**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практической работе №5**

**по дисциплине «ООП»**

Тема: **Полиморфная логика**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Рыбин А.С. |
| Преподаватель |  | Терентьев А. О. |

Санкт-Петербург

2018

**Цель работы.**

Объедение опыта предыдущих практических работ. Изучение возможности полиморфного хранения и полиморфной обработки объектов на примере C++.

**Постановка задачи.**

Объединить предыдущие работы в приложении, использующем логику полиморфного хранения объектов. Необходимо сгенерировать контейнер из 1000 фигур, которые хранятся как shared\_ptr<Shape>, и применить к ним 2 стандартных алгоритма по вариантам. В качестве предиката использовать предикат из дополнительного задания 2-й лабораторной.

Описание вашего задания приложить в виде файла README. Корректность алгоритмов доказать с помощью юнит тестов и/или ручного тестирования. В случае юнит-тестов тестирование достаточно провести для контейнера небольшого размера (5-10) заданных фигур.

**Ход работы.**

Определим вспомогательный класс для генерации случайных фигур.

class GenShape

{

public:

    GenShape();

    stepik::vector<stepik::shared\_ptr<Shape>> get\_shape\_vector(int len);

private:

    int get\_int(int a = 1, int b = 10);

    double get\_double(double a = 1.0, double b = 10.0);

    stepik::vector<double> get\_double\_vector(int len, double a = 1.0, int b = 10.0);

    stepik::vector<int> get\_int\_vector(int len, int a = 1, int b = 10);

    stepik::vector<std::string> get\_string\_vector(int len);

    std::mt19937 gen;

};

Данный класс в качестве интерфейса будет иметь единственный метод, который возвращает случайный вектор заданной длины, состоящий из умных указателей на базовый класс. Класс содержит генератор псевдослучайных чисел mt19937, который инициализируются в конструкторе класса с помощью недетерминированного генератора std::random\_device. Так, что при каждом запуске программы фигуры будут получаться разные.

Далее определим функции алгоритмов.

std::pair<bool, stepik::shared\_ptr<Shape>> unmodif\_algorithm(const stepik::vector<stepik::shared\_ptr<Shape>>& shapes,

    int left, int right, double value);

std::pair<stepik::list<stepik::shared\_ptr<Shape>>, stepik::list<stepik::shared\_ptr<Shape>>> modif\_algorithm(stepik::vector<stepik::shared\_ptr<Shape>>& shapes,

    int left, int right, double value, GenShape& gen);

* Первый алгоритм будет возвращать пару, содержащую true, если заданный диапазон содержит хотя-бы одну фигуру, удовлетворяющую предикату и указатель на неё. В противном случае false и nullptr.
* Второй алгоритм будет возвращать два списка. Первый содержит новые случайные фигуры, которые помещены вместо тех фигур, которые удовлетворяют предикату. Второй содержит заменённые фигуры.

Программа будет работать по следующим образом:

1. Запросить кол-во фигур.
2. Сгенерировать заданное кол-во фигур.
3. Запросить левую и правую границу диапазона для немодифицирующего алгоритма и значение для предиката.
4. Применить немодифицирующий алгоритм.
5. Вывести результат.
6. Аналогично для модифицирующего алгоритма.
7. Завершить работу.

Подробное описание алгоритмов представлено в файле README.md, а исходный код представлен в приложениях к данному отчёту.

**Выводы**

Применение абстрактного базового класса при проектировании иерархии классов даёт возможность использовать полиморфную логику при работе с дочерними классами, что позволяет уменьшить объём кода, упростить его и уменьшить вероятность ошибки, а также позволяет неограниченно масштабировать систему классов, наследуя новые классы от уже созданных и, не меняя при этом код для работы с ними.

**ПРИЛОЖЕНИЕ А**

**MyAlgorithm.hpp**

#pragma once

#include <random>

#include <string>

#include "MyVector.hpp"

#include "MySharedPtr.hpp"

#include "Shape.hpp"

#include "MyList.hpp"

class GenShape

{

public:

    GenShape();

    stepik::vector<stepik::shared\_ptr<Shape>> get\_shape\_vector(int len);

private:

    int get\_int(int a = 1, int b = 10);

    double get\_double(double a = 1.0, double b = 10.0);

    stepik::vector<double> get\_double\_vector(int len, double a = 1.0, int b = 10.0);

    stepik::vector<int> get\_int\_vector(int len, int a = 1, int b = 10);

    stepik::vector<std::string> get\_string\_vector(int len);

    std::mt19937 gen;

};

std::pair<bool, stepik::shared\_ptr<Shape>> unmodif\_algorithm(const stepik::vector<stepik::shared\_ptr<Shape>>& shapes,

    int left, int right, double value);

std::pair<stepik::list<stepik::shared\_ptr<Shape>>, stepik::list<stepik::shared\_ptr<Shape>>> modif\_algorithm(stepik::vector<stepik::shared\_ptr<Shape>>& shapes,

    int left, int right, double value, GenShape& gen);

**ПРИЛОЖЕНИЕ Б**

**MyAlgorithm.cpp**

#include "MyAlgorithm.hpp"

GenShape::GenShape()

{

    std::random\_device rd;

    gen.seed(rd());

}

int GenShape::get\_int(int a, int b)

{

    return (a + gen() % (b - a + 1));

}

double GenShape::get\_double(double a, double b)

{

    std::uniform\_real\_distribution<> dist(a, b);

    return dist(gen);

}

stepik::vector<double> GenShape::get\_double\_vector(int len, double a, int b)

{

    std::uniform\_real\_distribution<> dist(a, b);

    stepik::vector<double> result(len);

    for (int i = 0; i < len; i++) {

        result[i] = dist(gen);

    }

    return result;

}

stepik::vector<int> GenShape::get\_int\_vector(int len, int a, int b)

{

    std::uniform\_int\_distribution<> dist(a, b);

    stepik::vector<int> result(len);

    for (int i = 0; i < len; i++) {

        result[i] = dist(gen);

    }

    return result;

}

stepik::vector<std::string> GenShape::get\_string\_vector(int len)

{

    std::uniform\_int\_distribution<> dist(1, 7);

    stepik::vector<std::string> result(len);

    for (int i = 0; i < len; i++) {

        int res = dist(gen);

        switch (res)

        {

        case 1:

            result[i] = "green";

            break;

        case 2:

            result[i] = "blue";

            break;

        case 3:

            result[i] = "red";

            break;

        case 4:

            result[i] = "black";

            break;

        case 5:

            result[i] = "yellow";

            break;

        case 6:

            result[i] = "orange";

            break;

        case 7:

            result[i] = "white";

            break;

        default:

            result[i] = "unknown color";

            break;

        }

    }

    return result;

}

stepik::vector<stepik::shared\_ptr<Shape>> GenShape::get\_shape\_vector(int len)

{

    stepik::vector<stepik::shared\_ptr<Shape>> result(len);

    stepik::vector<std::string> colors = get\_string\_vector(len);

    stepik::vector<double> sizes = get\_double\_vector(len);

    stepik::vector<int> types = get\_int\_vector(len, 0, 2);

    for (int i = 0; i < len; i++) {

        Point point = { get\_double(), get\_double() };

        int angle = get\_int(0, 4);

        switch (types[i])

        {

        case CIRCLE:

            result[i].reset(new Circle(colors[i], sizes[i], point));

            break;

        case SQUARE:

            result[i].reset(new Square(colors[i], sizes[i], point));

            (\*result[i]).rotate(rot\_angle(angle));

            break;

        case TRIANGLE:

            result[i].reset(new Triangle(colors[i], sizes[i], point));

            (\*result[i]).rotate(rot\_angle(angle));

            break;

        default:

            break;

        }

    }

    return result;

}

std::pair<bool, stepik::shared\_ptr<Shape>> unmodif\_algorithm(const stepik::vector<stepik::shared\_ptr<Shape>>& shapes, int left, int right, double value)

{

    for (int i = left - 1; i < right; i++) {

        if ((\*shapes[i]).area() > value) {

            return std::pair<bool, stepik::shared\_ptr<Shape>>(true, shapes[i]);

        }

        if (i == right - 1) {

            return std::pair<bool, stepik::shared\_ptr<Shape>>(false, stepik::shared\_ptr<Shape>());

        }

    }

}

std::pair<stepik::list<stepik::shared\_ptr<Shape>>,

stepik::list<stepik::shared\_ptr<Shape>>> modif\_algorithm(stepik::vector<stepik::shared\_ptr<Shape>>& shapes,

                                                                            int left, int right, double value, GenShape& gen)

{

    stepik::vector<stepik::shared\_ptr<Shape>> new\_shapes = gen.get\_shape\_vector(right - left + 1);

    stepik::list<stepik::shared\_ptr<Shape>> new\_shapes\_list;

    stepik::list<stepik::shared\_ptr<Shape>> copied;

    for (int i = left - 1; i < right; i++) {

        if ((\*shapes[i]).area() > value) {

            copied.push\_back(shapes[i]);

            shapes[i].swap(new\_shapes[i - left + 1]);

            new\_shapes\_list.push\_back(shapes[i]);

        }

    }

    return { new\_shapes\_list, copied };

}

**ПРИЛОЖЕНИЕ В**

**main.cpp**

#include "MyAlgorithm.hpp"

#include <iostream>

int main()

{

    int num = 0;

    GenShape generator;

    std::cout << "Input num of figures to generate: ";

    std::cin >> num;

    auto shapes = generator.get\_shape\_vector(num);

    std::cout << std::endl << "FIGURES:" << std::endl << std::endl;

    for (int i = 0; i < shapes.size(); i++) {

        std::cout << "#" << i + 1 << std::endl;

        std::cout << shapes[i].get() << std::endl;

        shapes[i]->operator<<(std::cout);

    }

    int leftbound, rightbound;

    double area;

    std::cout << "non-modifying algorithm: Check that at least one element of the range has an area larger than the specified value" << std::endl;

    std::cout << "left bound: ";

    std::cin >> leftbound;

    std::cout << "right bound: ";

    std::cin >> rightbound;

    std::cout << "value: ";

    std::cin >> area;

    auto result1 = unmodif\_algorithm(shapes, leftbound, rightbound, area);

    if (result1.first) {

        std::cout << std::endl << "[True]" << std::endl << "area is : " << result1.second->area() << std::endl;

        std::cout << result1.second.get() << std::endl;

        result1.second->operator<<(std::cout);

    }

    else {

        std::cout << std::endl << "[False]" << std::endl;

    }

    std::cout << "modifying algorithm: Replace elements that have an area larger than a given value by new random elements, copying the entire range." << std::endl;

    std::cout << "left bound: ";

    std::cin >> leftbound;

    std::cout << "right bound: ";

    std::cin >> rightbound;

    std::cout << "value: ";

    std::cin >> area;

    auto result2 = modif\_algorithm(shapes, leftbound, rightbound, area, generator);

    if (result2.first.size()) {

        std::cout << std::endl << "NEW FIGURES: " << std::endl << std::endl;

        for (auto it = result2.first.begin(); it != result2.first.end(); it++) {

            std::cout << it->get() << std::endl;

            it->get()->operator<<(std::cout);

        }

        std::cout << std::endl << "COPYIED OLD FIGURES: " << std::endl << std::endl;

        for (auto it = result2.second.begin(); it != result2.second.end(); it++) {

            std::cout << it->get() << std::endl;

            it->get()->operator<<(std::cout);

        }

    }

    else {

        std::cout << std::endl << "[Empty] no figures to replace" << std::endl;

    }

return 0;

}