

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій

Кафедра ІОСУ

Лабораторна робота № 6

З дисципліни “Сучасні парадигми програмування”

Виконав
Студент групи КН-11
Стрижак В. М.

Тернопіль
2024 рік

Тема: Реалізація принципу поліморфізму в ООП. Перевантаження операцій..

Мета роботи: засвоїти принципи перевантаження операцій, навчитися створювати ієрархії класів та використовувати поліморфізм

Завдання: Ієрархія класів. (23)

Спроектуйте ієрархію класів для представлення графічних об'єктів. Головним базовим класом для усіх об'єктів є клас Point - точка на площині (у просторі) з її координатами. Опис класів слід розмістити у заголовочному файлі, а визначення функцій і головну функцію програми - в двох окремих файлах. Передбачте методи для створення об'єкта, його переміщення на екрані, зміни розмірів та кольору, обертання на заданий кут. Використайте захищення даних для ізоляції елементів даних класу від підпрограм, в яких цей клас використовується, а також поліморфізм для визначення дії певних функцій у класовій ієрархії. Напишіть головну функцію, що демонструє роботу з цим класом. Програма повинна містити меню, що дозволяє здійснити перевірку всіх методів класу.

Завдання: Перевантаження операторів (8)

Описати клас, що реалізовує вказаний нижче тип даних. Клас повинен містити множину конструкторів для створення об'єктів певного типу (конструктор по замочуванню та з параметрами конструктор копії) та подані у таблиці операції над об'єктами класу (плюс обов'язково операції присвоювання та порівняння) з використанням механізму перевантаження операцій:

Хід роботи

Варіант 23.

1. Створити файл main.cpp та shapes.h

Код програми в main.cpp:

```
#include <iostream>
#include "shapes.h"

void displayMenu() {
    std::cout << "1. Create Circle\n";
    std::cout << "2. Create Rectangle\n";
    std::cout << "3. Create Triangle\n";
    std::cout << "4. Create Pentagon\n";
}
```

```

std::cout << "5. Move Object\n";
std::cout << "6. Resize Object\n";
std::cout << "7. Change Color\n";
std::cout << "8. Rotate Object\n";
std::cout << "9. Draw Object\n";
std::cout << "0. Exit\n";
}

int main() {
    Shape* shape = nullptr;
    int choice;
    double x, y, radius, width, height, side, factor, angle;
    std::string color;

    do {
        displayMenu();
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                std::cout << "Enter x, y, radius, and color: ";
                std::cin >> x >> y >> radius >> color;
                shape = new Circle(x, y, radius, color);
                break;
            case 2:
                std::cout << "Enter x, y, width, height, and color: ";
                std::cin >> x >> y >> width >> height >> color;
                shape = new Rectangle(x, y, width, height, color);
                break;
            case 3:
                std::cout << "Enter x, y, base, height, and color: ";
                std::cin >> x >> y >> width >> height >> color;
                shape = new Triangle(x, y, width, height, color);
                break;
            case 4:
                std::cout << "Enter x, y, side, and color: ";
                std::cin >> x >> y >> side >> color;
                shape = new Pentagon(x, y, side, color);
                break;
            case 5:
                if (shape) {
                    std::cout << "Enter dx and dy: ";
                    std::cin >> x >> y;
                    shape->move(x, y);
                }
                else {
                    std::cout << "No shape created yet.\n";
                }
                break;
            case 6:
                if (shape) {
                    std::cout << "Enter resize factor: ";
                    std::cin >> factor;
                    shape->resize(factor);
                }
                else {
                    std::cout << "No shape created yet.\n";
                }
                break;
            case 7:
                if (shape) {
                    std::cout << "Enter new color: ";
                    std::cin >> color;
                    shape->changeColor(color);
                }
                else {
                    std::cout << "No shape created yet.\n";
                }
                break;
        }
    }
}

```

```

        case 8:
            if (shape) {
                std::cout << "Enter rotation angle: ";
                std::cin >> angle;
                shape->rotate(angle);
            }
            else {
                std::cout << "No shape created yet.\n";
            }
            break;
        case 9:
            if (shape) {
                shape->draw();
            }
            else {
                std::cout << "No shape created yet.\n";
            }
            break;
        case 0:
            std::cout << "Exiting...\n";
            break;
        default:
            std::cout << "Invalid choice. Please try again.\n";
            break;
    }
} while (choice != 0);

delete shape;
return 0;
}

```

Код програми в shapes.h:

```

#ifndef SHAPES_H
#define SHAPES_H

#include <iostream>
#include <cmath>
#include <vector>

class Point {
protected:
    double x, y;

public:
    Point(double x = 0, double y = 0) : x(x), y(y) {}
    virtual ~Point() = default;

    void move(double dx, double dy) {
        x += dx;
        y += dy;
    }

    virtual void draw() const = 0;
    virtual void resize(double factor) = 0;
    virtual void rotate(double angle) = 0;
    virtual void changeColor(const std::string& newColor) = 0;
};

class Shape : public Point {
protected:
    std::string color;

public:
    Shape(double x = 0, double y = 0, const std::string& color = "black")
        : Point(x, y), color(color) {}

    void changeColor(const std::string& newColor) override {

```

```

        color = newColor;
    }

    virtual void draw() const override = 0;
    virtual void resize(double factor) override = 0;
    virtual void rotate(double angle) override = 0;
};

class Circle : public Shape {
private:
    double radius;

public:
    Circle(double x = 0, double y = 0, double radius = 1.0, const std::string& color
= "black")
        : Shape(x, y, color), radius(radius) {}

    void draw() const override {
        std::cout << "Drawing Circle at (" << x << ", " << y << ") with radius " <<
radius
            << " and color " << color << std::endl;
    }

    void resize(double factor) override {
        radius *= factor;
    }

    void rotate(double angle) override {
        // Rotation doesn't change a circle's appearance
    }
};

class Rectangle : public Shape {
private:
    double width, height;

public:
    Rectangle(double x = 0, double y = 0, double width = 1.0, double height = 1.0,
const std::string& color = "black")
        : Shape(x, y, color), width(width), height(height) {}

    void draw() const override {
        std::cout << "Drawing Rectangle at (" << x << ", " << y << ") with width "
<< width
            << ", height " << height << " and color " << color << std::endl;
    }

    void resize(double factor) override {
        width *= factor;
        height *= factor;
    }

    void rotate(double angle) override {
        // Rotation logic for rectangle
    }
};

class Triangle : public Shape {
private:
    double base, height;

public:
    Triangle(double x = 0, double y = 0, double base = 1.0, double height = 1.0,
const std::string& color = "black")
        : Shape(x, y, color), base(base), height(height) {}

    void draw() const override {
        std::cout << "Drawing Triangle at (" << x << ", " << y << ") with base " <<
base
            << " and height " << height << " and color " << color << std::endl;
    }
};

```

```

    }

    void resize(double factor) override {
        base *= factor;
        height *= factor;
    }

    void rotate(double angle) override {
        // Rotation logic for triangle
    }
};

class Pentagon : public Shape {
private:
    double side;

public:
    Pentagon(double x = 0, double y = 0, double side = 1.0, const std::string& color
= "black")
        : Shape(x, y, color), side(side) {}

    void draw() const override {
        std::cout << "Drawing Pentagon at (" << x << ", " << y << ") with side " <<
side
            << " and color " << color << std::endl;
    }

    void resize(double factor) override {
        side *= factor;
    }

    void rotate(double angle) override {
        // Rotation logic for pentagon
    }
};

#endif // SHAPES_H

```

Результат коду:

```

1. Create Circle
2. Create Rectangle
3. Create Triangle
4. Create Pentagon
5. Move Object
6. Resize Object
7. Change Color
8. Rotate Object
9. Draw Object
0. Exit
Enter your choice: |

```

Варіант 8 – дроби (віднімання, множення).

Код програми:

```

#include <iostream>
#include <stdexcept>

```

```

class Fraction {
private:
    int numerator;
    int denominator;

public:
    Fraction(int numerator = 0, int denominator = 1) {
        if (denominator == 0) {
            throw std::invalid_argument("Denominator cannot be zero");
        }
        this->numerator = numerator;
        this->denominator = denominator;
    }

    Fraction(const Fraction& other) : numerator(other.numerator),
denominator(other.denominator) {}

    Fraction operator+(const Fraction& other) const {
        int newNumerator = numerator * other.denominator + other.numerator *
denominator;
        int newDenominator = denominator * other.denominator;
        return Fraction(newNumerator, newDenominator);
    }

    Fraction operator-(const Fraction& other) const {
        int newNumerator = numerator * other.denominator - other.numerator *
denominator;
        int newDenominator = denominator * other.denominator;
        return Fraction(newNumerator, newDenominator);
    }

    Fraction operator*(const Fraction& other) const {
        int newNumerator = numerator * other.numerator;
        int newDenominator = denominator * other.denominator;
        return Fraction(newNumerator, newDenominator);
    }

    Fraction& operator=(const Fraction& other) {
        numerator = other.numerator;
        denominator = other.denominator;
        return *this;
    }

    bool operator==(const Fraction& other) const {
        return numerator == other.numerator && denominator == other.denominator;
    }

    bool operator!=(const Fraction& other) const {
        return !(*this == other);
    }

    friend std::ostream& operator<<(std::ostream& os, const Fraction& fraction) {
        return os << fraction.numerator << "/" << fraction.denominator;
    }
};

int main() {
    Fraction f1(1, 2);
    Fraction f2(3, 4);

    Fraction f3 = f1 + f2;
    std::cout << "f3 = " << f3 << std::endl;

    Fraction f4 = f1 - f2;
    std::cout << "f4 = " << f4 << std::endl;

    Fraction f5 = f1 * f2;
    std::cout << "f5 = " << f5 << std::endl;
}

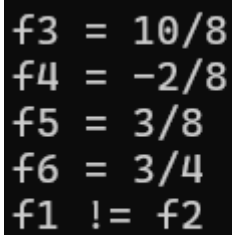
```

```
Fraction f6 = f1;
f6 = f2;
std::cout << "f6 = " << f6 << std::endl;

if (f1 == f2) {
    std::cout << "f1 == f2" << std::endl;
}
else {
    std::cout << "f1 != f2" << std::endl;
}

return 0;
}
```

Результат коду:



```
f3 = 10/8
f4 = -2/8
f5 = 3/8
f6 = 3/4
f1 != f2
```

Висновок: Я засвоїв принципи перевантаження операції та навчився створювати ієрархії класів та використовувати поліморфізм.