

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ЛІСОТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

Інститут комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерних наук

КУРСОВА РОБОТА

З дисципліни *«Об'єктно-орієнтоване програмування»*

Виконав

Студент групи КН-21

Гайванович О. А.

Керівник роботи

Сало М. Ф.

м. Львів – 2024

1. ЗАВДАННЯ НА КУРСОВУ РОБОТУ

студенту 2-го курсу групи КН-21

Гайвановичу Остапу

Варіант 9

1. Створити клас ВЕРШИНА з координатами точки на площині.
2. Створити базовий клас ФІГУРА, який містить масив об'єктів класу ВЕРШИНА. Визначити чисто віртуальний метод переміщення фігури на площині.
3. Для обох класів визначити конструктори ініціалізації, переміщення, копіювання, деструктори та методи для зміни і читання значень полів даного класу.
4. Перевантажити операції для класу ВЕРШИНА: + для додавання координат, - для віднімання, операцію * для множення на коефіцієнт, операцію / для ділення на коефіцієнт, операцію присвоєння об'єктів = та переміщення, потокові операції введення » та виведення « об'єктів.
5. Створити похідні класи: ПРЯМОКУТНИК та КВАДРАТ. Визначити функції переміщення фігури у межах площини. Визначити конструктори та деструктори, методи або операторні функції введення-виведення.
6. У межах ієрархії класів побудувати поліморфічний кластер на основі віртуального методу переміщення фігури. Продемонструвати механізм пізнього зв'язування.
7. Розробити клас-контейнер, що містить масив об'єктів класів ПРЯМОКУТНИК та КВАДРАТ. Знайти фігуру з найбільшою площею.
8. Для роботи з масивом об'єктів побудувати та використати клас-ітератор.
9. Реалізувати пункти 1-8 завдання мовою C++.
10. Реалізувати пункти 1-8 завдання відповідними засобами мови C# з використанням WinForms.
11. Вимоги до оформлення курсової роботи дивитись у методичних вказівках до виконання курсового проектування з ООП (диск Р)
12. Забезпечити читання і запис даних для формування масиву об'єктів згідно завдання із файлу.

Вимоги:

1. Вхідні дані до роботи: літературні джерела, технічна документація щодо розробки програм, ДСТУ з оформлення документації.
2. Зміст пояснювальної записки: Вступ. Специфікація роботи. Програмна документація. Висновки. Додатки.

3. Перелік графічного матеріалу: діаграма класів додатку; схема алгоритму реалізації одного з розроблених методів (за узгодженням з керівником курсової роботи).

Дата здачі студентом завершеної роботи 15.05.2024 р.

Календарний план виконання курсової роботи

№ з/п	Назва етапу роботи	Строк виконання	Відмітка про виконання
1	З'ясування загальної постановки завдання; розробка чернетки першого розділу пояснювальної записки.	з 20.11.2023 р. по 17.01.2024 р.	Виконано
2	Програмна реалізація, налагодження та тестування додатка; розробка чернетки пояснювальної записки (другий розділ, висновки, список використаних джерел, додатки).	з 18.01.2024 р. по 04.04.2024 р.	Виконано
3	Остаточне налагодження та тестування додатка; розробка чернетки пояснювальної записки (вступ, другий розділ, висновки, список використаних джерел, додатки).	з 05.04.2024 р. по 30.04.2024 р.	Виконано
4	Розробка остаточного варіанта пояснювальної записки (вступ, другий розділ, висновки, список використаних джерел, додатки).	з 30.04.2024 р. по 15.05.2024 р.	Виконано

Дата видачі завдання: 20.11.2023 р.

Керівник роботи: Сало М. Ф.

(підпис)

Студент: Гайванович О. А.

(підпис)

ЗМІСТ

1. ЗАВДАННЯ НА КУРСОВУ РОБОТУ	2
2. РЕФЕРАТ	5
ВСТУП.....	6
3. ФОРМУЛЮВАННЯ ЗАДАЧІ	8
4. МЕТОДИ ТА ЗАСОБИ РОЗВ’ЯЗУВАННЯ ЗАДАЧІ	9
5. ПРОЄКТУВАННЯ СТРУКТУРИ ПРОГРАМИ.....	10
5.1. Структура програми на C++	10
5.2. Структура програми на C#	11
6. ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА	13
6.1. Консольний інтерфейс на C++	13
6.2. Графічний інтерфейс засобами .NET і Windows Forms.....	15
6.2.1. Дизайн основної форми.	15
6.2.2. Дизайн форми створення фігури	15
6.2.3. Дизайн форми інформації про фігуру	16
6.2.4. Дизайн форми інформації про фігуру	17
7. РОЗРОБЛЕННЯ ГРАФІЧНИХ СХЕМ КЛАСІВ	18
7.1. Діаграма класів програми на C++.....	18
7.2. Діаграма класів програми на C#	19
8. РОЗРОБЛЕННЯ ПРОГРАМИ ТА ЇЇ ОПИС	21
9. ІНСТРУКЦІЯ КОРИСТУВАЧА	23
9.1. Інструкція до програми на C++	23
9.2. Інструкція до програми на C#	27
10. КОНТРОЛЬНИЙ ПРИКЛАД ТА АНАЛІЗ РЕЗУЛЬТАТІВ КОМП’ЮТЕРНОЇ РЕАЛІЗАЦІЇ ПРОГРАМИ.....	31
10.1. Контрольний приклад програми на C++.....	31
10.2. Контрольний приклад програми на C#	31
ВИСНОВКИ.....	32
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	33
ДОДАТКИ	34

2. РЕФЕРАТ

Пояснювальна записка: 45 стор., 8 рис., 2 табл., 3 додатки, 15 джерел.

ОБ'ЄКТ РОЗРОБЛЕННЯ: ієрархія класів для представлення геометричних фігур на площині та демонстрації концепцій об'єктно-орієнтованого програмування.

МЕТА РОБОТИ: реалізувати систему класів для опису вершин, фігур, їх властивостей та операцій, а також продемонструвати використання принципів ООП, таких як успадкування, поліморфізм, перевантаження операторів, використання контейнерів та ітераторів.

МЕТОДИ ТА ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ: об'єктно-орієнтоване програмування з використанням C++ та C#, середовища розробки Microsoft Visual Studio, бібліотек STL та Nlohmann-JSON (для C++), фреймворку Microsoft .NET, графічного API Windows Forms (для C#).

РЕЗУЛЬТАТИ ТА РЕКОМЕНДАЦІЇ: розроблено ієрархію класів для представлення вершин, фігур (прямокутників, квадратів) та операцій над ними. Реалізовано перевантаження операторів, використання контейнера та ітератора, візуалізацію за допомогою Windows Forms. Програма демонструє переваги ООП: інкапсуляцію, поліморфізм, модульність, повторне використання коду.

ЗНАЧУЩІСТЬ РОБОТИ: набуття практичних навичок об'єктно-орієнтованого програмування, реалізації принципів ООП, використання бібліотек та середовищ розробки. Отримані результати можуть бути використані для вивчення ООП та розширені для створення більш складних систем.

КЛЮЧОВІ СЛОВА: об'єктно-орієнтоване програмування, ієрархія класів, фігури, перевантаження операторів, віртуальні методи, контейнери, ітератори, c++, c#, WinForms.

ВСТУП

Об'єктно-орієнтоване програмування (ООП) — це парадигма програмування, яка базується на концепції об'єктів, що містять дані (властивості) та коди (методи). ООП забезпечує кращу модульність програм та повторне використання коду шляхом пакування даних та функціональності в об'єкти.

Основними принципами ООП є:

1. **Інкапсуляція:** об'єднання даних та методів, що оперують цими даними, в єдиний об'єкт, забезпечуючи приховування реалізації від зовнішнього світу.
2. **Успадкування:** можливість створення нових класів на основі існуючих, успадковуючи їхні властивості та методи.
3. **Поліморфізм:** здатність об'єктів різних класів обробляти той самий метод різними способами.
4. **Абстракція:** виділення суттєвих характеристик об'єкта та відкидання несуттєвих деталей.

Однією з ключових переваг ООП є підвищена модульність, повторне використання коду, спрощена розробка та супровід великих проєктів. Ці переваги роблять ООП широко застосовуваним у розробці програмного забезпечення, особливо у великих та складних системах.

C++ є однією з найпопулярніших та широко використовуваних мов програмування у світі. Вона була розроблений у 1980-х роках Б'ярном Страуструпом як розширення мови C, додаючи підтримку об'єктно-орієнтованого програмування, що стало його ключовою особливістю.

C++ повністю підтримує концепції ООП, надаючи можливість створювати класи, успадковувати їх, використовувати поліморфізм через механізм віртуальних функцій, а також реалізовувати інкапсуляцію та абстракцію. Крім того, C++ дозволяє поєднувати об'єктно-орієнтоване та процедурне програмування в одному проєкті.

C++ також має багато інших особливостей, таких як обробка винятків, перевантаження операторів, шаблони, багатопотоковість тощо, що роблять його потужним та гнучким інструментом для розробки різноманітних додатків, включаючи системне програмування, ігри, мультимедіа та багато іншого.

Мова програмування C# була розроблена в 1998—2001 рр. компанією Microsoft як частина її загальної .NET-стратегії. Головним архітектором C# був Андерс Хейлсберг, один з провідних фахівців у галузі мов програмування, який здобув світове визнання завдяки створенню успішного продукту Turbo Pascal у 1980-х роках.

C# безпосередньо пов'язана з C++ та Java, і це не випадково: адже ці три мови є найпопулярнішими та найулюбленішими мовами програмування у світі. Велика кількість професійних програмістів знають C, C++ та Java. Оскільки C# побудований на міцному, зрозумілому фундаменті цих «фундаментальних» мов, перехід до нього відбувається без особливих труднощів.

Ключові особливості C#:

- Уніфікована система типів, де все розглядається як об'єкт (класи, структури, масиви, вбудовані типи).
- Використання просторів імен замість включення файлів заголовків.
- Простота синтаксису: відсутність операторів "->" та ":", використання лише оператора ".".
- Сучасність: можливість використання псевдонімів для просторів імен та класів, атрибутів для полегшення відладки коду.
- Можливість прикріплення типізованих, розширюваних метаданих до об'єктів для зв'язку бізнес-логіки з кодом.

Порівняно з іншими мовами:

- Синтаксис C# успадкований від C++ та Java, тому для розробників, що мають досвід з цими мовами, він буде знайомим.
- На відміну від C++, C# вводить такі новації, як атрибути, делегати та події, які дозволяють застосовувати принципово нові прийоми програмування.
- Порівняно з Java, C# має підтримку просторів імен, властивостей, індексаторів та перелічень контейнерів.
- Унікальними рисами C# є події, індексатори, атрибути та делегати.

3. ФОРМУЛЮВАННЯ ЗАДАЧІ

Метою даної роботи є реалізація ієрархії класів для представлення геометричних фігур на площині та демонстрація різних концепцій об'єктно-орієнтованого програмування, таких як успадкування, поліморфізм, перевантаження операторів, використання контейнерів та ітераторів.

Вхідні дані:

- Координати по осях X та Y для створення об'єктів класу Вершина
- Координати нижньої лівої вершини та сторони (для Квадрату) чи двох сторін (для Прямокутника) для створення об'єктів класів Прямокутник, Квадрат
- Параметри для переміщення фігур (наприклад, зміщення по осях X та Y)

Вихідні дані:

- Відображення координат вершин створених фігур
- Результати операцій над фігурами (додавання/віднімання координат, множення/ділення на коефіцієнт, переміщення фігури)
- Візуалізація фігур на площині
- Фігура з найбільшою площею серед створених

Вимоги до інструментальних засобів:

- Середовище розробки для C++
- Середовище розробки для C# з підтримкою фреймворку Microsoft .NET та графічного API Windows Forms

Наведені вимоги визначають необхідну функціональність, вхідні/вихідні дані, форми їх представлення, а також інструментальні засоби, потрібні для успішної реалізації поставленого завдання курсової роботи.

4. МЕТОДИ ТА ЗАСОБИ РОЗВ'ЯЗУВАННЯ ЗАДАЧІ

Для реалізації поставленого завдання було використано об'єктно-орієнтований підхід до програмування. Він передбачає розбиття системи на об'єкти, які інкапсулюють дані та функціональність, і взаємодію між цими об'єктами для виконання певних операцій. Зокрема, були застосовані такі основні концепції ООП:

1. Абстракція та інкапсуляція для визначення класів (Вершина, Фігура, Прямокутник, Квадрат) з відповідними даними та методами.
2. Успадкування для створення ієрархії класів, де похідні класи успадковують властивості та поведінку базових класів.
3. Поліморфізм для реалізації віртуальних методів та пізнього зв'язування при роботі з об'єктами через базовий клас.
4. Перевантаження операторів для покращення читабельності коду та підвищення зручності роботи з об'єктами класу Вершина.
5. Шаблони для створення класів-контейнерів та ітераторів, що працюють з різними типами об'єктів.

Засоби розробки:

1. Середовище розробки: Microsoft Visual Studio зі встановленими відповідними компонентами для роботи з C++ з-під консолі та C# з .NET Windows Forms
2. Компілятори C++ (з підтримкою стандарту C++20) та C#
3. Бібліотека стандартних шаблонів STL C++
4. Для C++ — стороння бібліотека JSON автора Nlohmann — для серіалізації / десеріалізації фігур
5. Засоби для відладки та тестування програм

Вибір цих методів та засобів обумовлений їх широкою підтримкою та поширеністю у сфері об'єктно-орієнтованого програмування. Застосування ООП дозволяє створювати модульні, масштабовані та легкі для підтримки системи. Використання сучасних IDE, як-от Visual Studio, полегшує процес написання, компіляції та налагодження коду. Бібліотеки стандартних контейнерів та графічні бібліотеки забезпечують необхідну функціональність без потреби у створенні всіх компонентів «з нуля». А формат JSON дуже поширений для серіалізації об'єктів і легко читається людиною.

Переваги використаних підходів: висока гнучкість, модульність та можливість повторного використання коду. Крім того, ООП відповідає сучасним практикам розробки і широко підтримується на різних платформах.

5. ПРОЄКТУВАННЯ СТРУКТУРИ ПРОГРАМИ

Структура програми була спроектована з використанням концепцій ООП, таких як інкапсуляція, успадкування та поліморфізм. Структура класів була спроектована відповідно до вимог завдання та принципів ООП.

Застосування об'єктно-орієнтованого підходу дозволило створити модульну та легко розширювану структуру програми, з можливістю додавання нових типів фігур шляхом успадкування від базового класу. Інкапсуляція забезпечила приховування внутрішньої реалізації об'єктів від зовнішнього коду, а поліморфізм — можливість роботи з об'єктами різних класів через базовий клас.

Детальну структуру класів, їх відношення успадкування, атрибути та методи можна проілюструвати за допомогою діаграм класів (див. РОЗРОБЛЕННЯ ГРАФІЧНИХ СХЕМ КЛАСІВ).

5.1. Структура програми на C++

Клас *Vertex* (Вершина) представляє собою вершину з координатами точки на площині, по осях X та Y. Є параметризований конструктор (за замовчування значення координат рівні 0), конструктори копіювання та переміщення. Є гетери та сетери координат, перевантажені оператори присвоювання, порівняння, потокового введення та виведення, а також базові арифметичні оператори та оператори. Для зручності визначення типу даних координат було прийнято рішення ввести псевдонім типу *_Vertex_t*, який використовуватиметься у всій програмі; за замовчуванням цьому псевдоніму присвоєний тип *float*.

Клас *Shape* (Фігура) — абстрактний клас, що представляє геометричну фігуру з масивом вершин та віртуальними методами. Присутні конструктори ініціалізації масиву вершин трьома шляхами. Реалізований метод перевірки введених вершин на факт послідовного розташування за годинниковою стрілкою. Реалізований віртуальний метод переміщення фігури, оголошено кілька чистих віртуальних методів.

Класи *Square* та *Rectangle* — похідні від *Shape*. Містяться конструктори створення фігур за верхньою лівою вершиною та стороною чи сторонами (для квадрату чи прямокутника відповідно). Є можливість створити фігуру з усіх вершин — тоді відбуватиметься валідація тих вершин. Реалізовані усі чисті віртуальні методи батьківського класу. Серед методів є метод для обчислення площі фігури.

Простір імен *menuMethods* включає методи, що можуть використовуватись як дії меню. Окрім цього, тут також є внутрішні простіри імен, *cin_aux* та *os_aux*, де містяться допоміжні функції для роботи з потоками, що спрощують роботу при розробці нових дій пунктів меню.

Клас *Menu* описує консольне меню. Структуру меню потрібно визначити самостійно, використовуючи методи для дій з простору імен *menuMethods*, є можливість створювати підменю або ж додати пункт меню (невеликий опис та ім'я функції, тобто її вказівник).

Крім того, був реалізований шаблонний клас-контейнер *simpleVector* для зберігання фігур у контейнері для ітерації по об'єктах контейнера. Для назв методів використовується інтерфейс бібліотеки стандартних шаблонів STL.

В точці входу в програму, головному методі *main()*, створюється об'єкт класу *Menu* та його структура, з подальшим показом меню на консоль та його запуском. Також тут «ловляться» всі можливі виключення (якщо вони раніше не були опрацьовані, наприклад, в методах з простору імен *menuMethods*): в такому випадку виводиться повідомлення про таке виключення і робота програми завершується.

5.2. Структура програми на C#

Клас *Vertex* описує вершину. Реалізовує інтерфейс *Comparable<Vertex>*. Перевизначено кілька методів класу *Object*. Як і у програмі на C++, є параметризований конструктор зі значеннями за замовчуванням, конструктор копіювання, визначені оператори порівняння. Для координат замість полів використано властивості, яких немає в C++.

Абстрактний клас *Shape* представляє геометричну фігуру. Інтерфейс *ICloneable* має бути реалізований у похідних класах. Окрім властивостей імені та масиву вершин є ще одна, що визначає колір фігури на полотні. Перевизначено кілька методів класу *Object*. Присутній конструктори ініціалізації масиву вершин. Реалізований метод перевірки введених вершин на факт послідовного розташування за годинниковою стрілкою. Реалізований віртуальний метод переміщення фігури, оголошено кілька чистих віртуальних методів.

Класи *Square* та *Rectangle* — похідні від *Shape*. Містяться конструктори створення фігур за верхньою лівою вершиною та стороною чи сторонами (для квадрату чи прямокутника відповідно). Є можливість створити фігуру з усіх вершин — тоді відбуватиметься валідація тих вершин. Реалізовані усі чисті віртуальні методи батьківського класу та інтерфейси. Серед методів є метод для обчислення площі фігури.

Клас *SimpleDictionary* є узагальненим і являє собою просту реалізацію словника, де «під капотом ховається» масив з типом даних *KeyValuePair* (структура ключ-значення). Реалізовує інтерфейс *IDictionary*: містить методи для додавання, вилучення, очищення елементів, перевірки на існування елемента тощо. Також

створений клас-енумератор *SimpleDictionaryEnumerator* для даного контейнера, як альтернатива класу-ітератору в C++.

Клас *ColorConverter* — це клас-конвертор, що використовується для серіалізації/десеріалізації з/в JSON, для перетворення зі структури кольору *System.Drawing.Color* в стрічку та навпаки.

Створено статичний клас *glob*, щоб в будь-якій точці програми отримати доступ, коли знадобиться, до об'єкту головної форми та масиву фігур.

Клас *Program* містить лише один метод, *Main()*, в якому створюється головна форма і звідки починається точка входу в програму.

Також в проєкті містяться наступні класи форм Windows Forms:

- *fm_main*: основна форма програми, містить список фігур, перелік дій та полотно, що відображає фігури в обмеженій частині площини.
- *fm_info*: форма показу інформації про фігуру.
- *fm_new_shape*: форма створення нової фігури з усіма необхідними параметрами.
- *fm_move*: форма переміщення фігури, де можна задати нові координати.

Детальніше про опис створених форм, а також про інтерфейс користувача, описано в розділі «ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА».

6. ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

6.1. Консольний інтерфейс на C++

У застосунку на C++ розроблене консольне меню має наступний вигляд. Червоним кольором посередині показується назва меню чи підменю. Є голуба рамка, яка виконує суто декоративну функцію (її розмір в класі *Menu* заданий як константа). Далі розташовані елементи меню, що складаються з:

- Символу, який потрібно ввести для виклику даної дії чи переходу до підменю
- Опису елемента меню, виділеного зеленим кольором

Останній елемент в меню чи підменю — дія виходу з програми.

Внизу розташований рядок вводу символу; введені символи підсвічуються світло-жовтим кольором.

```
Menu
a -> Test action
s -> Submenu
0 -> Exit
Select item: |
```

Рис. 6.1 Інтерфейс меню з однією дією й одним підменю

Виконаємо тестову дію *Test action* — введемо символ *t* в полі для вводу, побачимо спочатку назву дії зеленого кольору, а нижче результат виконання (в даному випадку виведе на консоль *Testing menu*). Після виконаної дії знову знову малюється поточне меню.

```
Select item: a
Test action
Testing menu
```

Рис. 6.2 Виклик дії в меню

Після переходу до підменю, в нашому випадку *Submenu*, друкуються спочатку його елементи, а нижче розташований пункт переходу до попереднього, батьківського меню, а також пункт виходу з програми.

```
Select item: s
Submenu
a -> Test action
- -> Back
0 -> Exit
Select item: |
```

Рис. 6.3 Вибір підменю з однією дією

Загалом готову спроектовану структуру консольного меню можна описати у вигляді дерева.

- ❖ Main menu (Головне меню)
 - Create (Створити)
 - Rectangle (Прямокутник)
 - Square (Квадрат)
 - Edit (Редагувати)
 - Clear (Очистити фігури)
 - Move (Перемістити фігуру)
 - Remove (Вилучити створену фігуру)
 - File (Файл)
 - Save as json (Зберегти в JSON)
 - Load from json (Завантажити з JSON)
 - Save as txt (Зберегти в TXT)
 - Print shape info (Надрукувати інформацію про створені фігури)
 - Max square (Надрукувати максимальну площу)
 - Print shape names (Надрукувати ім'я створених фігур)

6.2 Графічний інтерфейс засобами .NET і Windows Forms

6.2.1. Дизайн основної форми.

Зліва розташовані список дій у вигляді груп з кнопок (позначено *Actions* — дії), які можна виконати відносно фігур. Доступне збереження інформації про фігури у вигляді форматів TXT та JSON, а також завантаження збережених фігур з JSON-файлів.

Далі, по центру (позначено *Shapes*), міститься список створених фігур; при подвійному кліці на одну з таких фігур відкриватиметься форма інформації про фігуру.

Справа (там, де *Canvas*) — полотно, де графічно малюватимуться створені фігури. Для спрощення реалізації прийнято рішення обмежити область, без можливості змінити її чи масштаб.

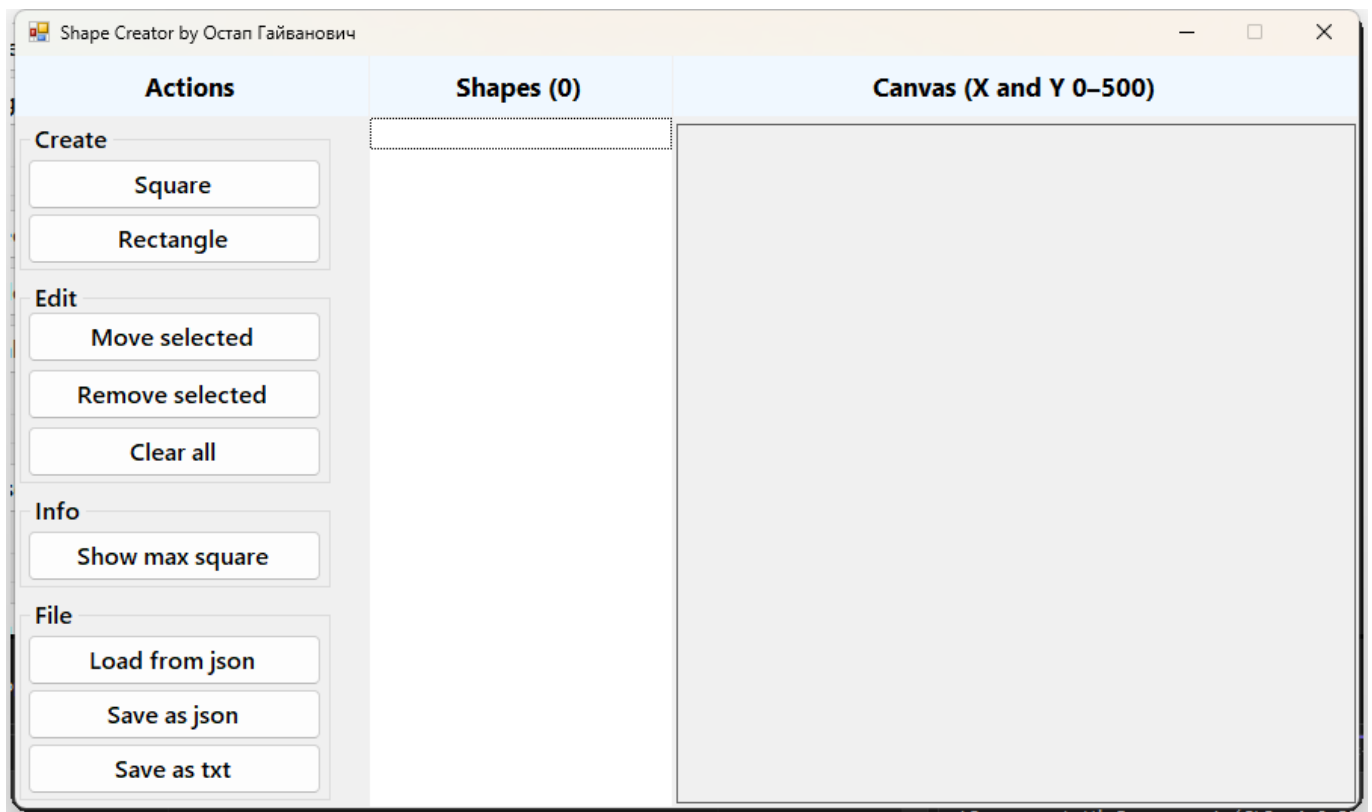


Рис. 6.4 Головна форма програми

6.2.2. Дизайн форми створення фігури

При створенні фігури потрібно вибрати її тип серед доступних зліва, вказати ім'я, координати лівої нижньої точки, сторону (для квадрата) / сторони (для прямокутника), вибрати колір фігури, яким буде відображатися ця фігура на полотні (за замовчуванням колір оранжевий). Також колір відображається у форматі HEX, проте змінити дане поле неможливо.

Внизу форми розташований статусбар, що повідомлятиме різні зауваження: наприклад, не заповнені всі поля, чи застереження про те, що фігура на полотні не буде видима повністю або частково.

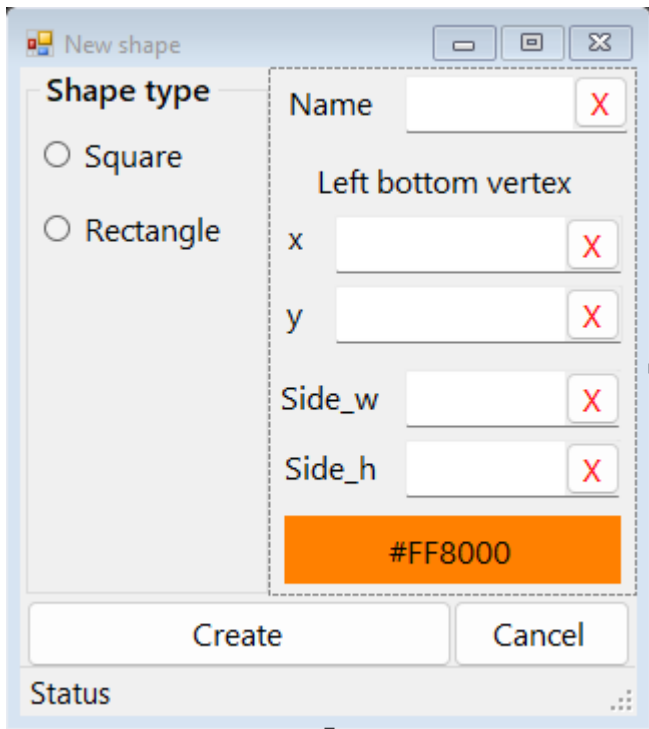
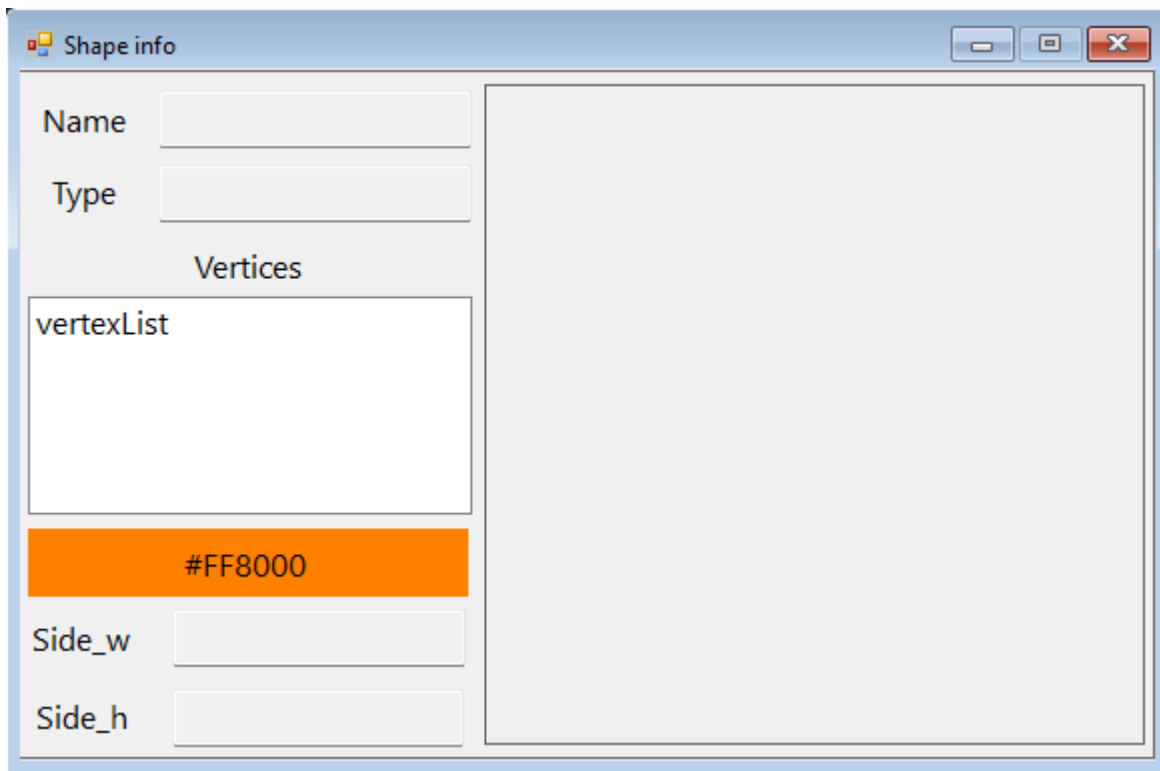


Рис. 6.5 Форма створення фігури

6.2.3. Дизайн форми інформації про фігуру

Виконавши подвійний клік по фігурі зі списку фігур на головній формі, можна викликати дану форму, де покажеться вся інформація про фігуру: ім'я, тип фігури, список вершин, колір фігури (з HEX-кодом) та сторона(и).

Зліва розміщена панель, де намалюється фігура відповідним кольором на весь доступний масштаб, зберігаючи при цьому пропорції сторін.



Shape info

Name

Type

Vertices

vertexList

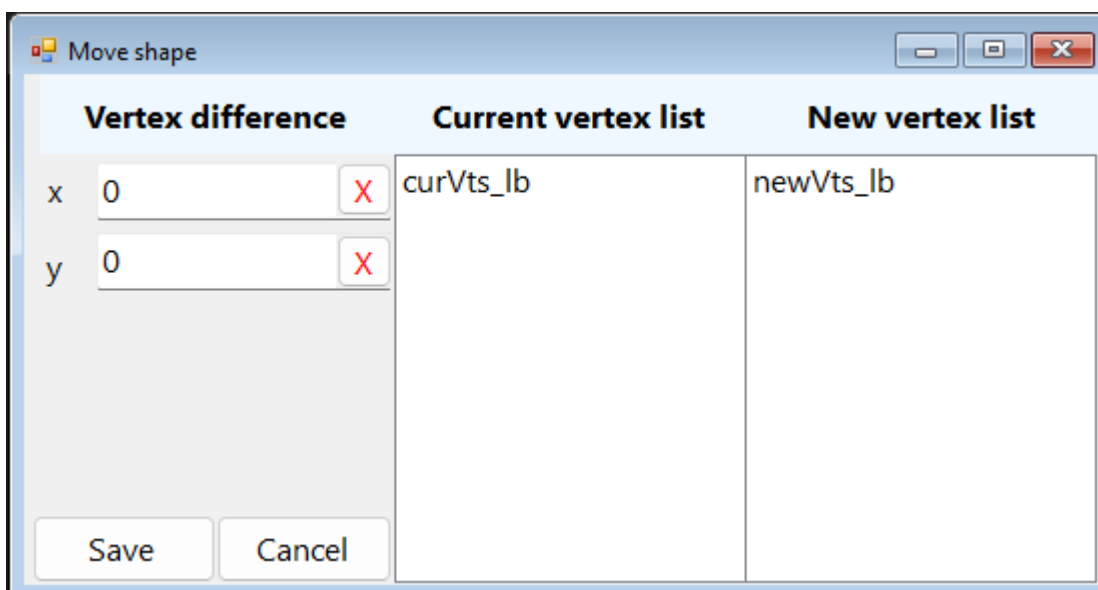
Side_w

Side_h

Рис. 6.6 Форма інформації про фігуру

6.2.4. Дизайн форми інформації про фігуру

Якщо користувач захоче змінити розташування фігури, він на головній формі вибере відповідну дію, і відкриється дана форма. Потрібно буде вказати різницю у координатах; за замовчуванням вона 0. Покажуться поточні координати фігури (*Current vertex list*) та нові (*New vertex list*). За бажанням можна скасувати переміщення фігури.



Move shape

Vertex difference		Current vertex list	New vertex list
x	<input type="text" value="0"/> <input type="button" value="X"/>	curVts_lb	newVts_lb
y	<input type="text" value="0"/> <input type="button" value="X"/>		

Рис. 6.7 Форма переміщення фігури

7. РОЗРОБЛЕННЯ ГРАФІЧНИХ СХЕМ КЛАСІВ

7.1. Діаграма класів програми на C++

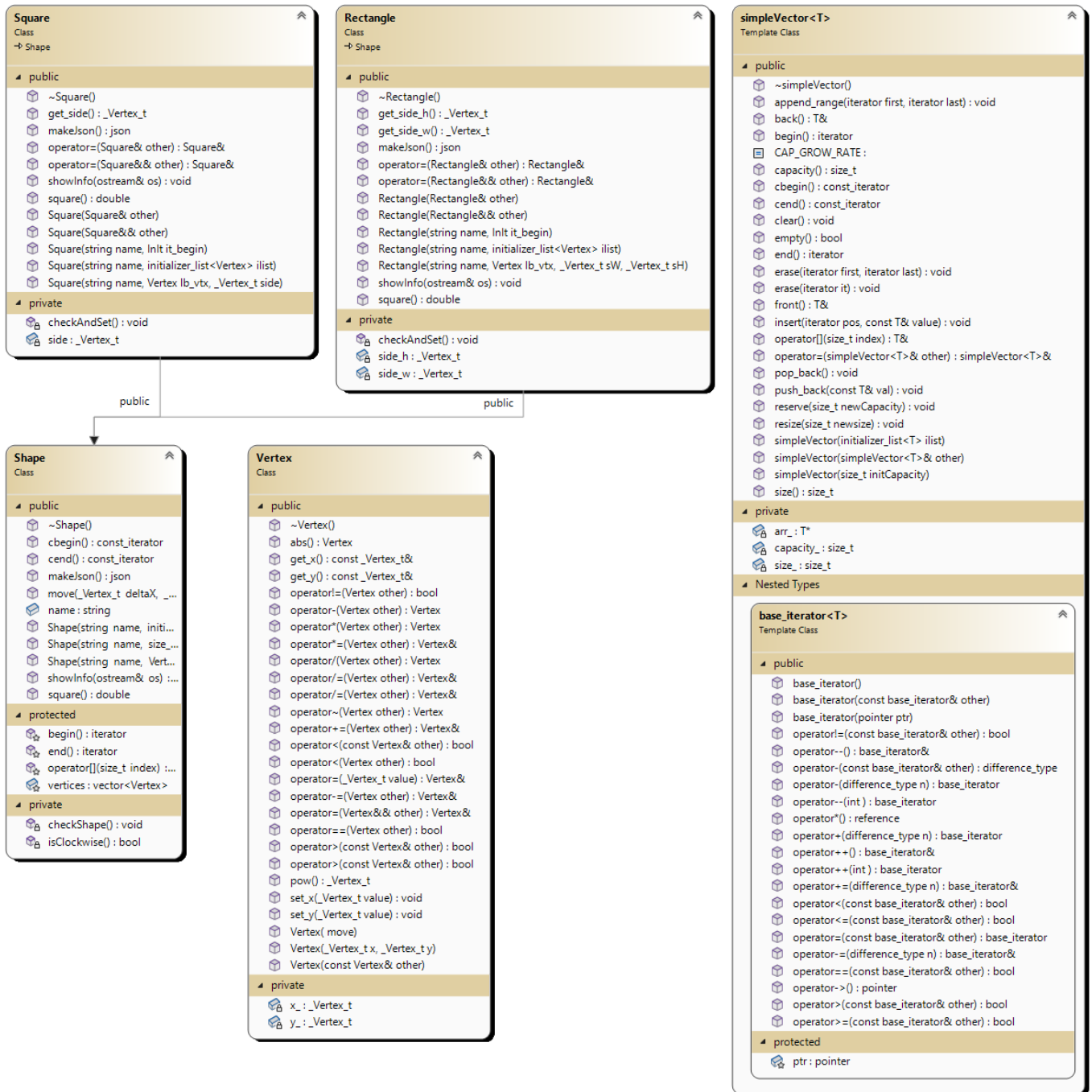


Рис. 7.1 Діаграма всіх класів, використаних в програмі на C++

7.2. Діаграма класів програми на С#

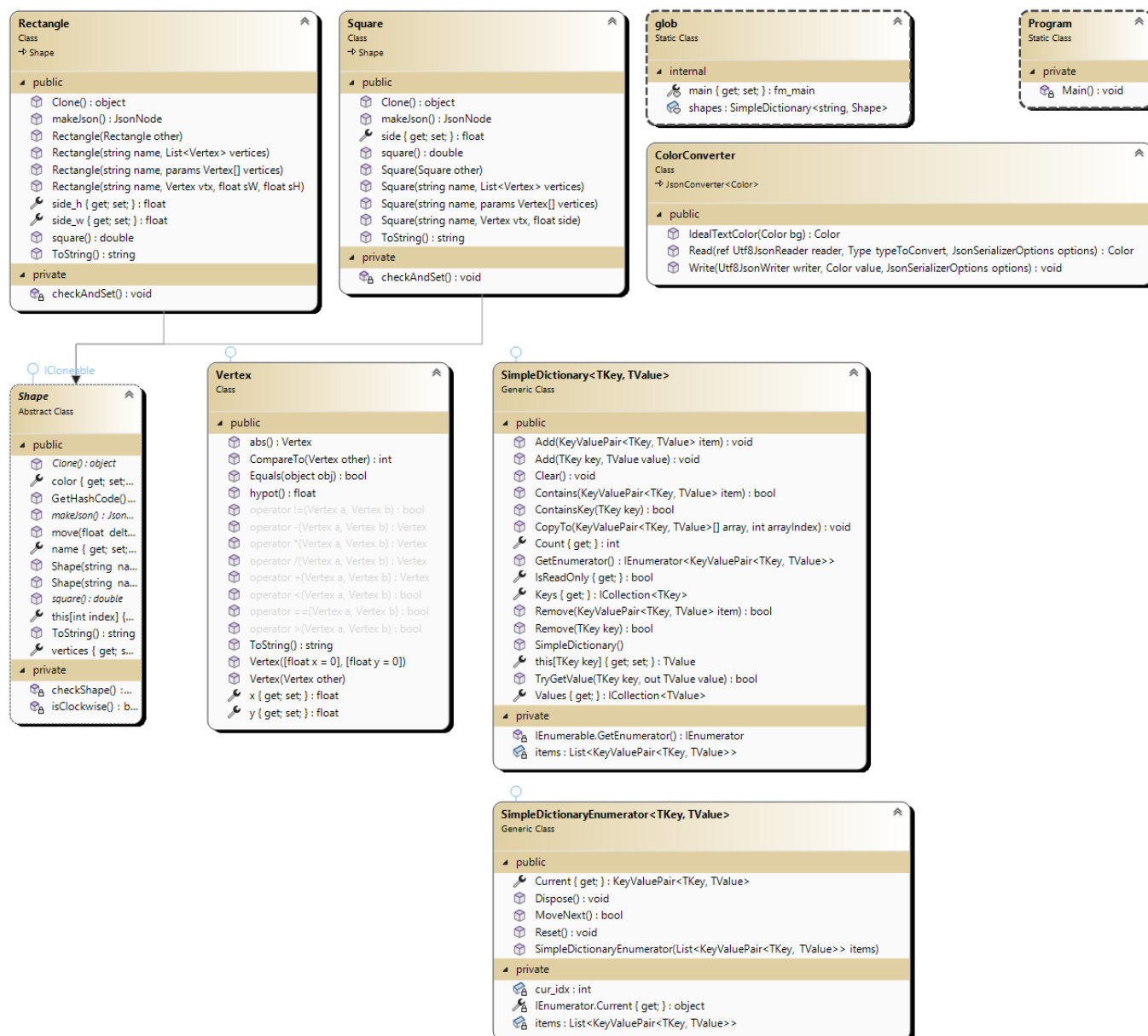


Рис. 7.2 Діаграма усіх класів, окрім форм Windows Forms, використаних в програмі на C#

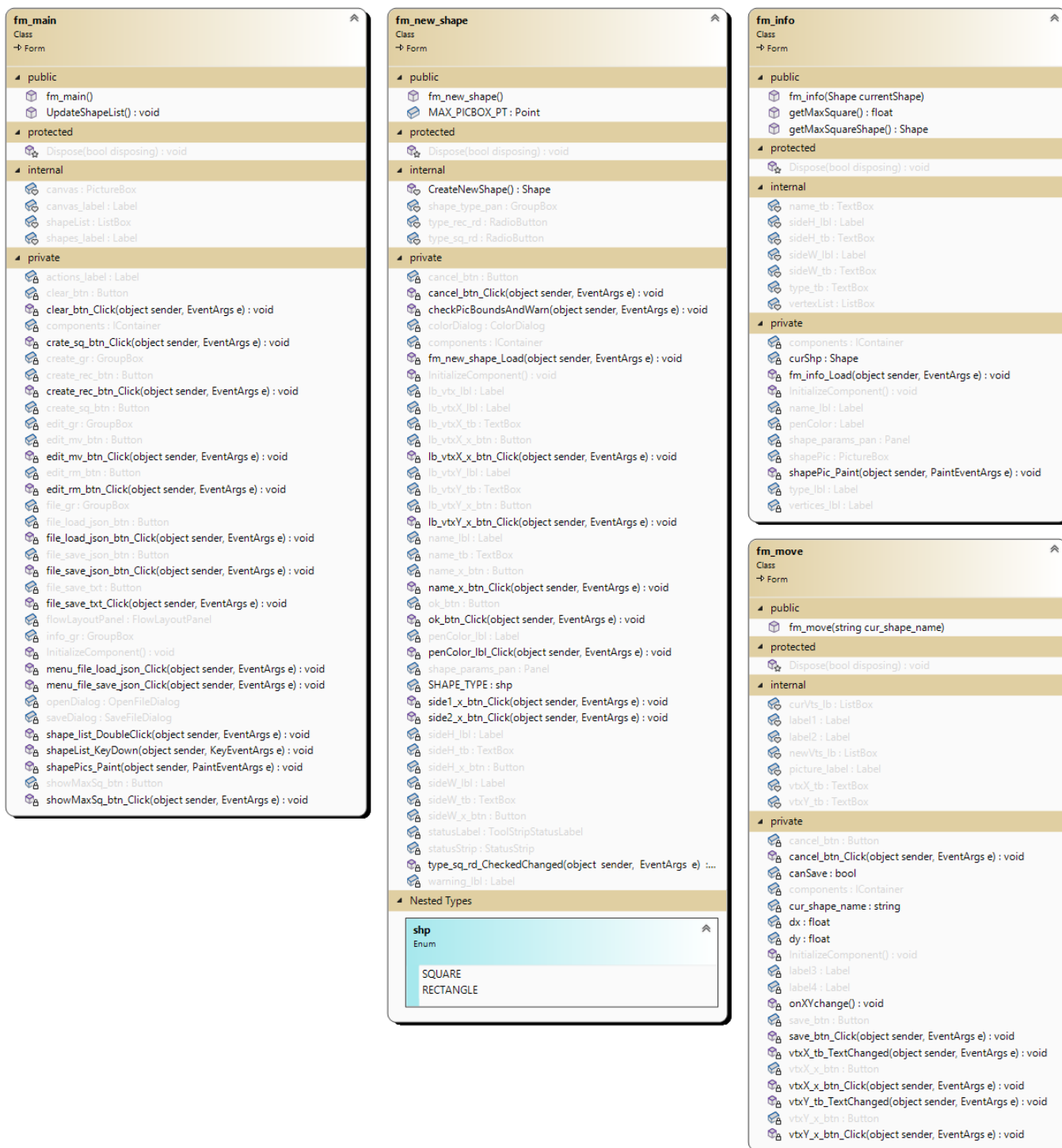


Рис. 7.3 Діаграма класів форм Windows Form

8. РОЗРОБЛЕННЯ ПРОГРАМИ ТА ЇЇ ОПИС

Назва програми. Shape Creator

Призначення програми. Ця програма дозволяє створювати і маніпулювати геометричними фігурами — прямокутниками та квадратами — на площині. Вона надає класи для представлення вершин і фігур, операції для переміщення фігур, обчислення площі, введення/виведення, збереження/зчитування з файлу даних про фігури.

Програма містить контейнер для масиву фігур, який дозволяє знайти фігуру з найбільшою площею, та ітератор для зручної роботи з цим масивом. Реалізовано поліморфізм для роботи з фігурами через батьківський клас, а також читання/запис даних фігур з файлу.

Мови програмування, якими написано програму. C++ та C#

Логічна структура програми. Коротко структуру програми описано в розділі 5. ПРОЄКТУВАННЯ СТРУКТУРИ ПРОГРАМИ. Для глибшого ознайомлення створено діаграми класів, що розташовані в розділі 7. РОЗРОБЛЕННЯ ГРАФІЧНИХ СХЕМ КЛАСІВ.

Вхідні дані. При завантаженні програми можна зчитати масив об'єктів фігур (прямокутників та квадратів) зі спеціально підготовленого JSON-файлу.

Під час роботи програми користувач може ввести координати нових вершин через текстові поля у графічному чи консольному інтерфейсах.

Вихідні дані. Інформація про створені фігури (ім'я фігури, тип, вершини, площі, фігуру з максимальною площею) виводиться у текстові поля графічного чи консольного інтерфейсів.

Дані про масив об'єктів фігур можуть бути збережені у JSON-файл. У такому файлі містяться наступні дані: ім'я фігури, тип фігури, масив вершин фігур з координатами x, y та колір фігури (у випадку програми на C#). Серед іншого, в розділі *info* міститься інформація про кількість фігур та фігуру з найбільшою площею: її ім'я та власне площа.

Результати маніпуляцій з фігурами також записуються в файл текстового формату TXT (лише запис, без можливості зчитування).

На екран виводиться повідомлення про фігуру з найбільшою площею серед створених.

Загалом дані кодуються як текст у форматах JSON (для зберігання/завантаження) та TXT (лише запис). Користувачські дані про вершини вводяться через GUI у текстові поля або через CLI у консоль.

Програмні засоби: Операційна система: 64-бітна Windows 10 або вище.

Інтегровані середовища розробки (IDE): Microsoft Visual Studio зі встановленими пакетами C++ для Desktop та .NET Framework версії 4 з Windows Forms.

Компілятори:

- Microsoft Visual C++ Compiler з підтримкою стандарту C++20 (2020р.)
- .NET Common Language Runtime (CLR)

Бібліотеки:

- Стандартна бібліотека C++ (включно з STL)
- .NET 4 Framework Class Library
- Nlohmann JSON

Редактори коду: вбудовані в Visual Studio 2022 або будь-які інші, наприклад Notepad++

Відлагоджувачі: вбудовані відлагоджувачі Visual Studio 2022

Додаткові інструменти: система контролю версій Git

9. ІНСТРУКЦІЯ КОРИСТУВАЧА

9.1. Інструкція до програми на C++

У папці з програмою виконати exe-файл *ShapeCreator.exe* на ОС Windows 10 або вище. Запуститься консоль, в якій намалюється головне меню програми.

```
      Main menu
c -> Create
e -> Edit
f -> File
i -> Print shape info
m -> Max square
n -> Print shape names
0 -> Exit
Select item: |
```

Рис. 9.1. Головне меню програми

Для початку треба створити хоча б одну фігуру. Для цього треба зайти в підменю *Create* (Створити) ввівши символ *c* та натиснувши *Enter*.

```
Select item: c
      Create
r -> Rectangle
s -> Square
- -> Back
0 -> Exit
Select item: |
```

Рис. 9.2 Підменю *Create*, де можна створити фігури

Вибираємо одну із запропонованих фігур, наприклад це буде *Square* (квадрат) — вводим в консоль *r*. Вводим дані фігури: ім'я, координату лівої нижньої вершини та сторони. Число можуть бути цілими або дробовими. Аналогічно з прямокутником, лише прийдеться вказати дві сторони замість одної.

```
Select item: r
Rectangle
Enter name: Myrect
Enter left bottom vertex (x & y): -3.91 901.52
Enter side_w: 17.4
Enter side_h: 22
```

Рис. 9.3 Приклад створення квадрату

Аналогічним чином можна створити ще декілька фігур; створимо ще квадрат з наступними параметрами.

```
Select item: s
Square
Enter name: Sqr
Enter left bottom vertex (x & y): 0 0
Enter side: 47
```

Рис. 9.4 Створення другої фігури, квадрату

Вводимо дефіс, щоб перейти до батьківського, головного меню. Вибираємо наступну дію: це може бути або вивід інформації про фігури, або одна з опцій редагування фігури. Давайте виведемо інформацію про вже створені фігури. Находячись в головному меню, вводимо символ *i* — бачимо всю доступну інформацію, включаючи вершини, ім'я, сторони, площу, а також і фігуру з максимальною площею.

```
Select item: i
Print shape info
Max square: 2209 (Sqr)

Name: Myrect
Type: Rectangle
Square: 348
Vertices:
(-3.91; 901.52)
(-3.91; 921.52)
(13.49; 921.52)
(13.49; 901.52)
Side_w: 17.4
Side h: 20

Name: Sqr
Type: Square
Square: 2209
Vertices:
(0; 0)
(0; 47)
(47; 47)
(47; 0)
Side: 47
```

Рис. 9.5 Вивід інформації про фігури

Щоб дізнатися лише максимальну площу та фігуру, в головному меню є окремий пункт *Max square*, до якого прив'язаний символ *m*.


```
Select item: m
Max square
Max square: 2209 (Sqr)
```

Рис. 9.6 Показ фігури з максимальною площею

Щоб перемістити фігуру на площині, потрібно зайти в підменю *Edit* та вибрати пункт *Move*. Далі вказуємо індекс фігури та наскільки потрібно змістити кожную з координат. Якщо якусь координату потрібно залишити без змін, вказуємо 0. Можна буде побачити нові координати фігури.

```
Select item: m
Move
1. Myrect
2. Sqr
Select name index: 2
How much the shape will be shifted in x & y: -100 50.5
The new vertices are:
(-103.91; 952.02)
(-103.91; 972.02)
(-86.51; 972.02)
(-86.51; 952.02)
```

Рис. 9.7 Приклад переміщення фігури

Також доступні інші дії в цьому підменю, такі як очистка всіх фігур (*Clear*) та видалення якоїсь конкретної фігури (*Remove*).

У випадку очистки всіх фігур потрібно буде підтвердити цю дію ввівши *y*; при введенні будь-якого іншого символу список не буде очищений.

```
Select item: c
Clear
Do you really want to clear all shapes? [y/n]: n|
```

Рис. 9.8 Підтвердження очистки всіх фігур

Щоб зберегти фігури у файл, находячись в головному меню заходимо в підменю *File* та вибираємо відповідні дію, вказуючи ім'я файлу або ж повний шлях до нього. Застереження: **якщо в шляху буде кирилиця, програма не зможе зберегти файл і видасть помилку.**

```
Select item: f
┌─────────── File ────────────┐
j -> Save as json
l -> Load from json
t -> Save as txt
- -> Back
0 -> Exit
Select item: j
Save as json
Enter path / file name: 2shapes
```

Рис. 9.9 Збереження списку фігур в форматі JSON

Збережений файл матиме наступну структуру:

```
{
  "info": {
    "count": 2,
    "max_square": {
      "shape_name": "Sqr",
      "square": 2209.0
    }
  },
  "shapes": [
    {
      "name": "Myrect",
      "type": "Rectangle",
      "vertices": [
        {
          "x": -103.91000366210938,
          "y": 952.02001953125
        },
        {
          "x": -103.91000366210938,
          "y": 972.02001953125
        },
        {
          "x": -86.51000213623047,
          "y": 972.02001953125
        },
        {
          "x": -86.51000213623047,
          "y": 952.02001953125
        }
      ]
    },
    {
      "name": "Sqr",
      "type": "Square",
      "vertices": [
        {
          "x": 0.0,
          "y": 0.0
        },
        {
          "x": 0.0,
          "y": 47.0
        },
        {
          "x": 47.0,
          "y": 47.0
        },
        {
          "x": 47.0,
          "y": 0.0
        }
      ]
    }
  ]
}
```

Рис. 9.10 Структура файлу JSON

При збереженні файлу в текстовий формат програма запише текст, аналогічний тому, що в пункті виводу інформації про фігуру; за винятком лише відсутності розділювачів темно-синього кольору.

Для зчитування фігур з JSON, потрібно вибрати відповідний файл в тому ж підменю та ввести ім'я файлу / шлях до нього. Якщо програма не зможе прочитати файл, вона видасть помилку. У такому випадку, можливо, такого файлу не існує або він відкритий у іншій програмі.

9.2. Інструкція до програми на C#

Находячись у папці з програмою, запустіть на виконання файл *ShapeCreator_WinForms.exe*. Якщо ОС видає помилку, можливо не встановлений .NET Framework версії 4. Спочатку потрібно створити хоча б одну фігуру.

Находячись на головній формі, вибираємо злів серед групи *Create* одну з фігур. Відкриється наступна форма, де потрібно ввести дані фігури. Зліва біля кожного поля є можливість його очистити, натиснувши на червоний хрестик. Є можливість вибрати колір: потрібно клікнути на текст оранжевого за замовчуванням кольору; відкриється діалог кольору. Тип фігури можна змінити вибравши іншу радіокнопку зліва. Увага:

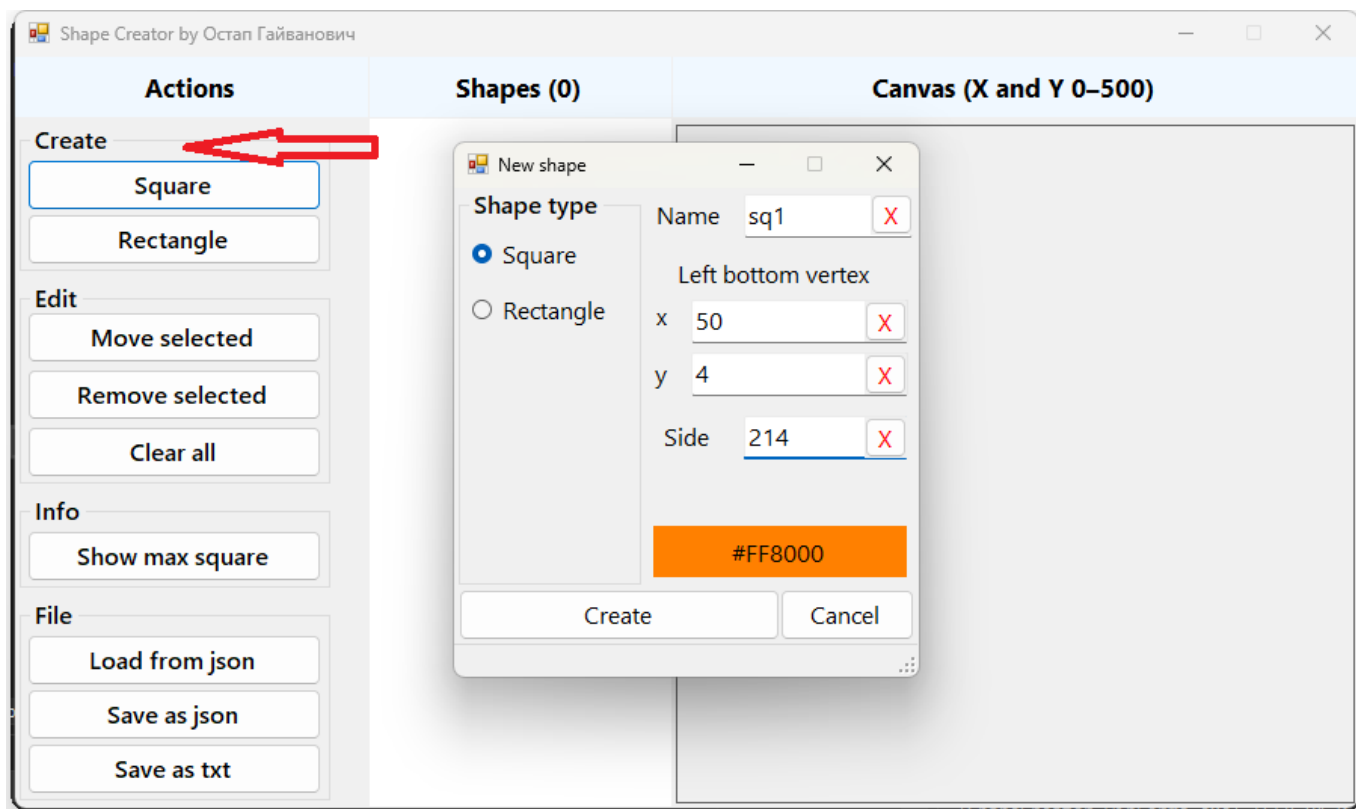


Рис. 9.11 Головна форма та форма створення фігури з введеними для прикладу даними

Створені фігури відображатимуться у списку фігур та малюватимуться на полотні, якщо знаходитимуться в межах в області від 0 до 500 по осях X та Y. Інакше такі

фігури будуть або частково малюватися, або взагалі не будуть. Про це буде повідомлено внизу форми створення фігури, у статусбарі.

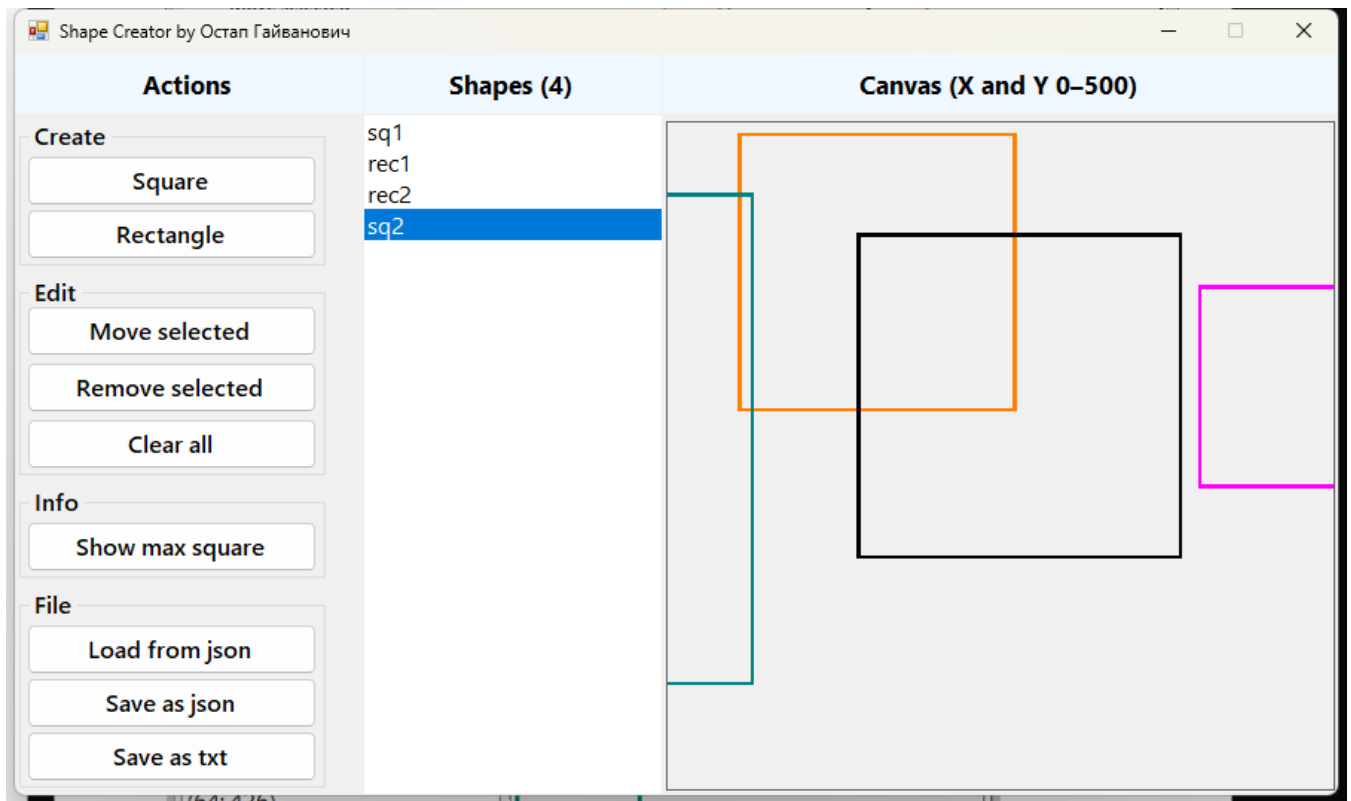


Рис. 9.12 Кілька створених та фігур та їхнє відображення на полотні

Клікнувши два рази по одній з фігур, зможемо побачити інформацію про дану фігуру. Малюється також дана фігура на полотні з максимально видимим масштабом, зберігаючи при цьому всі пропорції сторін.

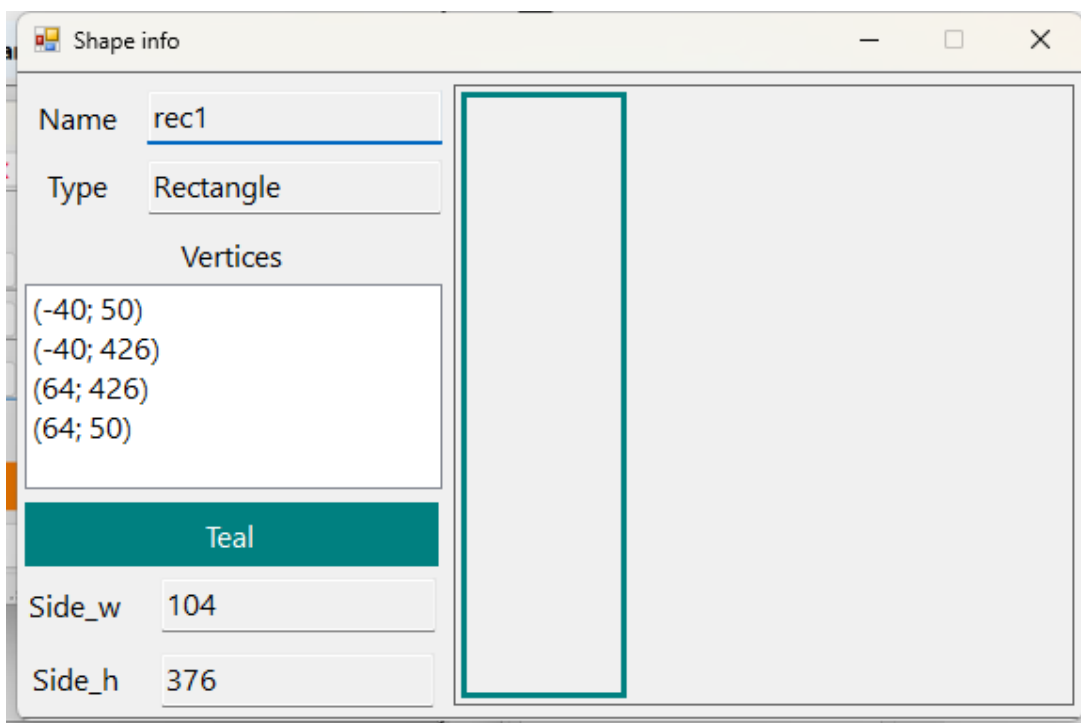


Рис. 9.13 Інформація про вибрану фігуру

Щоб видалити якусь фігуру, потрібно її спочатку вибрати зі списку фігур і натиснути або клавішу *Del* та підтвердити видалення, або вибрати дію *Remove selected*.

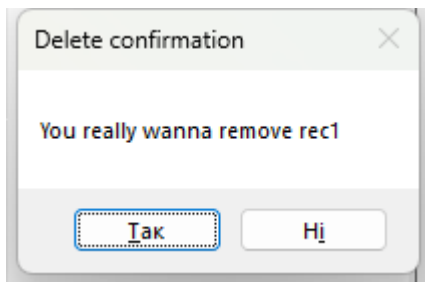


Рис. 9.14 Діалог підтвердження видалення фігури

Дія очищення всіх фігур схожа. З групи *Edit* вибираємо дію *Clear* та підтверджуємо дію.

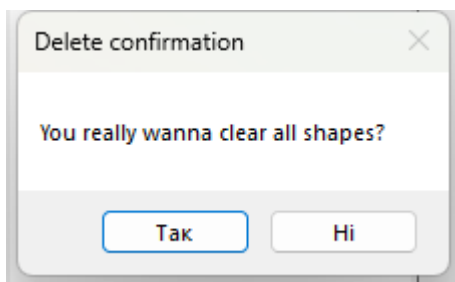


Рис. 9.15 Діалог підтвердження очищення всіх фігур

Щоб отримати інформацію про максимальну площу та ім'я фігури, в головній формі, у групі *Info* вибираємо дію *Show max square*: показується діалогове вікно з такою інформацією.

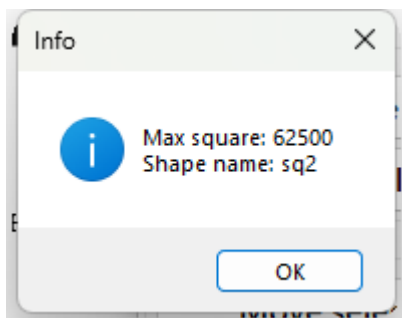


Рис. 9.15 Діалог з інформацією про максимальну площу фігури

Якщо знадобиться перемістити фігуру, за аналогією вибираємо відповідну дію (*Move selected*) та вказуємо різницю в координатах. Додатна різниця збільшить відповідну координату, від'ємна, навпаки, зменшить. Якщо координату потрібно залишити, вказуємо 0.

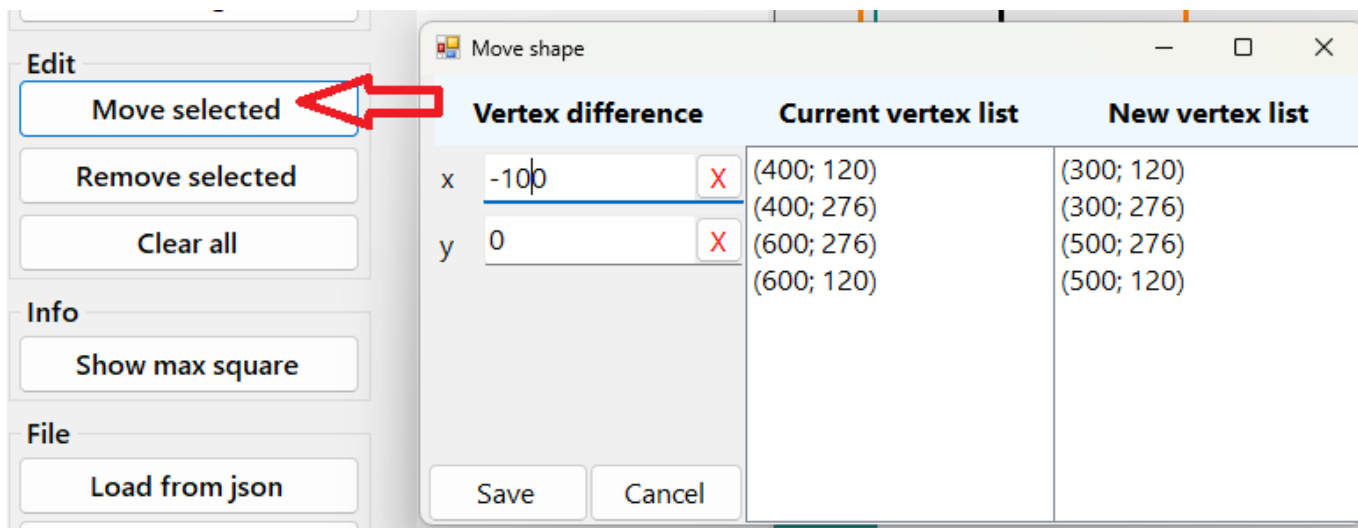


Рис. 9.16 Приклад переміщення фігури

Для збереження фігур у файл чи їх зчитування у списку дій є група *File*, де можна вибрати відповідну дію. При виборі таких дій відкриється діалогове вікно відкриття чи збереження, де потрібно буде вибрати потрібний файл чи ввести назву нового файлу. Зчитати можливо лише з формату JSON, як це видно на малюнку. Записати можна і в JSON, і в TXT.

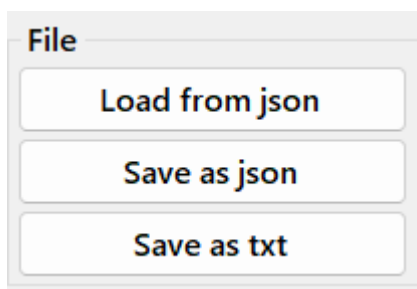


Рис. 9.17 Група дій File для роботи з файлами

10. КОНТРОЛЬНИЙ ПРИКЛАД ТА АНАЛІЗ РЕЗУЛЬТАТІВ КОМП'ЮТЕРНОЇ РЕАЛІЗАЦІЇ ПРОГРАМИ

10.1. Контрольний приклад програми на C++

10.2. Контрольний приклад програми на C#

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ