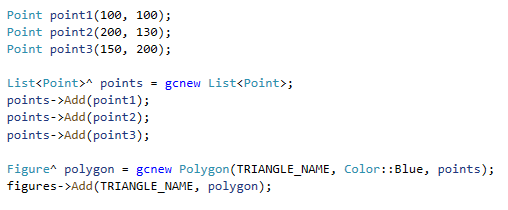
Після того, як натиснули всі кнопки маємо:



# Опис тестових прикладів

У якості тестових прикладів були створені фігури, що мають характеристики за замовчуванням (точки, колір, тип фігури)

Наприклад трикутник:



# Аналіз помилок

Оскільки чимало помилок було оброблено у самій курсовій роботі та чимало з них описано у документації, то буде доречним описати тільки нові обробки помилок. Наприклад, було додано перевірку у словник. Якщо елементу із відповідним ключем немає, то програма не зламається, а просто поверне пусти значення (яке буде оброблене):

