**Отчет о выполнении лабораторной работы 3**

**Олейников Артём, группа НБИ-БД-23-02**

**Задание:**

Написать компьютерную программу, содержащую описание классов для иерархии геометрических объектов (точка, линия, квадрат, ромб, прямоугольник, параллелограмм) с реализацией набора методов (изобразить, убрать, передвинуть, повернуть) .

Проверить возможность реализации различных вариантов регламентации доступа к компонентам классов.

Реализовать отождествление базовых объектов "Квадрат" для объектов "Параллелограмм" с использованием виртуального наследования.

Преобразовать класс "Точка" в абстрактный класс.

Проверить возможность позднего связывания для методов геометрических объектов.

**Код:**

#include <iostream>

#include <cmath>

#include <vector>

#include <memory> // Для std::unique\_ptr и std::make\_unique

using namespace std;

// Абстрактный класс "Точка"

class Point {

protected:

double x, y;

public:

Point(double x = 0, double y = 0) : x(x), y(y) {}

virtual void draw() const = 0;

virtual void erase() const = 0;

virtual void move(double dx, double dy) = 0;

virtual void rotate(double angle) = 0;

virtual ~Point() = default;

};

// Класс "Линия"

class Line : public Point {

protected:

double x2, y2;

public:

Line(double x1, double y1, double x2, double y2)

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

void draw() const override {

cout << "Рисуем Линию от (" << x << ", " << y << ") до (" << x2 << ", " << y2 << ")\n";

}

void erase() const override {

cout << "Стираем Линию\n";

}

void move(double dx, double dy) override {

x += dx;

y += dy;

x2 += dx;

y2 += dy;

}

void rotate(double angle) override {

double rad = angle \* M\_PI / 180.0;

double newX2 = x + (x2 - x) \* cos(rad) - (y2 - y) \* sin(rad);

double newY2 = y + (x2 - x) \* sin(rad) + (y2 - y) \* cos(rad);

x2 = newX2;

y2 = newY2;

}

};

// Класс "Параллелограмм" с виртуальным наследованием

class Parallelogram : public virtual Point {

protected:

double side1, side2;

double angle; // В градусах

public:

Parallelogram(double x, double y, double side1, double side2, double angle)

: Point(x, y), side1(side1), side2(side2), angle(angle) {}

void draw() const override {

cout << "Рисуем Параллелограмм в точке (" << x << ", " << y << ") со сторонами " << side1 << " и " << side2 << " и углом " << angle << "\n";

}

void erase() const override {

cout << "Стираем Параллелограмм\n";

}

void move(double dx, double dy) override {

x += dx;

y += dy;

}

void rotate(double angle) override {

this->angle += angle;

}

};

// Класс "Квадрат", наследующий от "Параллелограмма"

class Square : public Parallelogram {

public:

Square(double x, double y, double side)

: Parallelogram(x, y, side, side, 90) {}

void draw() const override {

cout << "Рисуем Квадрат в точке (" << x << ", " << y << ") со стороной " << side1 << "\n";

}

void erase() const override {

cout << "Стираем Квадрат\n";

}

};

// Класс "Ромб"

class Rhombus : public virtual Point {

protected:

double diagonal1, diagonal2;

public:

Rhombus(double x, double y, double diagonal1, double diagonal2)

: Point(x, y), diagonal1(diagonal1), diagonal2(diagonal2) {}

void draw() const override {

cout << "Рисуем Ромб в точке (" << x << ", " << y << ") с диагоналями " << diagonal1 << ", " << diagonal2 << "\n";

}

void erase() const override {

cout << "Стираем Ромб\n";

}

void move(double dx, double dy) override {

x += dx;

y += dy;

}

void rotate(double angle) override {

// Поворот ромба вокруг его центра не меняет его вид, так что метод может остаться пустым

}

};

// Класс "Прямоугольник"

class Rectangle : public Parallelogram {

public:

Rectangle(double x, double y, double width, double height)

: Parallelogram(x, y, width, height, 90) {}

void draw() const override {

cout << "Рисуем Прямоугольник в точке (" << x << ", " << y << ") с шириной " << side1 << " и высотой " << side2 << "\n";

}

void erase() const override {

cout << "Стираем Прямоугольник\n";

}

};

void displayMenu() {

cout << "Выберите фигуру:\n";

cout << "1. Линия\n";

cout << "2. Квадрат\n";

cout << "3. Прямоугольник\n";

cout << "4. Параллелограмм\n";

cout << "5. Ромб\n";

cout << "0. Выход\n";

}

void displayActionMenu() {

cout << "Выберите действие:\n";

cout << "1. Нарисовать\n";

cout << "2. Стереть\n";

cout << "3. Передвинуть\n";

cout << "4. Повернуть\n";

cout << "0. Назад\n";

}

int main() {

vector<unique\_ptr<Point>> shapes;

while (true) {

displayMenu();

int shapeChoice;

cin >> shapeChoice;

if (shapeChoice == 0) break;

unique\_ptr<Point> shape;

switch (shapeChoice) {

case 1: {

double x1, y1, x2, y2;

cout << "Введите координаты начала и конца линии (x1 y1 x2 y2): ";

cin >> x1 >> y1 >> x2 >> y2;

shape = unique\_ptr<Point>(new Line(x1, y1, x2, y2));

break;

}

case 2: {

double x, y, side;

cout << "Введите координаты левого нижнего угла и сторону квадрата (x y side): ";

cin >> x >> y >> side;

shape = unique\_ptr<Point>(new Square(x, y, side));

break;

}

case 3: {

double x, y, width, height;

cout << "Введите координаты левого нижнего угла, ширину и высоту прямоугольника (x y width height): ";

cin >> x >> y >> width >> height;

shape = unique\_ptr<Point>(new Rectangle(x, y, width, height));

break;

}

case 4: {

double x, y, side1, side2, angle;

cout << "Введите координаты левого нижнего угла, стороны и угол параллелограмма (x y side1 side2 angle): ";

cin >> x >> y >> side1 >> side2 >> angle;

shape = unique\_ptr<Point>(new Parallelogram(x, y, side1, side2, angle));

break;

}

case 5: {

double x, y, diagonal1, diagonal2;

cout << "Введите координаты центра и диагонали ромба (x y diagonal1 diagonal2): ";

cin >> x >> y >> diagonal1 >> diagonal2;

shape = unique\_ptr<Point>(new Rhombus(x, y, diagonal1, diagonal2));

break;

}

default:

cout << "Неправильный выбор. Попробуйте еще раз.\n";

continue;

}

while (true) {

displayActionMenu();

int actionChoice;

cin >> actionChoice;

if (actionChoice == 0) break;

switch (actionChoice) {

case 1:

shape->draw();

break;

case 2:

shape->erase();

break;

case 3: {

double dx, dy;

cout << "Введите смещение по x и y (dx dy): ";

cin >> dx >> dy;

shape->move(dx, dy);

break;

}

case 4: {

double angle;

cout << "Введите угол поворота: ";

cin >> angle;

shape->rotate(angle);

break;

}

default:

cout << "Неправильный выбор. Попробуйте еще раз.\n";

continue;

}

}

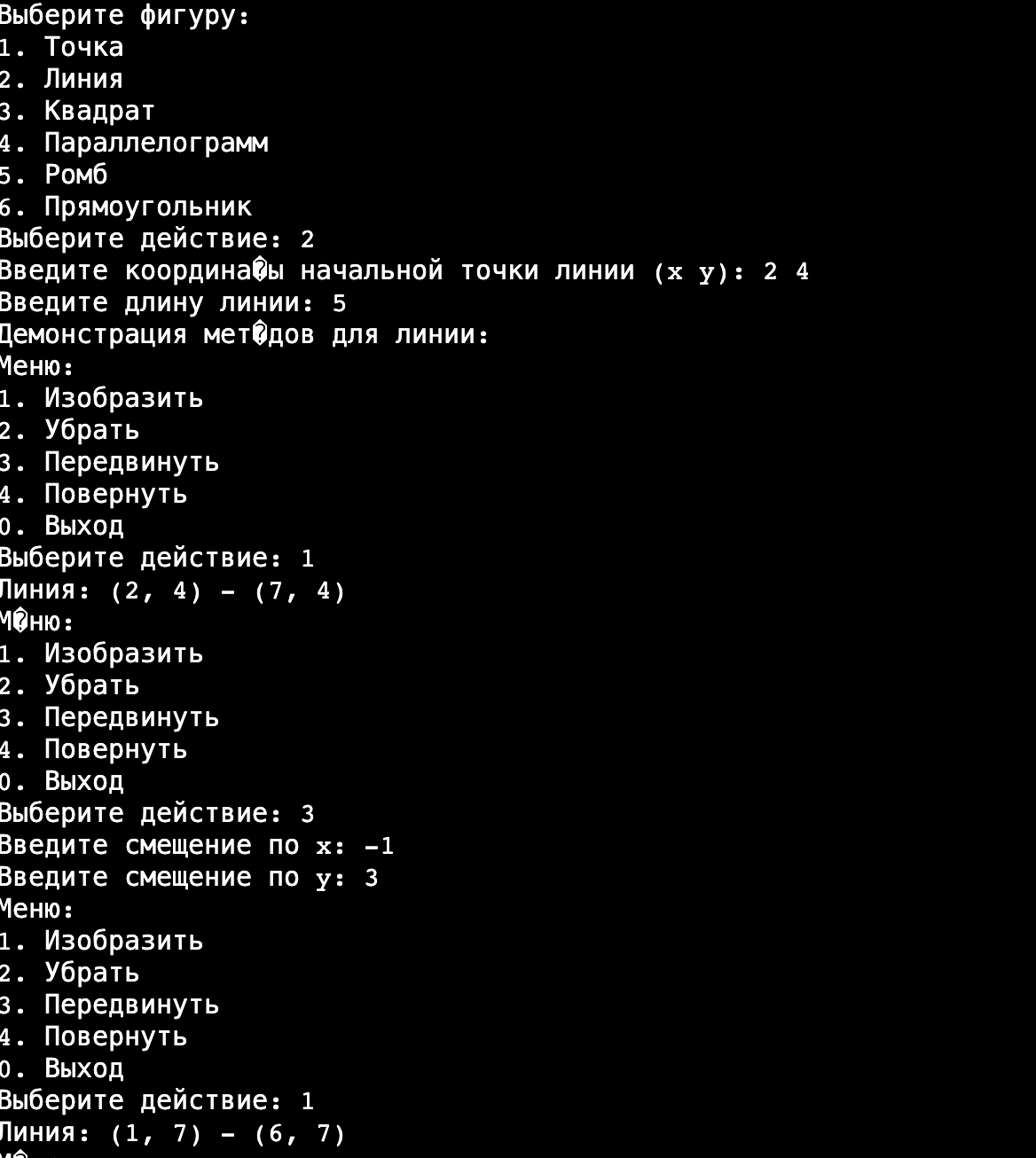
}

cout << "Программа завершена.\n";

return 0;

}

**Пример вывода:**

****

**Структура программы:**

1. **Абстрактный класс Point:**
   * Содержит защищенные члены x и y для хранения координат.
   * Виртуальные методы draw(), erase(), move(), и rotate() определены как чисто виртуальные функции.
   * Определен виртуальный деструктор.
2. **Класс Line:**
   * Наследуется от Point.
   * Дополнительно хранит координаты конца линии (x2, y2).
   * Реализует все виртуальные методы базового класса.
3. **Класс Parallelogram с виртуальным наследованием:**
   * Наследуется виртуально от Point.
   * Содержит параметры сторон (side1, side2) и угла (angle).
   * Реализует методы draw(), erase(), move(), и rotate().
4. **Класс Square:**
   * Наследуется от Parallelogram.
   * Конструктор принимает координаты и сторону квадрата, передавая их в конструктор Parallelogram с углом 90 градусов.
   * Переопределяет методы draw() и erase().
5. **Класс Rhombus:**
   * Наследуется виртуально от Point.
   * Содержит параметры диагоналей (diagonal1, diagonal2).
   * Реализует методы draw(), erase(), move(), и rotate().
6. **Класс Rectangle:**
   * Наследуется от Parallelogram.
   * Конструктор принимает координаты, ширину и высоту прямоугольника, передавая их в конструктор Parallelogram с углом 90 градусов.
   * Переопределяет методы draw() и erase().
7. **Функции отображения меню:**
   * displayMenu(): Отображает меню выбора фигуры.
   * displayActionMenu(): Отображает меню выбора действия.
8. **Функция main():**
   * Основной цикл программы, в котором пользователь выбирает фигуру и действие.
   * В зависимости от выбора пользователя создается соответствующий объект с помощью std::unique\_ptr.
   * Внутренний цикл для выполнения действий над выбранной фигурой.
   * Поддержка ввода пользователя и вызов соответствующих методов для каждого действия.

**Алгоритм работы программы:**

1. **Инициализация:**
   * Программа начинается с инициализации вектора shapes для хранения указателей на геометрические объекты.
2. **Выбор фигуры:**
   * Пользователь выбирает тип фигуры из основного меню.
   * В зависимости от выбора создается соответствующий объект (линия, квадрат, прямоугольник, параллелограмм, ромб).
3. **Выбор действия:**
   * Пользователь выбирает действие из меню действий.
   * В зависимости от выбора вызывается соответствующий метод объекта:
     + draw() - отображение фигуры.
     + erase() - удаление фигуры.
     + move() - перемещение фигуры на заданное смещение.
     + rotate() - поворот фигуры на заданный угол.
4. **Повторение:**
   * Пользователь может продолжать выбирать действия для текущей фигуры или вернуться к выбору другой фигуры.
5. **Завершение работы:**
   * Программа завершается, когда пользователь выбирает опцию выхода из основного меню.

**Проверка возможности позднего связывания:**

Позднее связывание (late binding) проверяется в программе с использованием виртуальных методов и полиморфизма. Виртуальные методы определены в базовом абстрактном классе Point и переопределены в производных классах (Line, Square, Rectangle, Parallelogram, Rhombus). Вызовы методов через указатели на базовый класс позволяют динамически связывать вызовы функций с их реализациями в производных классах во время выполнения программы.

Таким образом, структура программы и алгоритм обеспечивают требуемую функциональность для работы с геометрическими объектами, используя принципы объектно-ориентированного программирования, такие как наследование, полиморфизм и инкапсуляция.