МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3 по курсу

объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Васютинский Вадим Александрович, группа М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Цель работы

Целью лабораторной работы является:

Закрепление навыков работы с классами.

Знакомство с умными указателями.

Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий фигуры класса фигуры, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.

Требования к классу контейнера аналогичны требованиям из лабораторной работы 2.

Класс-контейнер должен соджержать объекты используя std:shared_ptr<…>.

Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

Стандартные контейнеры std.

Шаблоны (template).

Объекты «по-значению»

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер.

Распечатывать содержимое контейнера.

Удалять фигуры из контейнера.

Дневник отладки

Во время выполнения лабораторной работы была полностью изменена конструкция вектора, ввиду обнаружения более просього способа хранения объектов.

Недочёты

Недочётов не было обнаружено.

Выводы

Лабораторная работа №3 позволила мне полностью осознать концепцию умных указателей в языке C++ и отточить навыки в работе с ними. В процессе выполнения работы я на практике познакомился с умными указателями, изменил реализацию нескольких классов данных(фигуры), и для каждого из них - функции, заменив применение обычных указателей умными.

Исходный код

figure.h

```
#ifndef LAB1_FIGURE_H
#define LAB1_FIGURE_H

#include <cmath>
#include <iostream>
#include "point.h"

class Figure {
public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
    virtual void Print(std::ostream &os) = 0;
    virtual ~Figure() {};
};


#endif //LAB1_FIGURE_H
```

main.cpp

```
#include "rectangle.h"
#include "TVector.h"
#include <memory>
#include <string>

int main() {
    std::string command;
    TVector v;

    while (std::cin >> command) {
        if (command == "print")
            std::cout << v;
        else if (command == "insertlast") {
            Rectangle r;
            std::cin >> r;
            std::shared_ptr<Rectangle> d(new Rectangle(r));
            v.InsertLast(d);
        }
        else if (command == "removelast") {
            v.RemoveLast();
        }
        else if (command == "last") {
            std::cout << v.Last();
        }
```

```
        else if (command == "idx") {
            int idx;
            std::cin >> idx;
            std::cout << v[idx];
        }
        else if (command == "clear") {
            v.Clear();
        }
        else if (command == "empty") {
            if (v.Empty()) std::cout << "Yes" << std::endl;
            else std::cout << "No" << std::endl;
        }
    }
    return 0;
}
```

rectangle.cpp

```
#include "rectangle.h"

std::istream& operator>>(std::istream& is, Rectangle& r) {
    std::cout << "Enter data: " << std::endl;
    is >> r.a >> r.b >> r.c >> r.d;
    return is;
}

std::ostream& operator<<(std::ostream& os, Rectangle& r) {
    os << "Pentagon: " << r.a << r.b << r.c << r.d;
    return os;
}

Rectangle& Rectangle::operator=(const Rectangle &other) {
    this->a = other.a;
    this->b = other.b;
    this->c = other.c;
    this->d = other.d;
    return *this;
}

bool Rectangle::operator==(const Rectangle &other) {
    return a == other.a && b == other.b && c == other.c;
}

void Rectangle::Print(std::ostream &os) {
```

```
    os << "Rectangle: " << a << b << c << d << std::endl;
}

size_t Rectangle::VertexesNumber() {
    return 4;
}

double Rectangle::Area() {
    return a.dist(b) * a.dist(d);
}


Rectangle::Rectangle() {}

Rectangle::Rectangle(Point a, Point b, Point c, Point d) : a(a), b(b), c(c), d(d) {}

Rectangle::Rectangle(std::istream &is) {
    std::cout << "Enter data:" << std::endl;
    is >> a >> b >> c >> d;
    std::cout << "Rectangle created via istream" << std::endl;
}

Rectangle::Rectangle(const Rectangle &other) : Rectangle(other.a, other.b,
other.c, other.d) {
    std::cout << "Made copy of rectangle" << std::endl;
}

Rectangle::~Rectangle() {
    std::cout << "Rectangle deleted" << std::endl;
}
```

rectangle.h

```
#ifndef OOP1_RECTANGLE_H
#define OOP1_RECTANGLE_H

#include "figure.h"
class Rectangle : Figure {
public:
    friend std::istream& operator>>(std::istream& is, Rectangle& p);
    friend std::ostream& operator<<(std::ostream& os, Rectangle& p);
    bool operator==(const Rectangle& other);
    Rectangle& operator=(const Rectangle& other);
    void Print(std::ostream &os) override;
    size_t VertexesNumber() override;
    double Area() override;
    Rectangle();
```

```
    Rectangle(Point a, Point b, Point c, Point d);
    Rectangle(std::istream &is);
    Rectangle(const Rectangle &other);
    virtual ~Rectangle();
private:
    Point a, b, c, d;
};
#endif //OOP1_RECTANGLE_H
```

Point.cpp

```
#include "point.h"

#include <cmath>

bool Point::operator==(const Point &other) {
    return (this->x_ == other.x_ && this->y_ == other.y_);
}

Point::Point() : x_(0.0), y_(0.0) {}

Point::Point(double x, double y) : x_(x), y_(y) {}

Point::Point(std::istream &is) {
    is >> x_ >> y_;
}

double Point::dist(Point& other) {
    double dx = (other.x_ - x_);
    double dy = (other.y_ - y_);
    return std::sqrt(dx*dx + dy*dy);
}

std::istream& operator>>(std::istream& is, Point& p) {
    is >> p.x_ >> p.y_;
    return is;
}

std::ostream& operator<<(std::ostream& os, Point& p) {
    os << "(" << p.x_ << ", " << p.y_ << ")";
    return os;
}
```

Point.h

```
#ifndef LAB1_POINT_H
#define LAB1_POINT_H

#include <iostream>
```

```
class Point {
public:
    Point();
    Point(std::istream &is);
    Point(double x, double y);

    double dist(Point& other);

    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);
    bool operator==(const Point& other);
private:
    double x_;
    double y_;
};

#endif //LAB1_POINT_H
```

TVector.cpp

```
#include "TVector.h"
#include "rectangle.h"
#include <cassert>

TVector::TVector()
        : data_(new std::shared_ptr<Rectangle>[32]),
          length_(0), capacity_(32) {}

TVector::TVector(const TVector &vector)
        : data_(new std::shared_ptr<Rectangle>[vector.capacity_]),
          length_(vector.length_), capacity_(vector.capacity_) {
    std::copy(vector.data_, vector.data_ + vector.length_, data_);
}

TVector::~TVector() {
    delete[] data_;
}

void TVector::_Resize(const size_t new_capacity) {
    std::shared_ptr<Rectangle> *newdata = new std::shared_ptr<Rectangle>[new_capacity];
    std::copy(data_, data_ + capacity_, newdata);
    delete[] data_;
    data_ = newdata;
    capacity_ = new_capacity;
}

void TVector::InsertLast(const std::shared_ptr<Rectangle> &item) {
    if (length_ >= capacity_)
```

```
        _Resize(capacity_ << 1);
    data_[length_++] = item;
}

void TVector::EmplaceLast(const Rectangle &&item) {
    if (length_ >= capacity_)
        _Resize(capacity_ << 1);
    data_[length_++] = std::make_shared<Rectangle>(item);
}

void TVector::Remove(const size_t index) {
    std::copy(data_ + index + 1, data_ + length_, data_ + index);
    --length_;
}

void TVector::Clear() {
    delete[] data_;
    data_ = new std::shared_ptr<Rectangle>[32];
    length_ = 0;
    capacity_ = 32;
}

std::ostream &operator<<(std::ostream &os, const TVector &vector) {
    for (size_t i = 0; i < vector.length_; ++i)
        os << (*vector.data_[i]);
    os << std::endl;
    return os;
}
```

TVector.h

```
#ifndef OOP2_TVECTOR_H
#define OOP2_TVECTOR_H


#include <iostream>
#include <memory>
#include <cstdlib>


#include "rectangle.h"


class TVector {
public:
    TVector();


    TVector(const TVector &);
```

```
    virtual ~TVector();

    size_t Length() const {
        return length_;
    }

    bool Empty() const {
        return !length_;
    }

    const std::shared_ptr<Rectangle> &operator[](const size_t index) const {
        return data_[index];
    }

    std::shared_ptr<Rectangle> &Last() const {
        return data_[length_ - 1];
    }

    void InsertLast(const std::shared_ptr<Rectangle> &);

    void EmplaceLast(const Rectangle &&);

    void Remove(const size_t index);

    Rectangle RemoveLast() {
        return *data_[--length_];
    }

    void Clear();

    friend std::ostream &operator<<(
            std::ostream &, const TVector &);

private:
    void _Resize(const size_t new_capacity);

    std::shared_ptr<Rectangle> *data_;
    size_t length_, capacity_;
};
#endif //OOP2_TVECTOR_H
```