МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФГАОУ ВО «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ

ТЕХНОЛОГИЙ И ТЕЛЕКОММУНИКАЦИЙ

КАФЕДРА ПРИКЛАДНОЙ ИНФОРМАТИКИ

**Лабораторная работа №2**

по дисциплине

«Объектно-ориентированное программирование»

**Выполнил:**

Ермолаев Кирилл Александрович

Студент 2 курса группы \_ПИН-б-о-22-1

Направления подготовки

09.03.03 Прикладная информатика

очной формы обучения

Ставрополь, 2023 г.

**Тема:** Основы объектно-ориентированного программирования на ЯП C++.

**Цель работы:** изучить базовые понятия (классы, подклассы и методы).

Реализовать фундаментальные принципы объектно-ориентированного программирования.

**Ход работы:**

**Вариант 7**

Составить описание класса прямоугольников со сторонами, параллельными осям координат. Предусмотреть возможность перемещения прямоугольников на плоскости, изменение размеров, построение наименьшего прямоугольника, содержащего два заданных прямоугольника, и прямоугольника, являющегося общей частью (пересечением) двух прямоугольников.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

// main.cpp

#include "Rectangle.h"

#include <iostream>

int main() {

// Create two rectangles

Rectangle rect1(1, 1, 4, 4);

Rectangle rect2(3, 2, 6, 5);

int choice;

do {

// Отображаем меню

std::cout << "\nMenu:\n";

std::cout << "1. Move Rectangle\n";

std::cout << "2. Resize Rectangle\n";

std::cout << "3. Minimum Bounding Rectangle\n";

std::cout << "4. Intersection Rectangle\n";

std::cout << "5. Display Rectangles\n";

std::cout << "0. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice) {

case 1: {

double dx, dy;

std::cout << "Enter displacement along x and y: ";

std::cin >> dx >> dy;

rect1.move(dx, dy);

rect2.move(dx, dy);

break;

}

case 2: {

double dw, dh;

std::cout << "Enter change in width and height: ";

std::cin >> dw >> dh;

rect1.resize(dw, dh);

rect2.resize(dw, dh);

break;

}

case 3: {

Rectangle boundingRect = rect1.minimumBoundingRectangle(rect2);

std::cout << "Minimum Bounding Rectangle:\n";

boundingRect.display();

break;

}

case 4: {

Rectangle intersectionRect = rect1.intersection(rect2);

std::cout << "Intersection Rectangle:\n";

intersectionRect.display();

break;

}

case 5:

std::cout << "Rectangles:\n";

std::cout << "Rectangle 1: ";

rect1.display();

std::cout << "Rectangle 2: ";

rect2.display();

break;

case 0:

std::cout << "Exiting program.\n";

break;

default:

std::cout << "Invalid choice. Try again.\n";

}

} while (choice != 0);

return 0;

}

// rectangle.cpp

#include "Rectangle.h"

#include <iostream>

#include <algorithm>

Rectangle::Rectangle(double x1, double y1, double x2, double y2)

: x1(x1), y1(y1), x2(x2), y2(y2) {}

void Rectangle::move(double dx, double dy) {

x1 += dx;

y1 += dy;

x2 += dx;

y2 += dy;

}

void Rectangle::resize(double dw, double dh) {

x2 += dw;

y2 += dh;

}

Rectangle Rectangle::minimumBoundingRectangle(const Rectangle& other) const {

double minX = std::min(x1, other.x1);

double minY = std::min(y1, other.y1);

double maxX = std::max(x2, other.x2);

double maxY = std::max(y2, other.y2);

return Rectangle(minX, minY, maxX, maxY);

}

Rectangle Rectangle::intersection(const Rectangle& other) const {

double intX1 = std::max(x1, other.x1);

double intY1 = std::max(y1, other.y1);

double intX2 = std::min(x2, other.x2);

double intY2 = std::min(y2, other.y2);

if (intX1 < intX2 && intY1 < intY2) {

return Rectangle(intX1, intY1, intX2, intY2);

}

else {

// Нет пересечения

return Rectangle(0, 0, 0, 0);

}

}

void Rectangle::display() const {

std::cout << "Rectangle: (" << x1 << ", " << y1 << ") - (" << x2 << ", " << y2 << ")\n";

}

// rectangle.cpp

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

class Rectangle {

private:

double x1, y1;

double x2, y2;

public:

Rectangle(double x1, double y1, double x2, double y2);

void move(double dx, double dy);

void resize(double dw, double dh);

Rectangle minimumBoundingRectangle(const Rectangle& other) const;

Rectangle intersection(const Rectangle& other) const;

void display() const;

double getX1() const { return x1; }

double getY1() const { return y1; }

double getX2() const { return x2; }

double getY2() const { return y2; }

};

#endif

// UnitTest2.cpp

#include "pch.h"

#include "CppUnitTest.h"

#include "..\ConsoleApplication1\rectangle.h"

#include "..\ConsoleApplication1\rectangle.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest2

{

TEST\_CLASS(RectangleTests)

{

public:

TEST\_METHOD(TestMove)

{

Rectangle rect(1, 1, 4, 4);

rect.move(2, 3);

Assert::AreEqual(3.0, rect.getX1());

Assert::AreEqual(4.0, rect.getY1());

Assert::AreEqual(6.0, rect.getX2());

Assert::AreEqual(7.0, rect.getY2());

}

TEST\_METHOD(TestResize)

{

Rectangle rect(1, 1, 4, 4);

rect.resize(2, 3);

Assert::AreEqual(1.0, rect.getX1());

Assert::AreEqual(1.0, rect.getY1());

Assert::AreEqual(6.0, rect.getX2());

Assert::AreEqual(7.0, rect.getY2());

}

TEST\_METHOD(TestMinimumBoundingRectangle)

{

Rectangle rect1(1, 1, 4, 4);

Rectangle rect2(3, 2, 6, 5);

Rectangle boundingRect = rect1.minimumBoundingRectangle(rect2);

Assert::AreEqual(1.0, boundingRect.getX1());

Assert::AreEqual(1.0, boundingRect.getY1());

Assert::AreEqual(6.0, boundingRect.getX2());

Assert::AreEqual(5.0, boundingRect.getY2());

}

TEST\_METHOD(TestIntersection)

{

Rectangle rect1(1, 1, 4, 4);

Rectangle rect2(3, 2, 6, 5);

Rectangle intersectionRect = rect1.intersection(rect2);

Assert::AreEqual(3.0, intersectionRect.getX1());

Assert::AreEqual(2.0, intersectionRect.getY1());

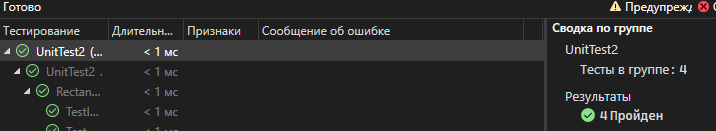
Assert::AreEqual(4.0, intersectionRect.getX2());

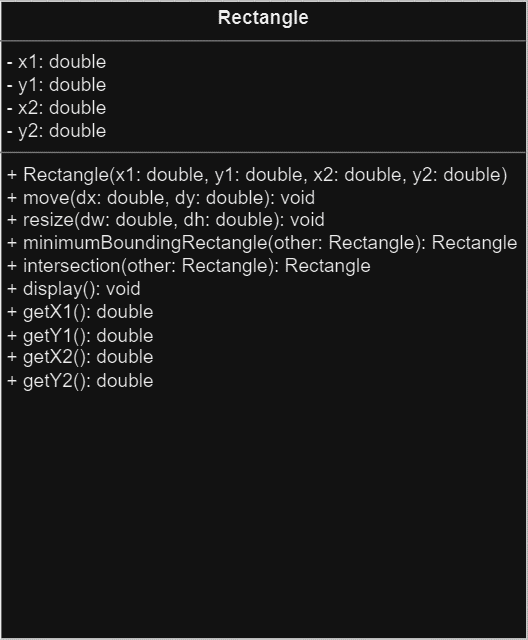
Assert::AreEqual(4.0, intersectionRect.getY2());

}

};

}





<https://github.com/MoeTomatoki/OOP>

**Вывод:** изучил базовые понятия (классы, подклассы и методы). Реализовал фундаментальные принципы объектно-ориентированного программирования.