**ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА**

**Факультет прикладної математики**

**Кафедра математичного забезпечення ЕОМ**

**КУРСОВА РОБОТА**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Патерни проектування»

Студента 2 курсу групи ПЗ-20-3

Спеціальності 121 Інженерія програмного забезпечення

         Малиш. О. С.

 (прізвище та ініціали)

Керівник

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів \_\_\_\_\_\_\_\_\_\_\_\_

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії :  \_\_\_\_\_\_\_\_\_\_\_

          (підпис)                               (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_

          (підпис)                               (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_

          (підпис)                               (прізвище та ініціали)

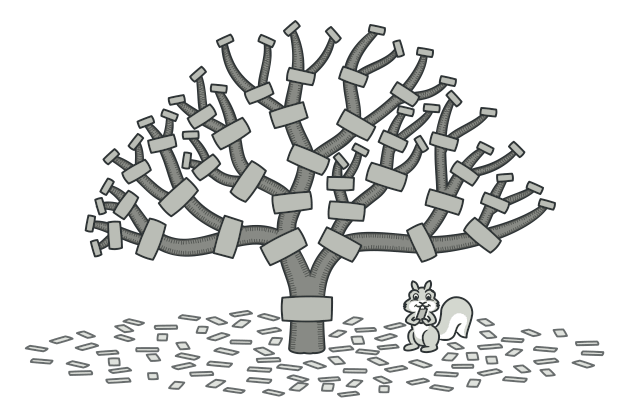
м. Дніпро, 2022 р.

1. Постановка задачі: Провести рефакторинг лабораторної роботи 4 (геометричні фігури) з використанням патернів проектування композит (Composite), прототип (Prototype), одинак (Singleton) і знімок (Memento), таким чином, щоб:
   1. агрегат представляв собою патерн композит (Composite);
   2. для збирання фігур в агрегат використовувався патерн прототип (Prototype);
   3. патерн одинак (Singleton) забезпечував існування лише одного контроллера сцени;
   4. патерн знімок (Memento) використовувався для зберігання стану наявних фігур у файл на диску.
2. Теоретична частина:
   1. **Патерн Компонувальник**

Також відомий як: Дерево, Composite

**Суть паттерну**

Компонувальник - це структурний патерн проектування, який дозволяє згрупувати безліч об'єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об'єкт.

 **Проблема**

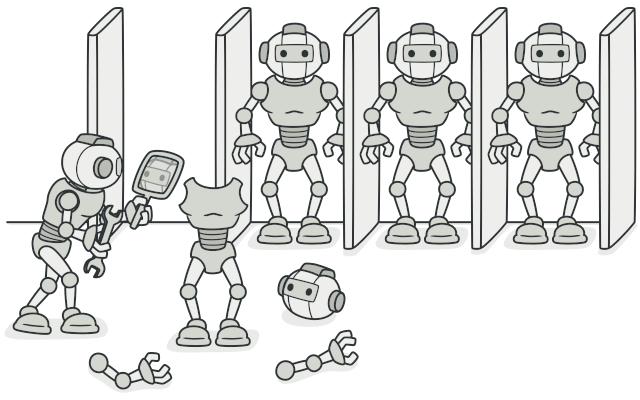
Паттерн Компонувальник має сенс лише тоді, коли основна модель вашої програми може бути структурована як дерева.

# Прототип

**Также відомий як :**Клон, Prototype

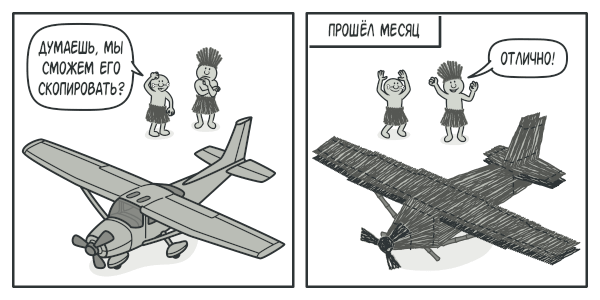
## Суть паттерну

**Прототип** — це породжуючий патерн проектування, який дозволяє копіювати об'єкти, не вдаючись у подробиці реалізації.



## Проблема

Ви маєте об'єкт, який потрібно скопіювати. Як це зробити? Потрібно створити порожній об'єкт такого ж класу, а потім по черзі скопіювати значення всіх полів зі старого об'єкта на новий.

Прекрасно! Але є нюанси. Не кожен об'єкт вдасться скопіювати таким чином, адже частина його стану може бути приватною, а отже, недоступною для решти коду програми.

Копіювання «ззовні» не завжди можливе насправді.

Але є й інша проблема. Копіюючий код буде залежати від класів об'єктів, що копіюються. Адже щоб перебрати всі поля об'єкта, потрібно прив'язатися до його класу. Через це ви не зможете копіювати об'єкти, знаючи лише їхні інтерфейси, а не конкретні класи.

# 3.Одинак

**Також відомий як:**Singleton

## Суть патерну

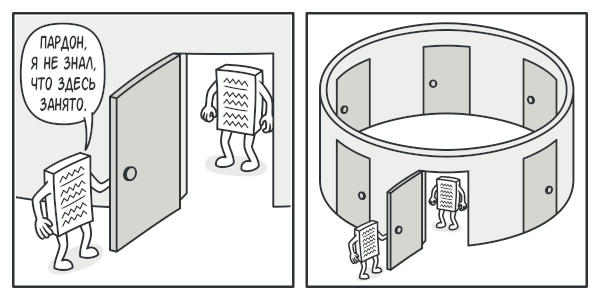
**Одинак — це породжуючий патерн проектування, який гарантує, що клас має лише один екземпляр, і надає до нього глобальну точку доступу.**

## Проблема

Одинак вирішує відразу дві проблеми, порушуючи принцип єдиної відповідальності класу.

1. Гарантує наявність єдиного екземпляра класу. Найчастіше це корисно для доступу до якогось загального ресурсу, наприклад, бази даних.

Уявіть, що ви створили об'єкт, а через деякий час намагаєтеся створити ще один. У цьому випадку хотілося б отримати старий об'єкт замість створення нового.

Така поведінка неможливо реалізувати за допомогою звичайного конструктора, оскільки конструктор класу завжди повертає новий об'єкт.

Клієнти можуть не підозрювати, що працюють з одним і тим самим об'єктом.

## ****2.** Надає глобальну точку доступу. Це не просто глобальна змінна, якою можна достукатися до певного об'єкта. Глобальні змінні не захищені від запису, тому будь-який код може змінювати значення без вашого відома.**

## **Але є й інший аспект. Непогано зберігати в одному місці і код, який вирішує проблему №1, а також мати до нього простий і доступний інтерфейс.**

## **Цікаво, що наш час патерн став настільки відомий, що тепер люди називають «одинаками» навіть ті класи, які вирішують лише одну з проблем, перерахованих вище.**

## Рішення

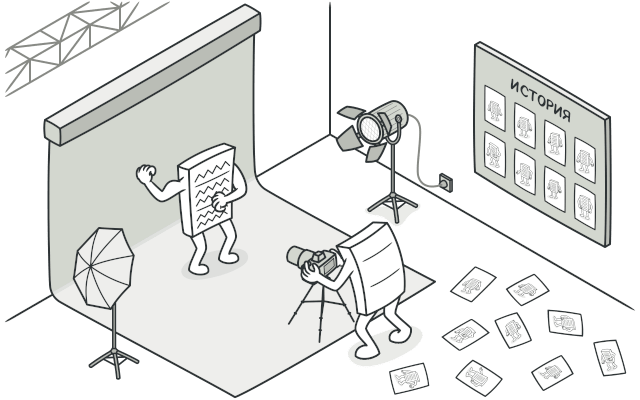
Всі реалізації одинака зводяться до того, щоб приховати конструктор за умовчанням і створити публічний статичний метод, який і контролюватиме життєвий цикл об'єкта-одинака.

Якщо у вас є доступ до класу одинак, значить, буде доступ і до цього статичного методу. З якої точки коду ви б його не викликали, він завжди віддаватиме той самий об'єкт.

# Знімок

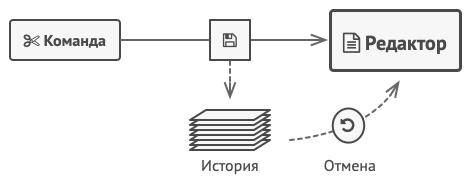
**Також відомий як:**Хранитель, Memento

**Суть паттерну**

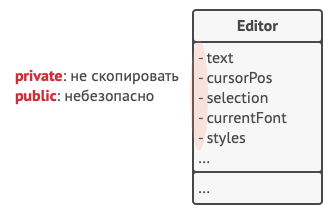
**Знімок –** це поведінковий патерн проектування, який дозволяє зберігати та відновлювати минулі стани об'єктів, не розкриваючи подробиць їх реалізації.

**Проблема**

Припустимо, що ми пишете програму текстового редактора. Крім звичайного редагування, ваш редактор дозволяє змінювати форматування тексту, вставляти картинки та інше.

У якийсь момент ви вирішили зробити всі ці дії скасовуються. Для цього вам потрібно зберігати поточний стан редактора, перш ніж виконати будь-яку дію. Якщо користувач вирішить скасувати свою дію, ви дістанете копію стану з історії та відновите старий стан редактора.

Перед виконанням команди можна зберегти копію стану редактора, щоб потім мати можливість скасувати операцію.Чтобы сделать копию состояния объекта, достаточно скопировать значение его полей. Таким чином, якщо ви зробили клас редактора досить відкритим, будь-який інший клас зможе заглянути всередину, щоб скопіювати його стан.

Здавалося б, що ще потрібне? Адже тепер будь-яка операція зможе зробити резервну копію редактора перед своєю дією. Але такий наївний підхід забезпечить вам безліч проблем у майбутньому. Адже якщо ви вирішите провести рефакторинг – прибрати або додати парочку полів у клас редактора – то доведеться змінювати код усіх класів, які могли копіювати стан редактора.

Як команді створити знімок стану редактора, якщо всі його поля приватні?

Але це ще не все. Давайте розглянемо самі копії стану редактора. Із чого складається стан редактора? Навіть найпримітивніший редактор повинен мати кілька полів для зберігання поточного тексту, позиції курсору та прокручування екрана. Щоб зробити копію стану, вам потрібно записати значення всіх цих полів у «контейнер».

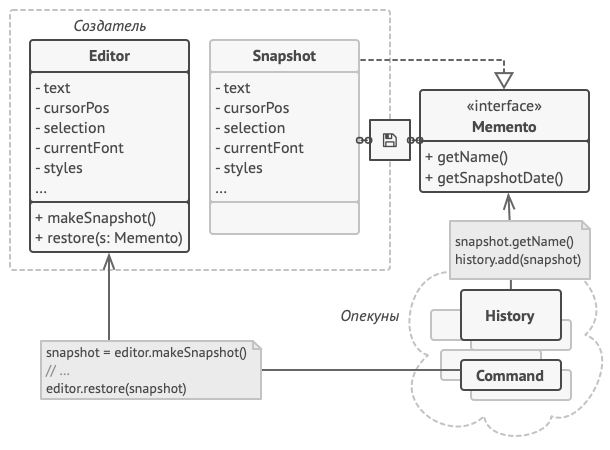
Швидше за все, вам знадобиться зберігати масу таких контейнерів як історію операцій, тому найзручніше зробити їх об'єктами одного класу. Цей клас повинен мати багато полів, але ніяких методів. Щоб інші об'єкти могли записувати і читати дані, вам доведеться зробити його поля публічними. Але це призведе до тієї самої проблеми, як і з відкритим класом редактора. Інші класи стануть залежними від будь-яких змін у класі контейнера, який схильний до тих самих змін, що й клас редактора.

Виходить, нам доведеться або відкрити класи для всіх бажаючих, зазнаючи маси клопоту з підтримкою коду, або залишити класи закритими, відмовившись від ідеї скасування операцій. Чи немає якогось іншого шляху?

**Рішення**

Усі проблеми, описані вище, виникають через порушення інкапсуляції. Це коли одні об'єкти намагаються зробити роботу за інших, влазячи в їхню приватну зону, щоб зібрати необхідні для операції дані.

Паттерн Знімок доручає створення копії стану об'єкта самому об'єкту, який володіє. Натомість, щоб робити знімок «ззовні», наш редактор сам зробить копію своїх полів, адже йому доступні всі поля, навіть приватні.

Паттерн пропонує тримати копію стану у спеціальному об'єкті-знімку з обмеженим інтерфейсом, що дозволяє, наприклад, дізнатися дату виготовлення або назву знімка. Але, з іншого боку, знімок має бути відкритим для свого творця, дозволяючи прочитати та відновити його внутрішній стан.

# Знімок повністю відкритий для творця, але лише частково відкритий для опікунів.

# Така схема дозволяє творцям робити знімки та віддавати їх для зберігання іншим об'єктам, які називають опікунами. Опікунам буде доступний лише обмежений інтерфейс знімка, тому вони ніяк не зможуть вплинути на «начинки» самого знімка. У потрібний момент опікун може попросити автора відновити свій стан, передавши відповідний знімок.

# У прикладі з редактором ви можете зробити опікуном окремий клас, який зберігатиме список виконаних операцій. Обмежений інтерфейс знімків дозволить демонструвати користувачеві гарний список із назвами та датами виконаних операцій. А коли користувач вирішить відкотити операцію, клас історії візьме останній знімок зі стека і відправить його редактору для відновлення.

1. Опис розв’язку:
   1. Для реалізації композиту в мене є клас “Copmosite”, що наслідує клас – Figure. Також реалізовано поле типу list, що містить фігури композиту і перевизначені методи базового класу Figure

Базовий клас фігура, в якій реалізований патерн прототип для збереження фігур в агрегат:

* 1. Також для створення агрегату з фігур використувується патерн «прототип». Кожна фігура може бути клонована відповідним методом. Виділені фігури також можна клонувати.

Базовий клас фігура, в якій реалізований паттерн прототип для збереження фігур в агрегат:

* 1. SceneController використовуєт патерн сінглтон. Він надає можливість мати більше одного екземпляру цього класу

Клас, в якому реалізований патерн Сінглтон та інтерфейс для всієї програми:

* 1. У Класі Serializer реалізований патернн «мементо». Мементо надає можливість зберігати файл на комп'ютері із станом сцени, котра може бути потім відновлена. Я створив метод, який записує поля об’єкту у відповідному форматі форматі «поле;поле;поле;….;» та повертають його. Кожна фігура починається з окремого рядку.

Клас серіалізації, в якому реалізовано патерн мементо:

1. Записка користувачеві:

Для взаємодії з додатком можна використовувати зручний інтерфейс. Нижче будуть представлені усі гарячі клавіші:

\*\*\*ИСПОЛЬЗУЙТЕ СТРЕЛОЧКИ ДЛЯ ПЕРЕМЕЩЕНИЯ ФИГУР\*\*\*

Використання клавіш керування курсором для переміщення виділених фігур

Home – Изменить цвет

Клавіша home змінює колір виділених фігур, у додатку використовується RGB

End – вкл/выкл след

Клавіша End перемикає два положення увімкнути та вимкнути слід від виділених фігур

Insert – изменить видимось

Клавіша Insert перемикає два положення увімкнути та вимкнути видимість виділених фігур

f1 – выбрать все фигуры

Клавіша F1 перемикає два положення увімкнути та вимкнути виділення фігур

f2 – очистить все

Клавіша F2 видаляє усі фігури які були відмальовані на сцені

left shift - Выбрать выбраные фигуры в стартовую позицию

Клавіша Left Shift повертає виділені фігури на стартове положення

left alt – очистить текущий выбор

Клавіша Left Alt повністю очищає з вікна програми поточне виділення фігур

Right alt – принудительное обновление

Клавіша Right Alt робить примусове оновлення вікна у разі підвисання або іншої несправності

0-9 выбрать фигуру на сцене

Клавішами від 0 до 9 вибираються намальовані фігури для подальшої взаємодії

v – добавить заготовленый композит

Клавіша V малює у вікні програми заздалегідь створений агрегат для подальшого використання

r - добавить прямоугольник

Клавіша R малює у вікні програми заздалегідь створений квадрат для подальшого використання

t - добавить треугольиник

Клавіша T малює у вікні програми заздалегідь створений трикутник для подальшого використання

c - копировать выбраные фигуры

Клавіша С дублює у вікні програми виділені фігури для подальшого використання

g - сгрупировать выбраные фигуры и добавить их на сцену

Клавіша G групує у вікні програми виділені фігури в окремий агрегат для подальшого використання

s - сохранить сцену в файл

Клавіша S зберігає сцену вікна програми окремо винесений файл

u - загрузить сохраненую сцену

Клавіша U відкриває сцену вікна програми з окремо винесеного файлу

d - удалить все из файла

Клавіша D повністю очищає сцену вікна програми з окремо винесеного файлу

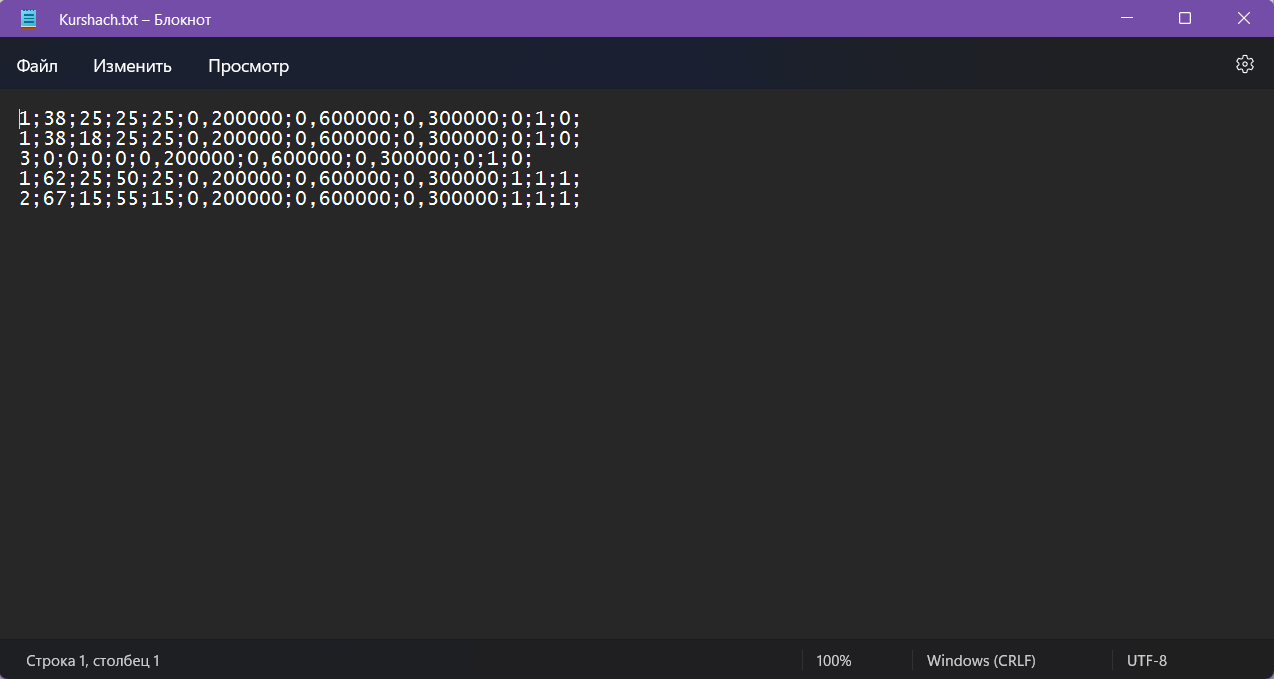
- - удалить выбраную фигуру

Клавіша - повністю очищає виділені фігури у сцені вікна програми

1. Тестові приклади роботи:

Серіалізація:

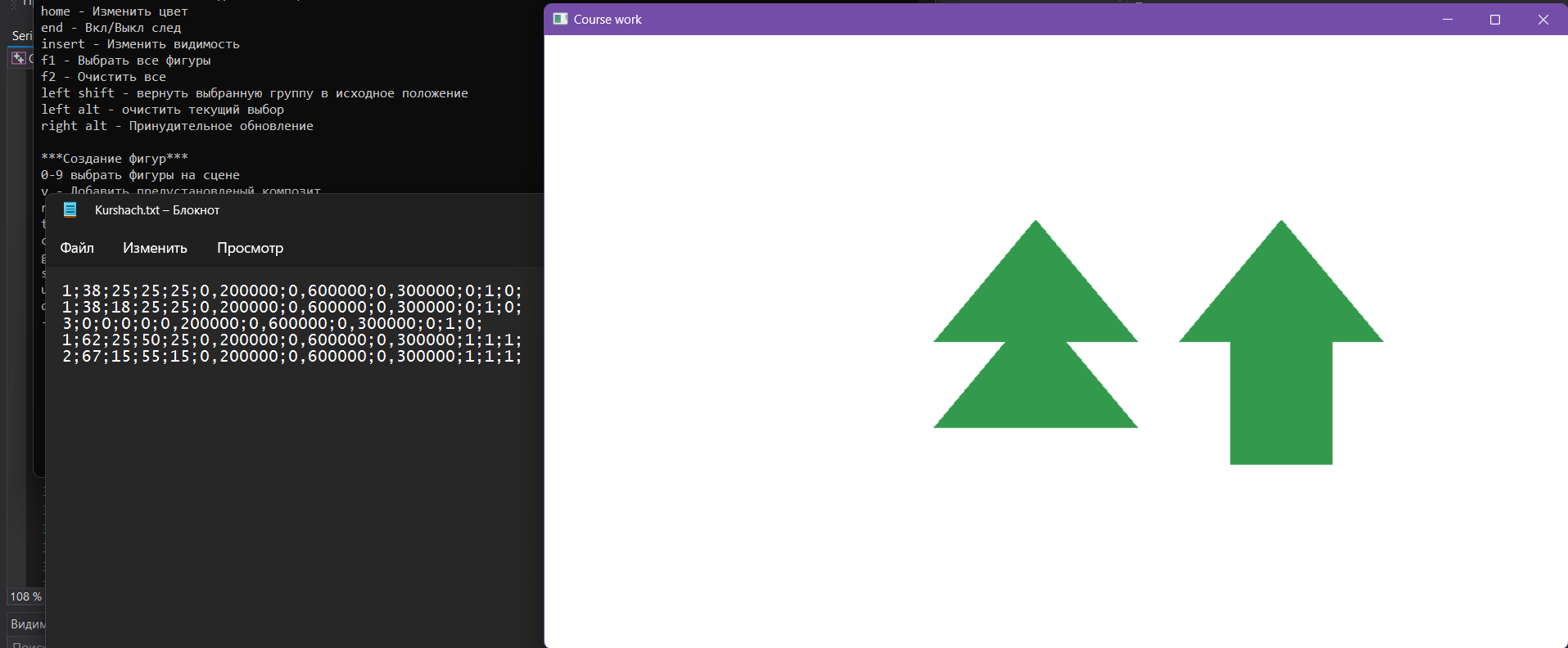
У нас є файл, в якому зберігаються данні фігур та композитів:



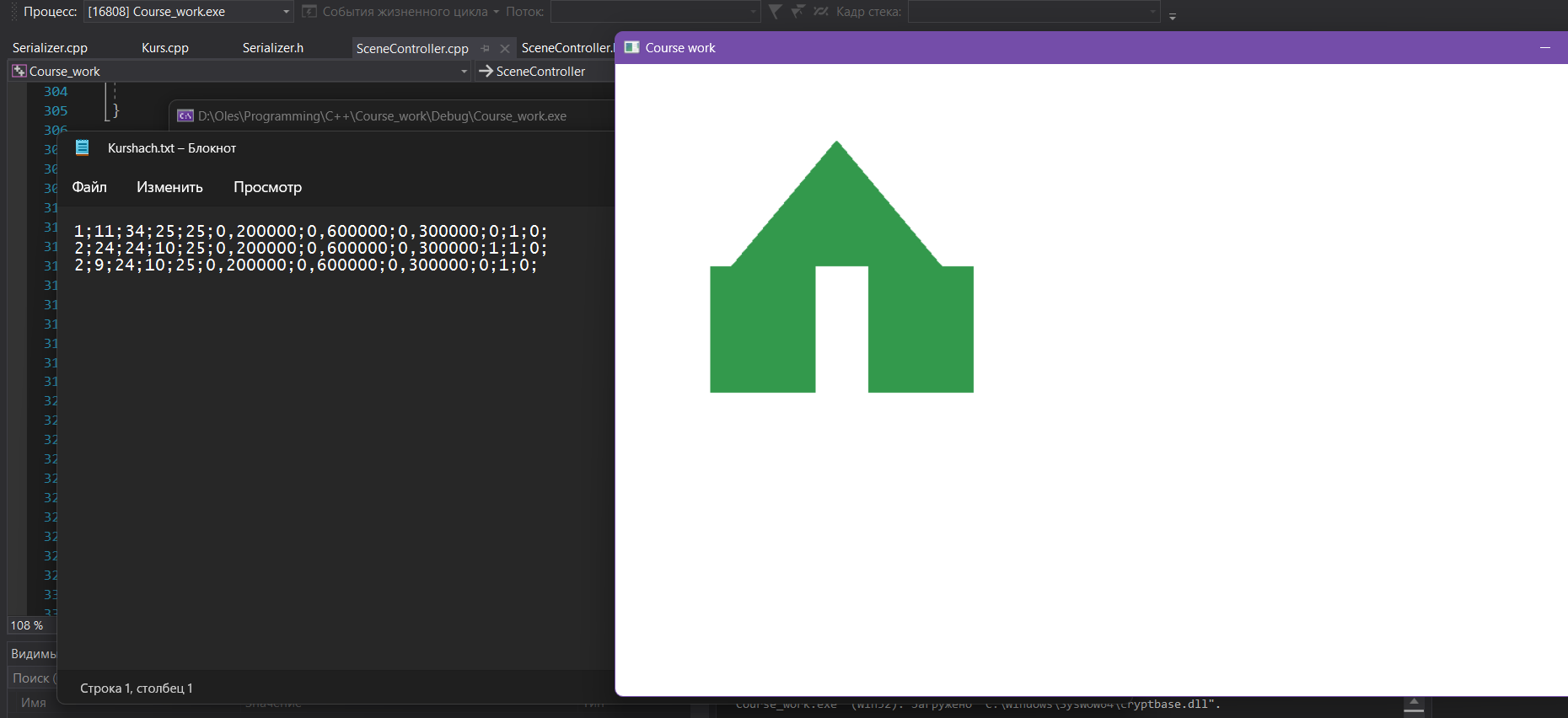
де, відповідно, є координати, колір, перевірка на видимість, чи активна фігура, та чи є фігура наслідником.

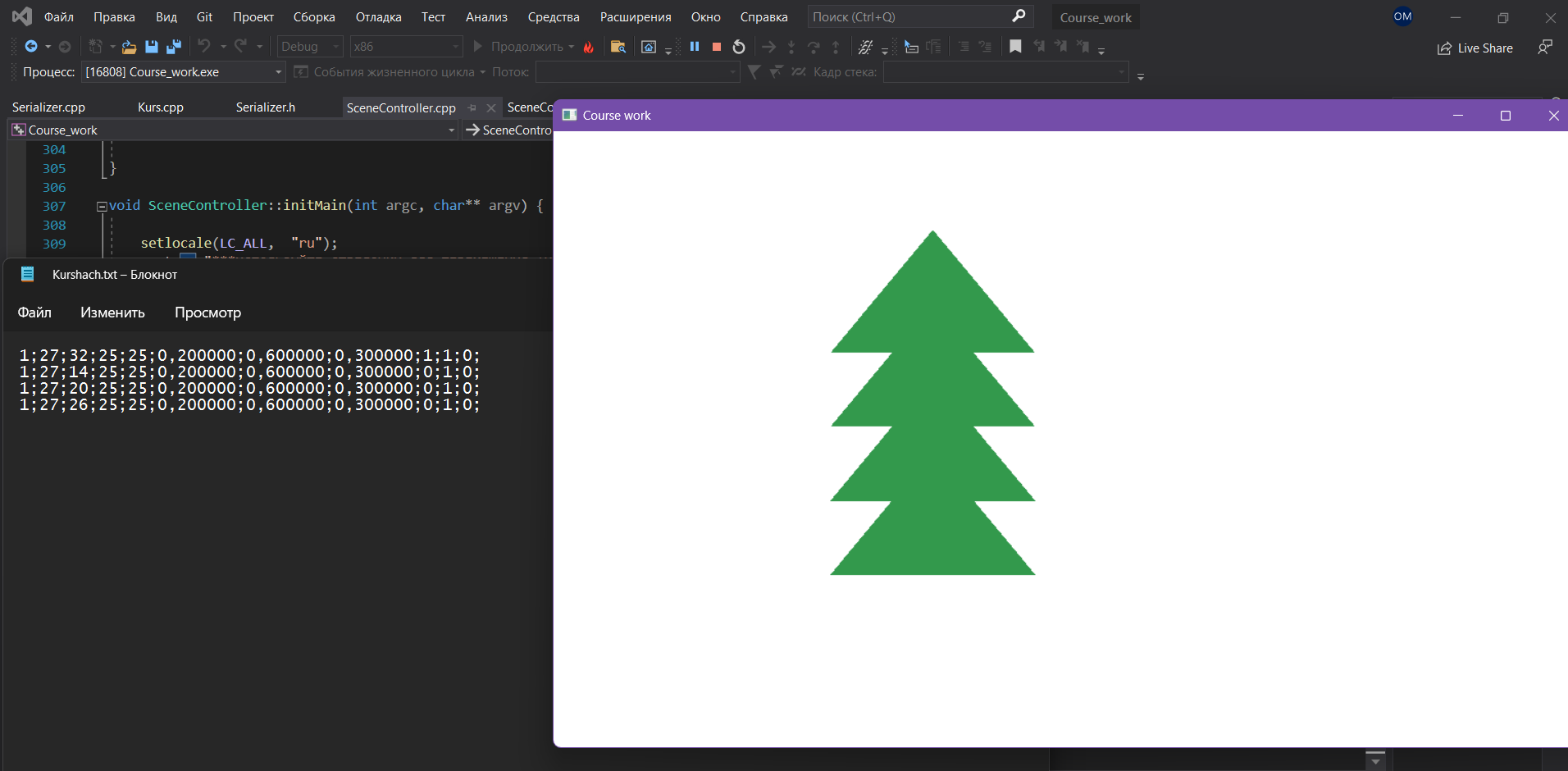
Кожне нове поле – це нова фігура.

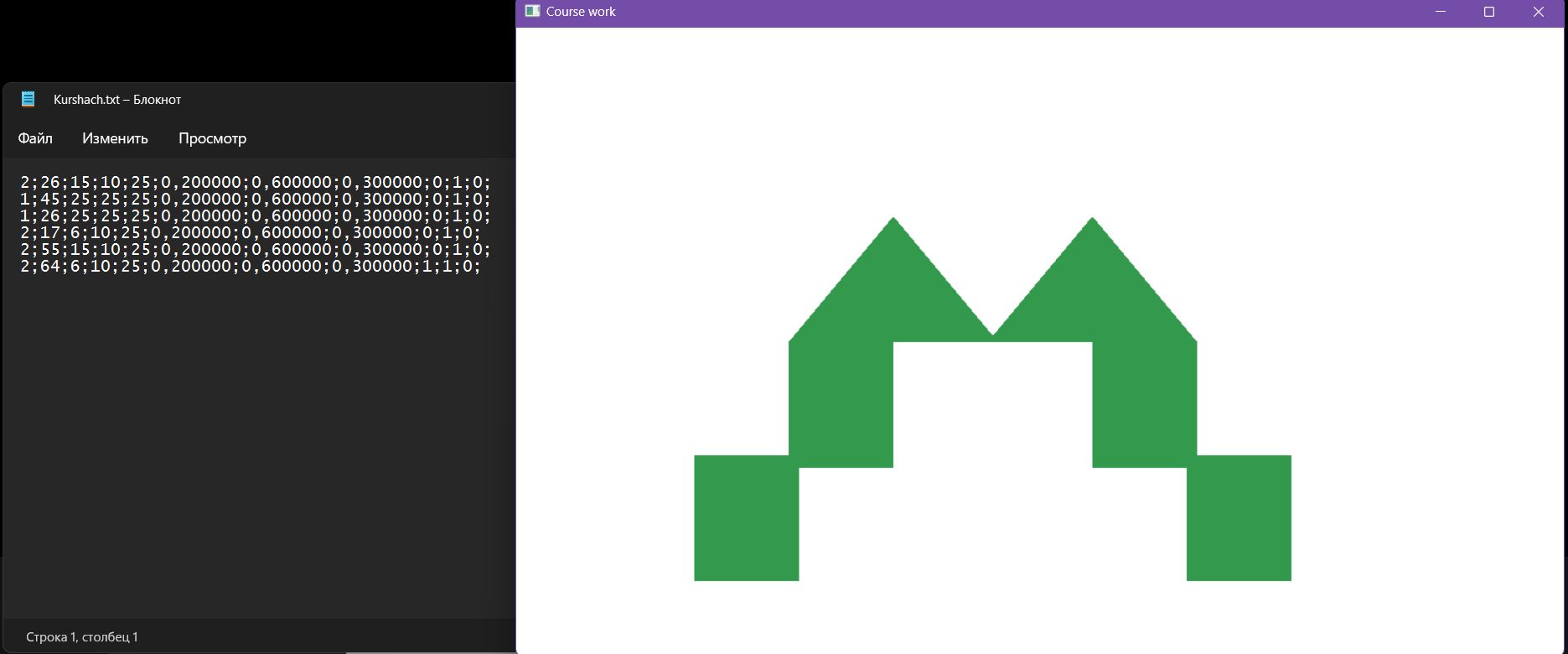
Сцена після визову метода десеріализації:



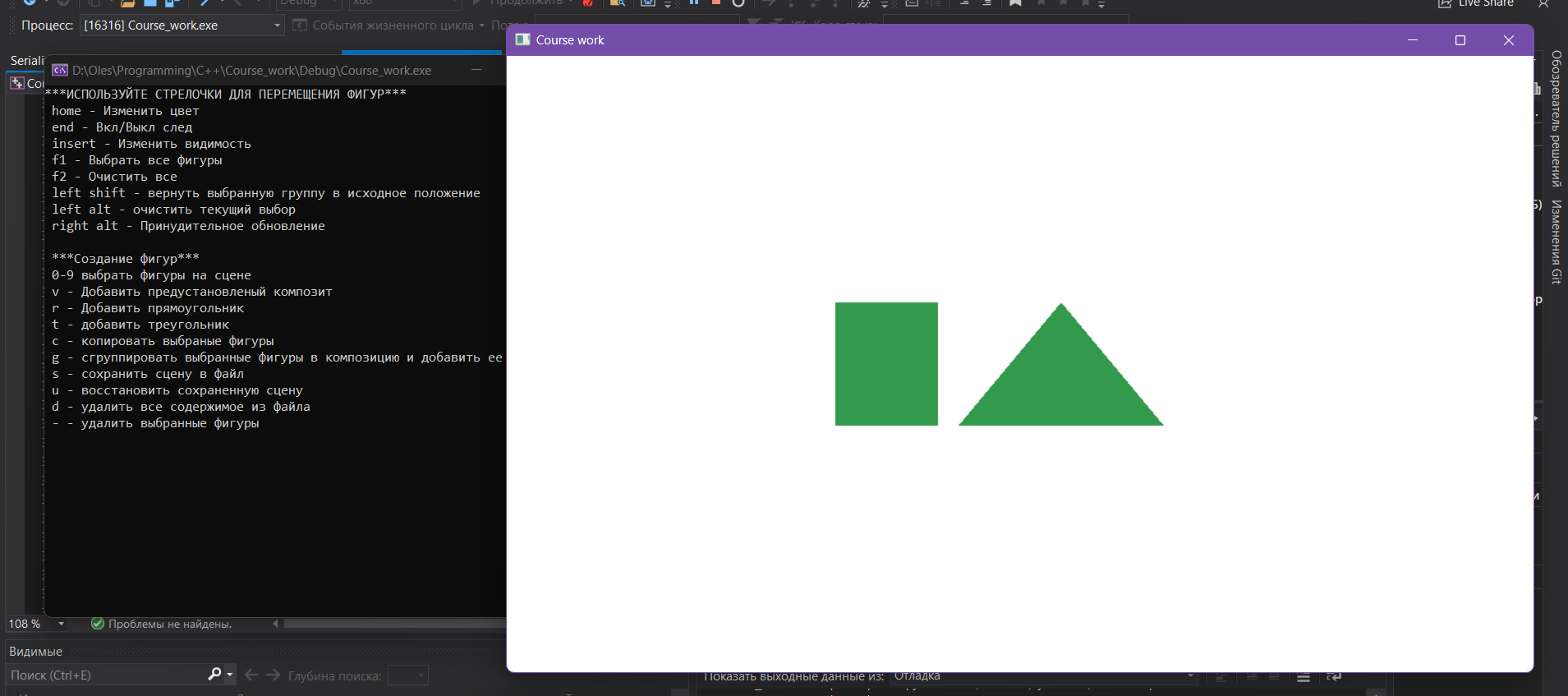
Також декілька інших прикладів:

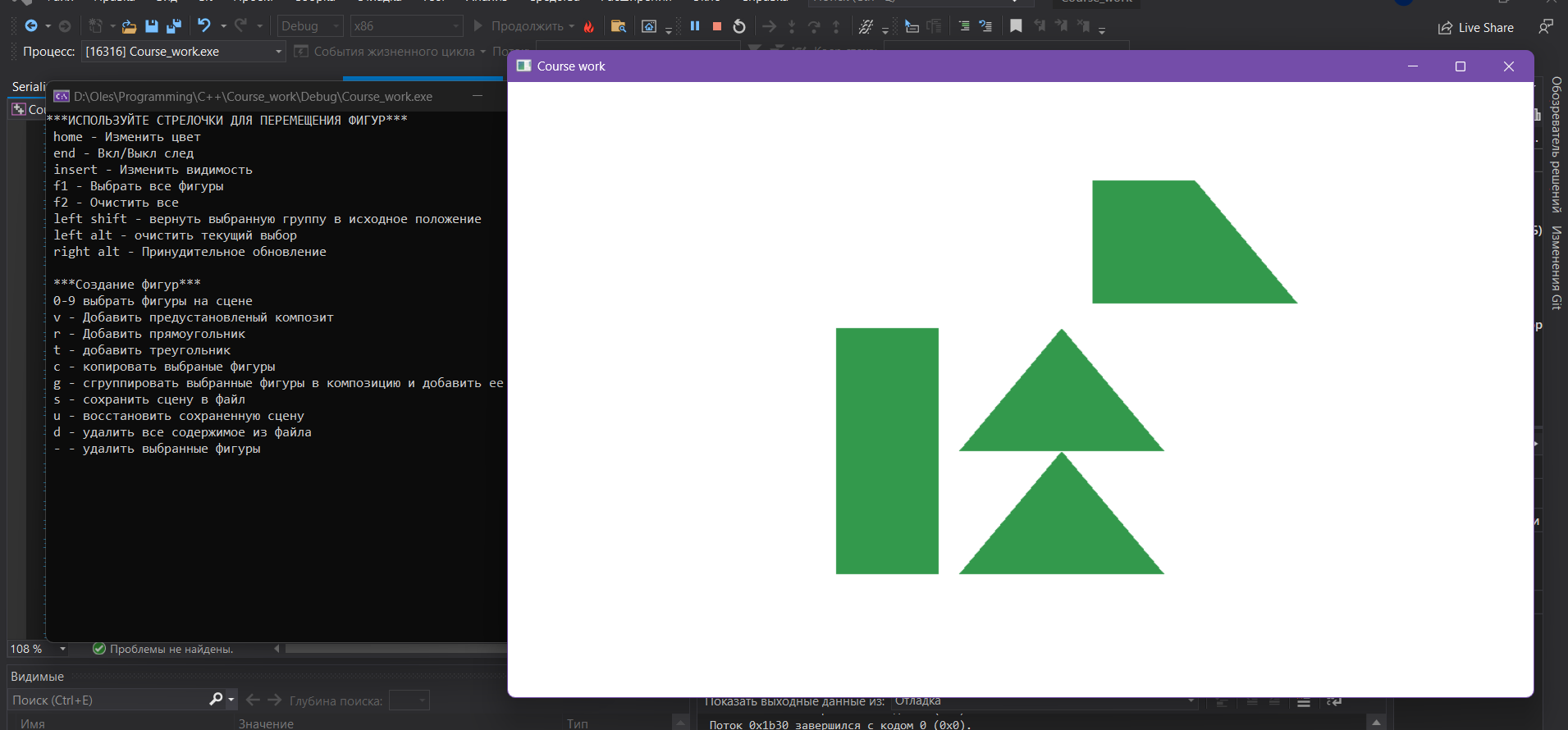


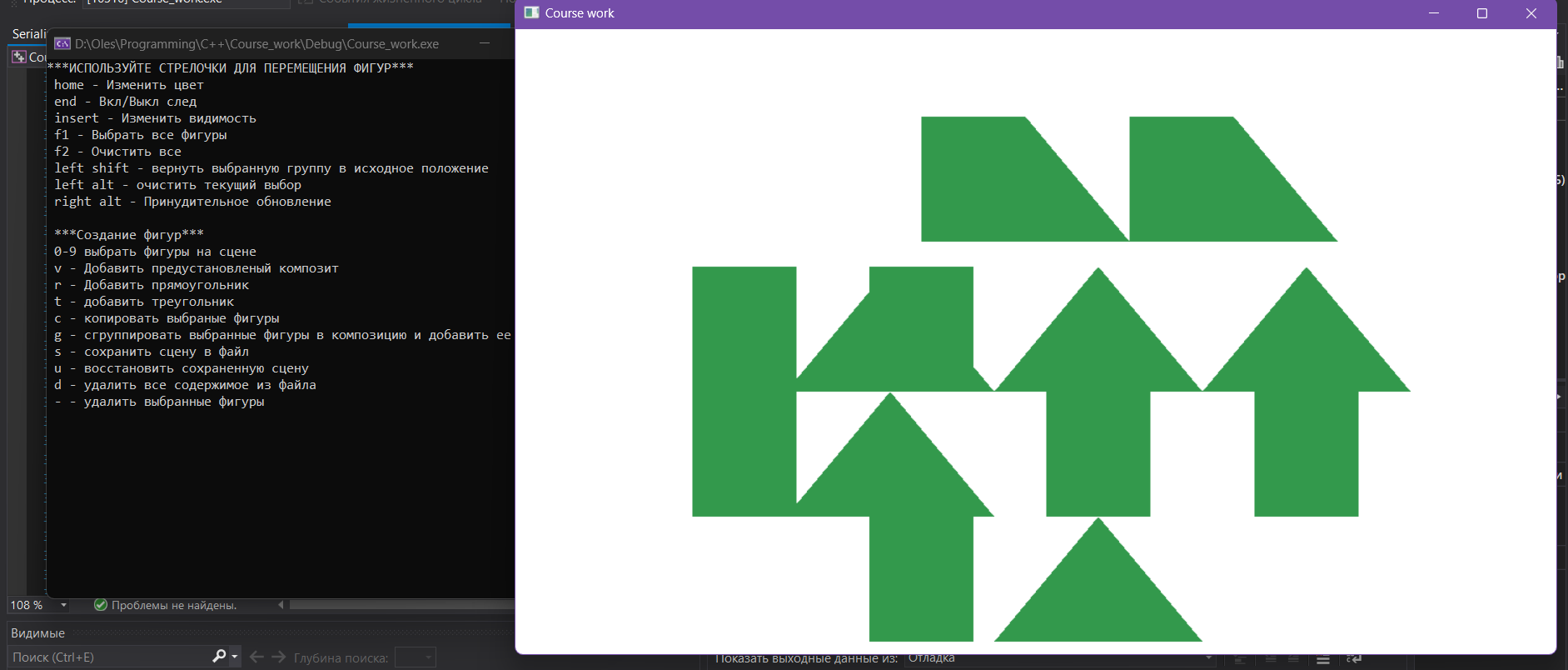




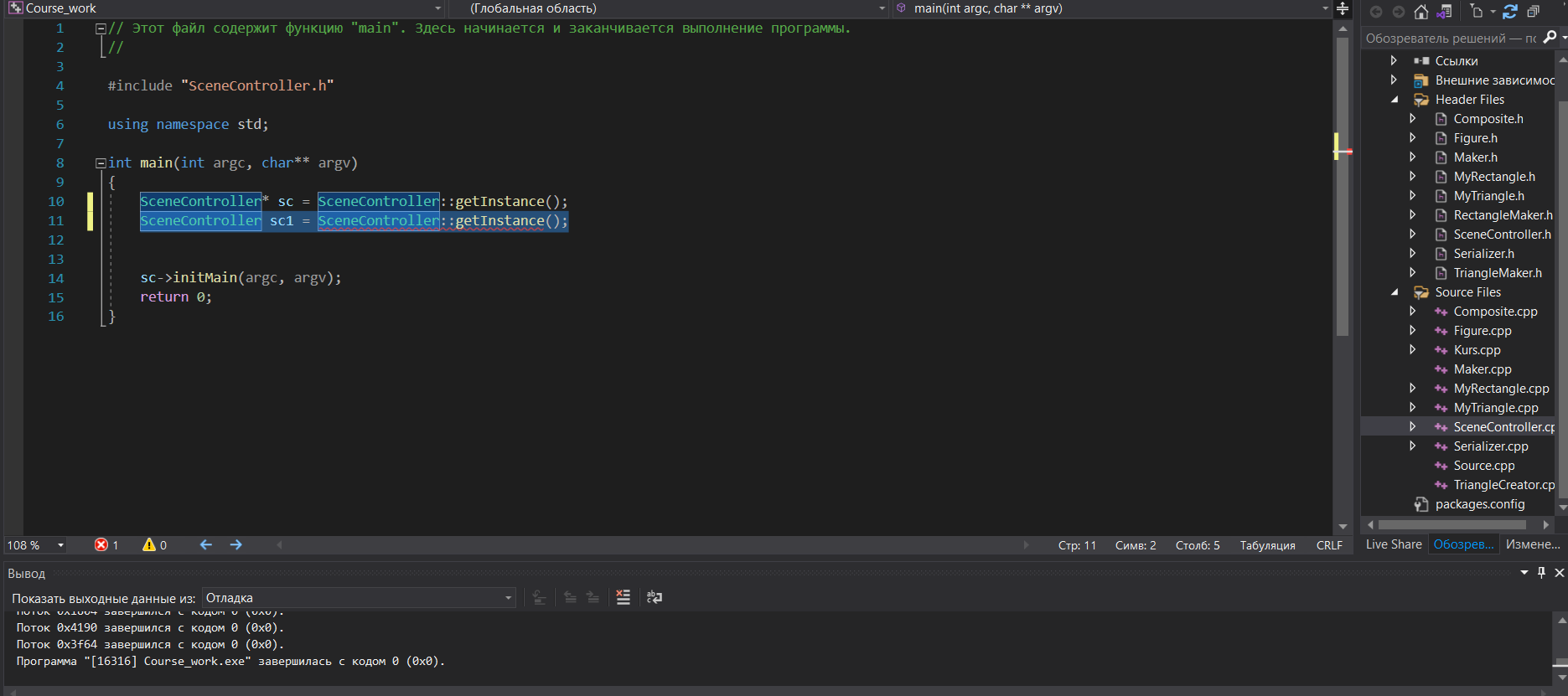
Збирання фигур у агрегат:







Спроба створити ще один екземпляр класу SceneController (Singltone pattern):



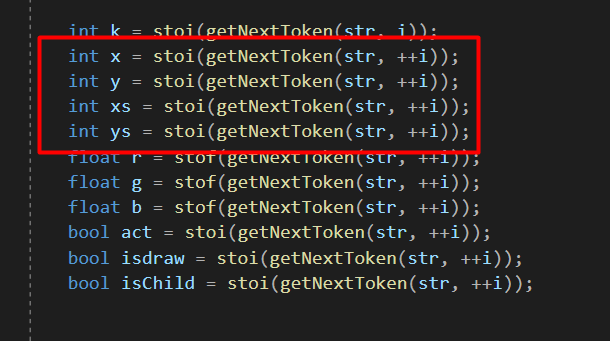
1. Аналіз помилок і зауважень

Було таке зауваження: Чому у збереженому файлі (Мементо) для трикутника прописується 2 пари координат.

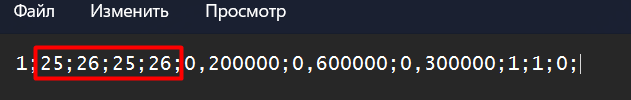
Пояснення:

У нас є поля x, y, xs, ys.

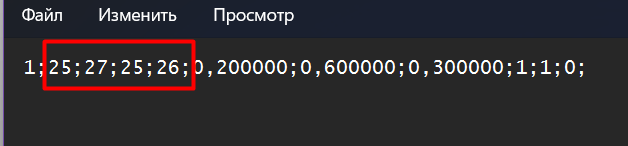
X та Y це поточні координати нашої фігури, тобто її розсташування в реальному часі. А поля xs та xy це початкові координати при створені фігури, і вони не змінюються.



Наприклад, створимо фігуру з координатами x = 25, y = 26.



Давайте здвинемо фигуру на 1 в гору (тобто по y), та збережемо данні в файл. Ось що в нас буде:



1. Список літератури:
   1. opengl.org: Підключення та маршрутизація бібліотеки GLUT

(<https://www.opengl.org/resources/libraries/glut/glut_downloads.php>)

* 1. metanit.com: Успадкування

(<https://metanit.com/cpp/tutorial/5.10.php>)

* 1. refactoring.guru: Патерни проектування (<https://refactoring.guru/uk/design-patterns>)
  2. wikipedia.com: Паттерни проектування ([https://ru.wikipedia.org/wiki/](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%82%D1%82%D0%B5%D1%80%D0%BD))
  3. habr.com: Робота з бібліотекою OpenGL

(<https://habr.com/ru/post/111175/>)