Технологии конструирования программного обеспечения

Отчет по экзаменационной работе № 02

Группа: 221-329 **Студент:** Минчаков Аркадий Сергеевич

Задание на экзаменационную работу:

Паттерн «Посетитель». Имеется несколько геометрических фигур (окружность, прямоугольник, квадрат, треугольник... - не менее 3). Написать приложение, реализующее вычисление периметра и площади этих фигур. Для устранения тесной связи между алгоритмами и объектами использовать паттерн «Посетитель».

UML-диаграмма классов:

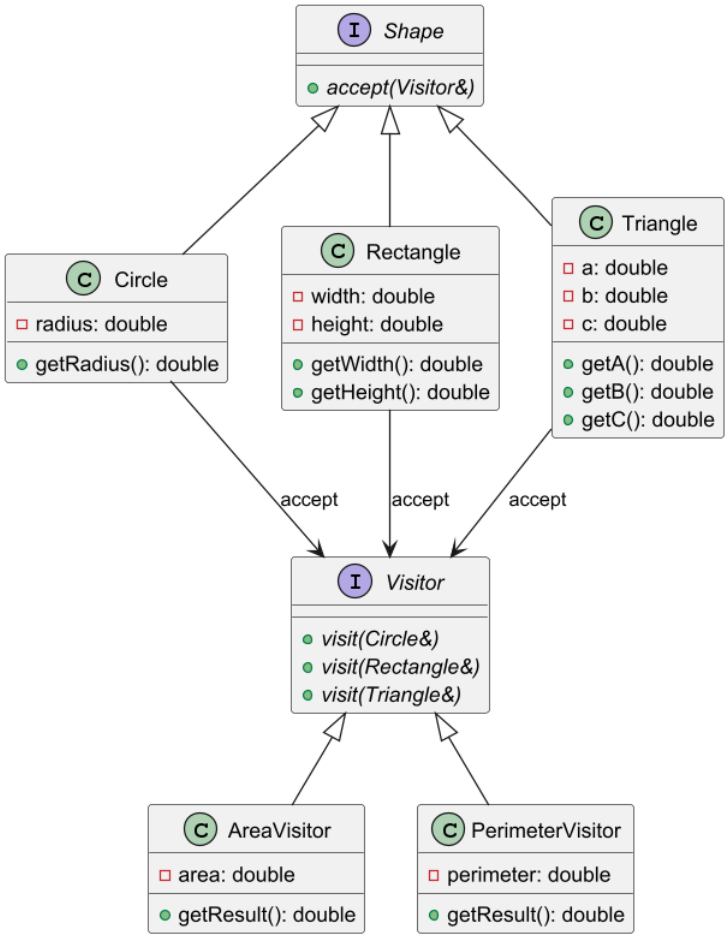

Figure 1: UML-диаграмма классов

Исходный код программы:

main.cpp

```
/*
Программа реализует паттерн "Посетитель" для вычисления площади и периметра геометрических фигур.
Автор: Минчаков Аркадий Сергеевич, группа 221-329

Идея реализации:
Отделение алгоритмов вычисления площади и периметра от классов фигур с помощью посетителей,
что позволяет добавлять новые операции без изменения существующей иерархии классов.
*/

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

using namespace std;

class Visitor;

// Абстрактный класс геометрической фигуры
class Shape {
public:
    virtual ~Shape() = default;
    virtual void accept(Visitor& visitor) = 0;
};

// Базовый класс посетителя
class Visitor {
public:
    virtual ~Visitor() = default;
    virtual void visit(class Circle& circle) = 0;
    virtual void visit(class Rectangle& rectangle) = 0;
    virtual void visit(class Triangle& triangle) = 0;
};

// Класс окружности
class Circle : public Shape {
    double radius;
public:
    Circle(double r) : radius(r) {}
    void accept(Visitor& visitor) override { visitor.visit(*this); }
    double getRadius() const { return radius; }
};

// Класс прямоугольника
class Rectangle : public Shape {
    double width, height;
public:
    Rectangle(double w, double h) : width(w), height(h) {}
    void accept(Visitor& visitor) override { visitor.visit(*this); }
    double getWidth() const { return width; }
    double getHeight() const { return height; }
};

// Класс треугольника
class Triangle : public Shape {
    double a, b, c;
public:
    Triangle(double a, double b, double c) : a(a), b(b), c(c) {}
    void accept(Visitor& visitor) override { visitor.visit(*this); }
    double getA() const { return a; }
    double getB() const { return b; }
```

```
    double getC() const { return c; }
};

// Посетитель для вычисления площади
class AreaVisitor : public Visitor {
    double area;
public:
    void visit(Circle& circle) override {
        area = M_PI * pow(circle.getRadius(), 2);
    }
    void visit(Rectangle& rect) override {
        area = rect.getWidth() * rect.getHeight();
    }
    void visit(Triangle& tri) override {
        double p = (tri.getA() + tri.getB() + tri.getC()) / 2;
        area = sqrt(p * (p - tri.getA()) * (p - tri.getB()) * (p - tri.getC()));
    }
    double getResult() const { return area; }
};

// Посетитель для вычисления периметра
class PerimeterVisitor : public Visitor {
    double perimeter;
public:
    void visit(Circle& circle) override {
        perimeter = 2 * M_PI * circle.getRadius();
    }
    void visit(Rectangle& rect) override {
        perimeter = 2 * (rect.getWidth() + rect.getHeight());
    }
    void visit(Triangle& tri) override {
        perimeter = tri.getA() + tri.getB() + tri.getC();
    }
    double getResult() const { return perimeter; }
};

int main() {
    vector<Shape*> shapes = {
        new Circle(5.0),
        new Rectangle(4.0, 6.0),
        new Triangle(3.0, 4.0, 5.0)
    };

    AreaVisitor area_calculator;
    PerimeterVisitor perimeter_calculator;

    for (auto shape : shapes) {
        shape->accept(area_calculator);
        shape->accept(perimeter_calculator);

        cout << "Area: " << area_calculator.getResult()
             << "\nPerimeter: " << perimeter_calculator.getResult()
             << "\n--------------------\n";
    }

    // Освобождение памяти
    for (auto shape : shapes) delete shape;
    return 0;
}
```