**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-18-2

Спеціальності 121 Інженерія програмного забезпечення

Чакова В. В.

(прізвище та ініціали)

Керівник

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

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

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

(підпис) (прізвище та ініціали)

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

(підпис) (прізвище та ініціали)

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

(підпис) (прізвище та ініціали)

м. Дніпро, 2020 р.

Зміст

[1. ПОСТАНОВКА ЗАДАЧІ 3](#_Toc38558143)

[2. ОПИС РОЗВ'ЯЗКУ 3](#_Toc38558144)

[2.1 Патерн Composite 3](#_Toc38558145)

[2.2. Патерн Prototype 4](#_Toc38558146)

[2.3. Патерн Singleton 4](#_Toc38558147)

[2.4. Патерн Memento 4](#_Toc38558148)

[3. ВИХІДНИЙ ТЕКСТ ПРОГРАМИ РОЗВ'ЯЗКУ ЗАДАЧІ 5](#_Toc38558149)

[4. ОПИС ІИНТЕРФЕЙСУ (КЕРІВНИЦТВО КОРИСТУВАЧА) 8](#_Toc38558150)

[5. ОПИС ТЕСТОВИХ ПРИКЛАДІВ 9](#_Toc38558151)

[6. АНАЛІЗ ПОМИЛОК 12](#_Toc38558152)

# 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.

# ОПИС РОЗВ'ЯЗКУ

## 2.1 Патерн Composite

Aggregate – це клас, що містить вектор покажчиків на тип Figure. Цей вектор містить фігури, які унаслідуються від типу Figure. Також у композиті перевизначені віртуальні функції класів простих фігур (Circle та Rectangle) таким чином, щоб кожна фігура з вектору figures викликала відповідну функцію «на самій собі» зі свого класу. Наприклад,

void Aggregate::changeColor()

{

for (auto& figure : figures)

{

figure->changeColor();

}

}

Тобто агрегат є такою самою фігурою, як, наприклад, фігура Rectangle. Також клієнт має можливість звертатися до об’єкта типу Aggregate через покажчик Figure\*.

## Патерн Prototype

У класі Figure оголошена чиста virtual функція: virtual Figure\* clone() = 0; яка обов'язково перевизначена в класах-наслідовниках так, щоб повертати вказівник на новий об'єкт, який є копією об’єкта, на якому вона була викликана.

## Патерн Singleton

З інтерфейсом працює клас Controller. Можна створити лише один об’єкт цього класу, и ось що я зробив для цього:

1. Оголошено статичний метод getState, що використовувається для отримання об'єкта і створює його під час першого виклику метода.
2. Конструктор класа – private.
3. У клієнтському коді замість прямого виклику конструктора, викликається метод getState.

Наприклад, работа з інтерфейсом, через getState викликаємо createRectangle

//1 - Create Rectangle

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Num1)

{

Controller::getState().createRectangle();

}

## Патерн Memento

Класи FigureMemento та AggregateMemento відповідають за збереження стану фігури та агрегату.

Хазяїном виступають класи Сircle, Rectangle та Aggregate відповідно, які містять функції:

1. для збереження свого стану у Memento (у клас FigureMemento / AggregateMemento).
2. для відтворення свого стану з Memento (FigureMemento / AggregateMemento).

Клас AggregateCaretaker бере на себе роль Опікуна та відтворює з файлу та зберігає у нього знімок об'єктов, при тому не проводить жодних операцій над знімком та не має уяви про його внутрішній зміст. Має він дві функції:

1. saveState(), яка робить знімок, та зберігає його у файл.
2. restoreState(), яка відтворює фігури з файлу за допомогою паттерну Memento.

Коли користувач робить знімок, усі фігури на екрані, записуються до файлу.

Коли користувач відновлює стан з Memento, відтворюється прогрес програми на момент створення знімку.

# ВИХІДНИЙ ТЕКСТ ПРОГРАМИ РОЗВ'ЯЗКУ ЗАДАЧІ

Патерн Composite

class Aggregate: public Figure

{//----------------------------------COMPOSITE------------------------||

private:

std::vector<Figure\*> figures;

friend class AggregateMemento;

public:

Aggregate();

~Aggregate();

std::vector<PropertiesOfFigure\*> getFigures();

int containsFigure(Figure\* figure);

int containsType(std::string type);

bool addFigure(Figure\* figure);

Figure\* getFigure(int index);

FigureMemento\* save();

void restore(FigureMemento\* memento);

// Color Changing

void changeColor();

// Type

std::string getType();

// Movement

void move(float speed, int direction);

void autoMove(float speed);

// Cloning

Figure\* clone();

};

Патерн Prototype/Menento

/----------------------------------------PROTOTYPE------------------------||

// Клонирование фигуры (паттерн - Прототип)

virtual Figure\* clone() = 0;

//--------------------------MEMENTO-------------------------||

class AggregateMemento: private FigureMemento

{

private:

std::string data;

//Aggregate info > file < file

std::string serializeAggregate(Aggregate\* \_aggregate);

Aggregate\* deserializeAggregate(std::stringstream& stream);

public:

AggregateMemento(std::string \_data);

AggregateMemento(Aggregate\* \_aggregate);

Figure\* getFigure();

std::string getData();

};

class AggregateCaretaker

{

private:

const std::string NAMES\_FILE = "names.txt";

public:

AggregateCaretaker();

std::vector<Aggregate\*> restoreState();

void saveState(std::vector<Aggregate\*> aggregates);};

//----------------------MEMENTO------------------||

class FigureMemento

{

private:

std::string data;

protected:

std::string serializeFigure(Figure\* figure);

Figure\* deserializeFigure(std::string \_data);

FigureMemento();

public:

FigureMemento(Figure\* figure);

FigureMemento(std::string data);

virtual Figure\* getFigure();

virtual std::string getData();

};

Патерн Singleton

class Controller

{

private:

std::vector<Aggregate\*> aggregates;

Aggregate\* selected\_aggregate;

bool is\_aggregated;

bool is\_delete;

const std::string FILE = "list.txt";

Controller();

Controller(const Controller&);

~Controller();

Controller\* operator=(Controller&);

void checkTouch();

Aggregate\* getSelectedAggregate(float mouse\_position\_x, float mouse\_position\_y);

Figure\* getSelectedFigure(float mouse\_position\_x, float mouse\_position\_y, Aggregate\* aggregate);

void generateFigure(std::string type);

std::vector<std::string> getAggregatesNames();

void saveAggregatesNames();

void doAggregation(float mouse\_position\_x, float mouse\_position\_y);

void selectAggregate(float mouse\_position\_x, float mouse\_position\_y);

public:

**static Controller& getState();**

**…**

**}**

//---------------------------------SINGLETON----------------------------------------||

Controller& Controller::getState()

{

static Controller state;

return state;

}

# ОПИС ІИНТЕРФЕЙСУ (КЕРІВНИЦТВО КОРИСТУВАЧА)

При запуску програми відкривається вікно з чорним фоном, з яким і буде взаємодіяти користувач. Далі можна побачити клавіші для управління сценою:

* ЛКМ – обрати фігуру курсором.
* ПКМ – почати агрегірованіє.
* 1 – створити коло.
* 2 – створити квадрат
* ↑ ↓ ← → – переміщувати об’єкт.
* F5 – зробити знімок.
* F9 – відновити стан зі знімка.
* P – зробити копію активного об’єкта.
* Q – змініти колір фігури.

# ОПИС ТЕСТОВИХ ПРИКЛАДІВ

Створюємо дві фігури та об’єднуємо іх у один агрегат.

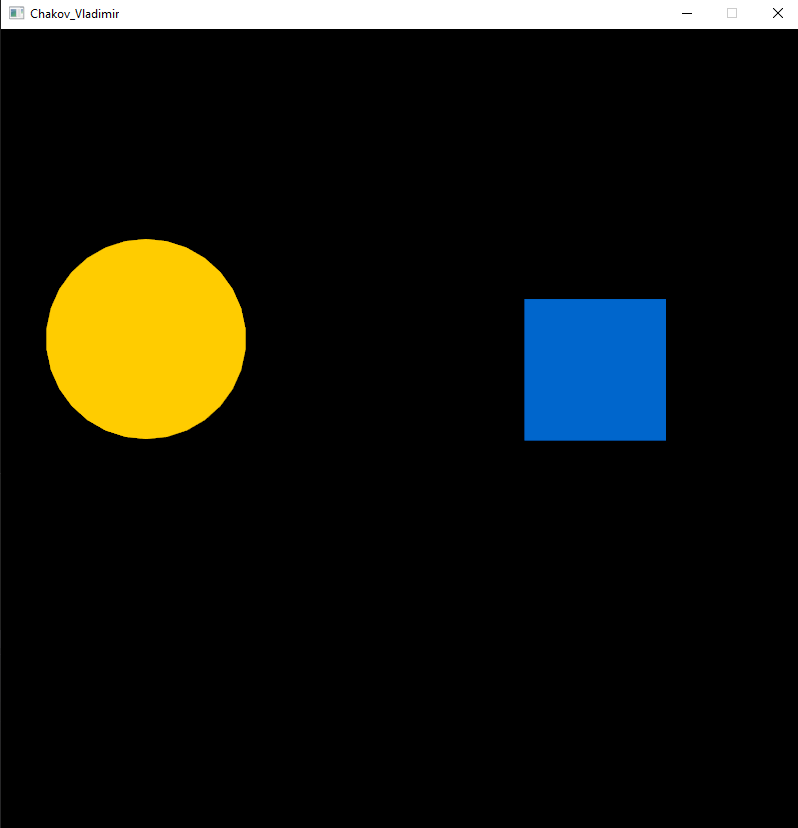


Рисунок 1

Рухаються разом.

Створимо ще один агрегат та об’єднаємо його з минулим.

Тепер два агрегата стали одним. Рухаються разом.

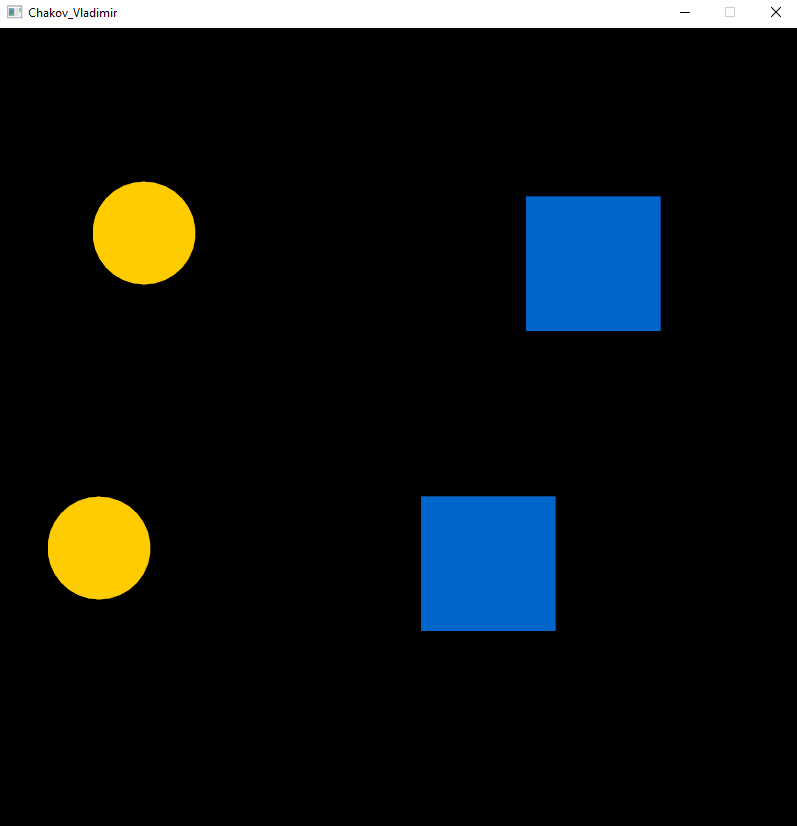


Рисунок 2

Зробимо копію агрегату за допомогою патерну Prototype

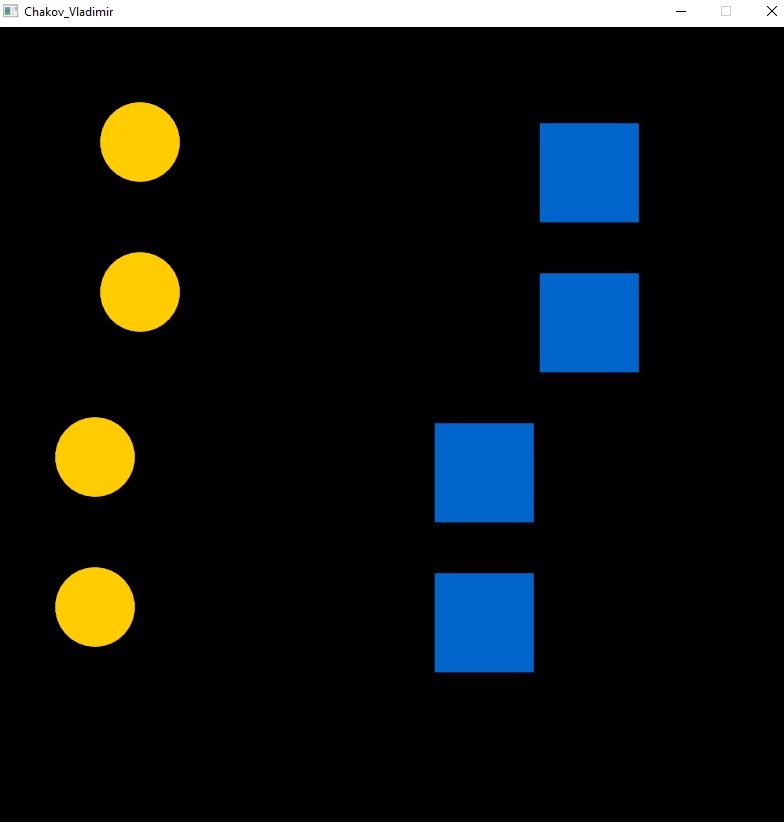


Рисунок 3

Клонуємо агрегат, бачимо результат клонування.

Зробимо знімок. Вікно до знімку зображено на рис. 4.

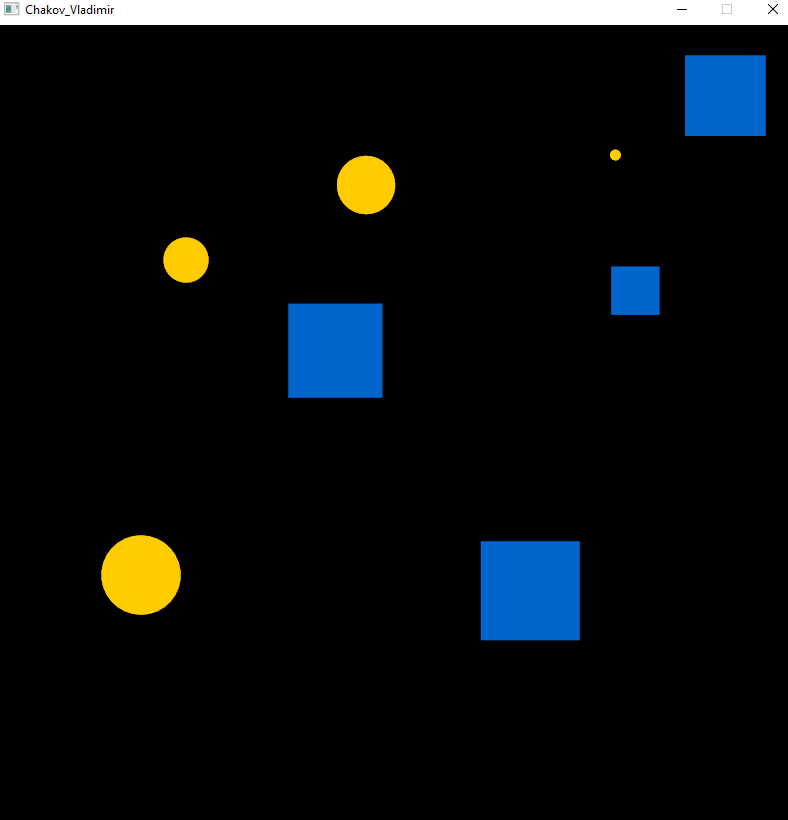


Рисунок 4

Додамо фігур та змінимо кольора.

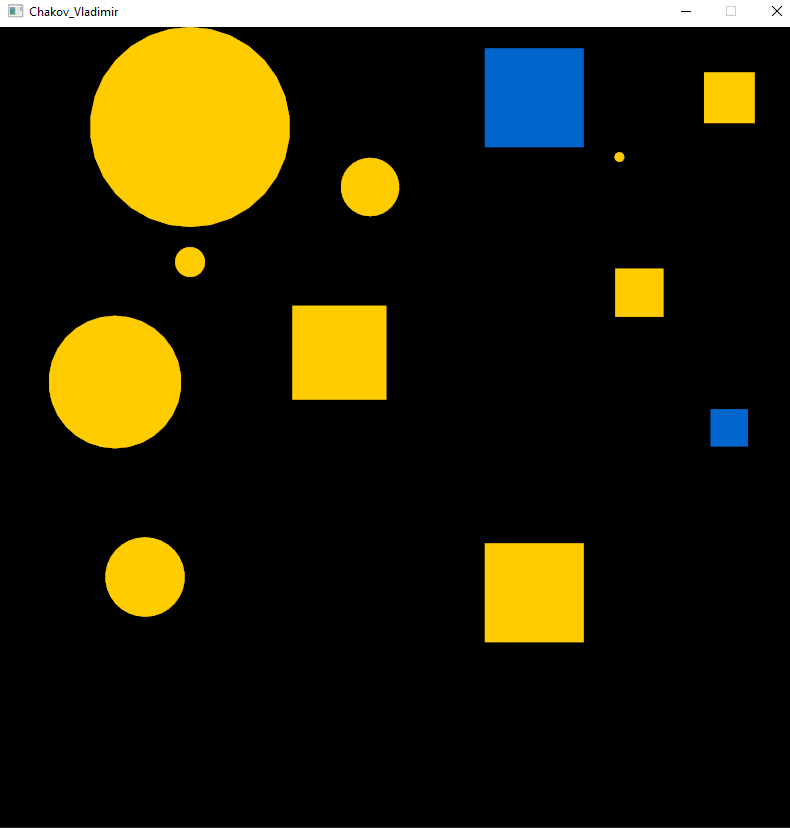


Рисунок 5

Відновимо стан з файлу куди було завантажено знімок.

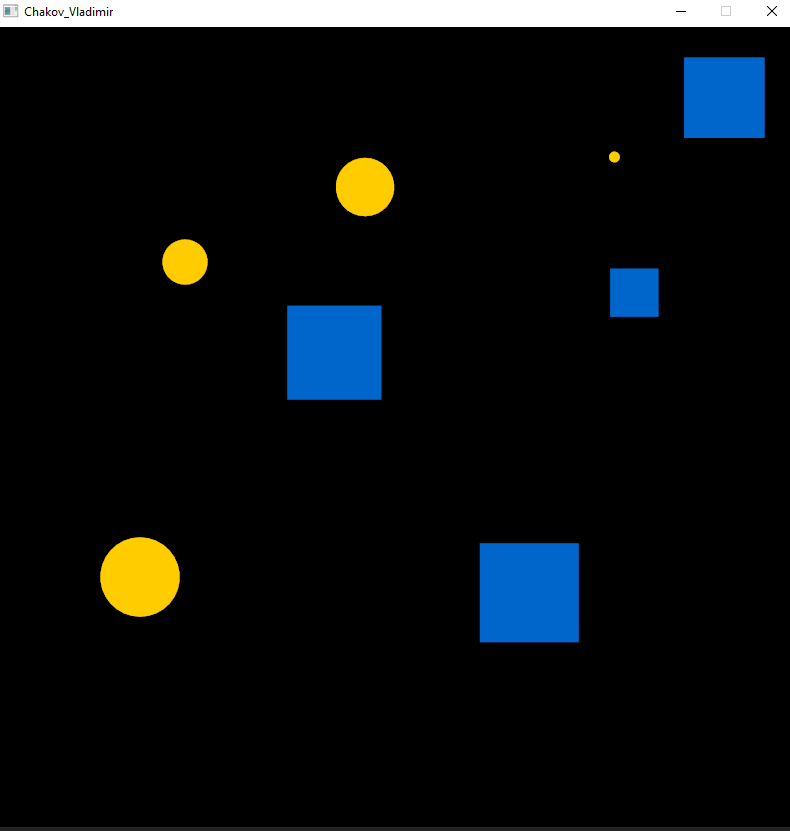


Рисунок 6

Зміст одного з файлів для зберігання:

Aggregate

Rectangle 100.000000 0 102 204 534.289307 565.710693 0.698337

Aggregate

Circle 100.000000 255 204 0 145.000000 550.000000 0.397214

/Aggregate

Aggregate

Rectangle 100.000000 0 102 204 639.289307 265.710693 0.340562

Aggregate

Circle 100.000000 255 204 0 190.000000 235.000000 0.225935

/Aggregate

/Aggregate

/Aggregate

1. АНАЛІЗ ПОМИЛОК