ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

Кафедра ВМиК

# ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

по предмету **«Объектно-Ориентированное Программирование»**

Выполнил: студент группы МО-204Б

Исламов Ильнур Фандасович.

Проверил:
доцент каф. ВМиК

Котельников В.А.

**Уфа 2025г.**

## Цель лабораторной работы

В рамках лабораторной работы необходимо разобраться:

- каким образом определяются и реализуются классы в C++;
- каким образом создаются, инициализируются и уничтожаются объекты;
- каким образом работает наследование и композиция классов;
- в чем разница между статическим и динамическим созданием объектов, а также между хранением объекта и указателя на объект.

**Задание**

**Часть 1. Создание и базовое использование класса**

- Создать класс Point, представляющий точку на плоскости.
  - **Атрибуты:** x, y (координаты типа double).
  - **Методы:**
    - void print() - выводит координаты в консоль.
    - double distanceToOrigin() - возвращает расстояние от точки до начала координат.
  - **Конструкторы:**
    - Конструктор по умолчанию (без параметров).
    - Конструктор с двумя параметрами (x, y).
    - Конструктор копирования (с параметром const Point&).
  - **Деструктор:** Выводит сообщение в консоль о разрушении объекта.
  - Все конструкторы и деструктор должны содержать отладочный вывод на консоль.
- **Реализация и тестирование:**
  - Реализовать все методы класса.
  - В функции main() продемонстрировать:
    - Создание статического объекта с помощью каждого конструктора.
    - Создание динамического объекта с помощью каждого конструктора.
    - Обращение к атрибутам и вызов методов для всех созданных объектов.

        - Явное уничтожение динамических объектов с помощью delete.

**Часть 2. Изучение наследования и переопределения методов**

- Создать класс-наследник ColoredPoint, который добавляет свойство цвета.
    - **Атрибут:** color (типа std::string).
    - **Конструкторы:**
        - Конструктор по умолчанию.
        - Конструктор с параметрами (x, y, color).
        - Использовать списки инициализации для вызова конструкторов базового класса.
    - **Методы:**
        - Переопределить метод print(), чтобы он выводил также и цвет.
        - В переопределенном методе print() продемонстрировать вызов унаследованного метода Point::print().

3. **Реализация и тестирование:**
    - Продемонстрировать создание объектов ColoredPoint статически и динамически.
    - Показать порядок вызова конструкторов и деструкторов базового и производного классов.
    - Показать работу переопределенного метода.

**Часть 3. Изучение композиции объектов**

- Создать класс Line, представляющий отрезок, состоящий из двух точек.
    - **Вариант А (хранение объектов):**
        - **Атрибуты:** start (тип Point), end (тип Point).
        - **Конструктор:** с параметрами (Point start, Point end). Инициализировать атрибуты в списке инициализации.
    - **Вариант В (хранение указателей):**
        - **Атрибуты:** start (тип Point*), end (тип Point*).
        - **Конструктор:** с параметрами (Point* start, Point* end). Создать новые объекты Point в куче.
        - **Деструктор:** Удалить объекты, на которые ссылаются указатели start и end.

- **Реализация и тестирование:**
    - Продемонстрировать создание объекта Line обоими способами.
    - Показать разницу в поведении:
        - При изменении исходных точек после создания отрезка (для варианта с указателями изменения отразятся на отрезке).
        - В порядке вызова конструкторов и деструкторов композируемых объектов Point.

**Часть 4. Работа с указателями и присваивание**

- **Присваивание объектов:**
    - Создать два объекта Point p1(1,1) и Point p2(2,2).
    - Выполнить p1 = p2. Показать, что объекты стали независимыми копиями (изменить p1.x и убедиться, что p2.x не изменился).
- **Присваивание указателей:**
    - Создать два динамических объекта Point* ptr1 = new Point(1,1) и Point* ptr2 = new Point(2,2).
    - Выполнить ptr1 = ptr2. Показать проблему утечки памяти и то, что об указателях теперь ссылаются на один и тот же объект.
- **Полиморфизм и типы переменных:**
    - Создать переменную базового типа, содержащую объект потомка: Point* polyPoint = new ColoredPoint(3, 3, "red").
    - Продемонстрировать:
        - Вызов метода print() через этот указатель. Объяснить результат (без virtual будет вызван метод Point).
        - Что произойдет при попытке обратиться к атрибуту color через polyPoint (это невозможно).
        - Правильное уничтожение такого объекта (деструктор базового класса должен быть virtual для корректного вызова деструктора потомка).

**Ход выполнения лабораторной работы**

```
=== 1. Демонстрация создания объектов ===

--- Статическое создание ---
Shape default constructor: Unknown Shape
Shape parameterized constructor: Generic Shape

--- Динамическое создание ---
Shape parameterized constructor: Dynamic Shape
```

```

--- Конструктор копирования ---
Shape copy constructor: Generic Shape (copy)
Drawing black Unknown Shape
Drawing green Generic Shape
Drawing yellow Dynamic Shape
Drawing green Generic Shape (copy)
Shape destructor: Dynamic Shape
Shape destructor: Generic Shape (copy)
Shape destructor: Generic Shape
Shape destructor: Unknown Shape
```

Рис .1, Рис .2 Демонстрация создания объектов

```
--- Создание Circle ---
Shape parameterized constructor: Circle
Point constructor: (0, 0)
Circle default constructor
Shape parameterized constructor: Circle
Point constructor: (0, 0)
Circle parameterized constructor, radius: 2.5
```

```
--- Создание Rectangle ---
Shape parameterized constructor: Rectangle
Rectangle default constructor
Shape parameterized constructor: Rectangle
Rectangle parameterized constructor, size: 3x4
Drawing circle: red circle with radius 1 (area: 3.14159)
Drawing circle: purple circle with radius 2.5 (area: 19.635)
Drawing rectangle: blue rectangle 1x1 (area: 1)
Drawing rectangle: orange rectangle 3x4 (area: 12)
```

```
--- Копирование объектов-наследников ---
Shape copy constructor: Circle (copy)
Point constructor: (0, 0)
Circle copy constructor, radius: 2.5
Drawing circle: purple circle with radius 2.5 (area: 19.635)
Circle destructor: Circle (copy) with radius 2.5
Point destructor: (0, 0)
Shape destructor: Circle (copy)
Rectangle destructor: Rectangle size 3x4
Shape destructor: Rectangle
Rectangle destructor: Rectangle size 1x1
Shape destructor: Rectangle
Circle destructor: Circle with radius 2.5
Point destructor: (0, 0)
Shape destructor: Circle
Circle destructor: Circle with radius 1
Point destructor: (0, 0)
Shape destructor: Circle
```

Рис .3, Рис .4, Рис .5 Демонстрация наследования

```
--- Указатели базового класса на объекты наследников ---
Shape parameterized constructor: Circle
Point constructor: (0, 0)
Circle parameterized constructor, radius: 3
Shape parameterized constructor: Rectangle
Rectangle parameterized constructor, size: 2x5
Drawing circle: pink circle with radius 3 (area: 28.2743)
Drawing rectangle: brown rectangle 2x5 (area: 10)
Shape1 color: pink
Shape2 color: brown
Circle destructor: Circle with radius 3
Point destructor: (0, 0)
Shape destructor: Circle
Rectangle destructor: Rectangle size 2x5
Shape destructor: Rectangle
```

Рис .6 Демонстрация работы с указателями

```
=== Демонстрация РАЗНИЦЫ композиции ===

--- 1. Композиция с ПРЯМЫМ объектом ---
Shape parameterized constructor: CircleWithPoint
Point constructor: (10, 20)
CircleWithPoint constructor complete
CircleWithPoint: red at (10, 20) radius 2
CircleWithPoint destructor
Point destructor: (10, 20)
Shape destructor: CircleWithPoint
```

```
--- 2. Композиция с УКАЗАТЕЛЕМ ---
Shape parameterized constructor: Circle
Point constructor: (0, 0)
Circle parameterized constructor, radius: 3
ShapeWithPointer constructor: PointerComp
Pointer container 'PointerComp' contains: Drawing circle: blue circle with radius 3 (area: 28.2743)
ShapeWithPointer destructor: PointerComp
Circle destructor: Circle with radius 3
Point destructor: (0, 0)
Shape destructor: Circle
```

Рис .7, Рис .8 Демонстрация РАЗНИЦЫ композиции

```
--- Массив указателей на базовый класс ---
Shape parameterized constructor: Circle
Point constructor: (0, 0)
Circle parameterized constructor, radius: 1
Shape parameterized constructor: Rectangle
Rectangle parameterized constructor, size: 2x3
Shape parameterized constructor: Circle
Point constructor: (0, 0)
Circle parameterized constructor, radius: 1.5
Drawing circle: red circle with radius 1 (area: 3.14159)
Area: 3.14159
Drawing rectangle: blue rectangle 2x3 (area: 6)
Area: 6
Drawing circle: green circle with radius 1.5 (area: 7.06858)
Area: 7.06858
Circle destructor: Circle with radius 1
Point destructor: (0, 0)
Shape destructor: Circle
Rectangle destructor: Rectangle size 2x3
Shape destructor: Rectangle
Circle destructor: Circle with radius 1.5
Point destructor: (0, 0)
Shape destructor: Circle
```

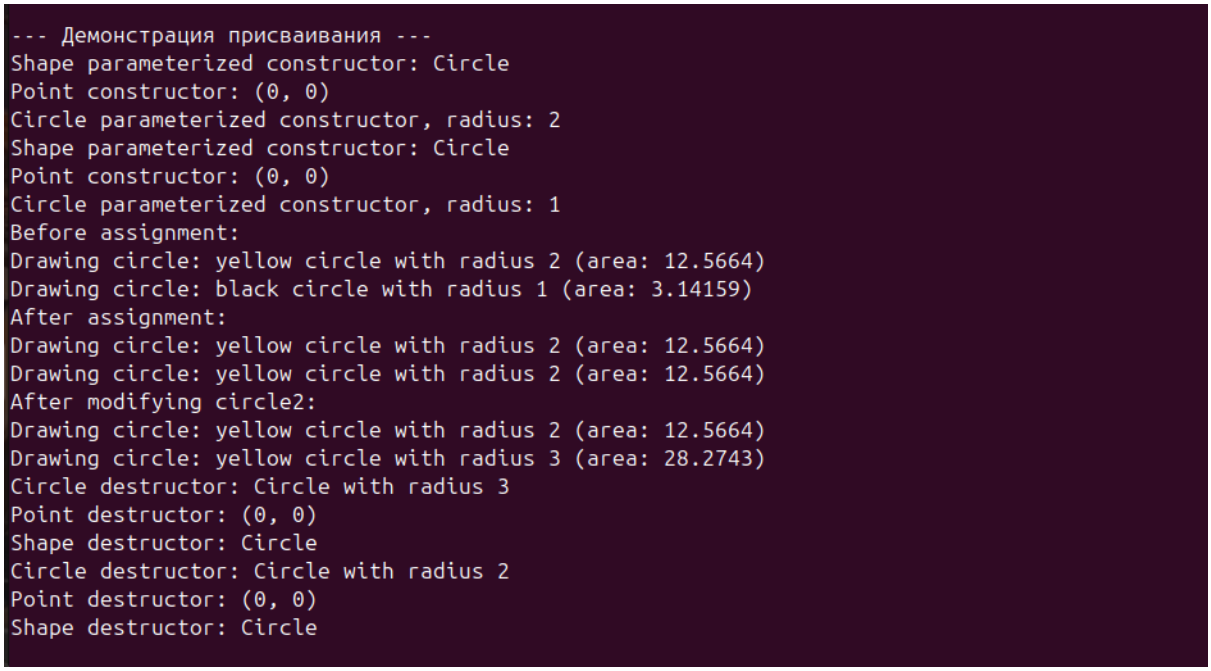


Рис .9, Рис .10 Дополнительные демонстрации

## Выводы по лабораторной работе

В ходе лабораторной работы были успешно освоены ключевые аспекты ООП в C++: создание иерархий классов, работа с конструкторами и деструкторами, применение наследования и полиморфизма, управление динамической памятью через указатели. Практически изучены принципы композиции объектов и различия между способами хранения данных. Разработанное приложение наглядно демонстрирует жизненный цикл объектов и взаимодействие между классами, что формирует фундамент для дальнейшего изучения объектно-ориентированного программирования.

**Приложение №1**

```
#include <iostream>
#include <string>
#include "shape.h"
#include "circle.h"
#include "rectangle.h"

// Функция для демонстрации создания объектов
void demonstrateObjectCreation() {
    std::cout << "\n=== 1. Демонстрация создания объектов ===" <<
std::endl;

    std::cout << "\n--- Статическое создание ---" << std::endl;
```

```
    Shape shape1;  // конструктор без параметров
    Shape shape2("green", "Generic Shape");  // конструктор с параметрами

    std::cout << "\n--- Динамическое создание ---" << std::endl;
    Shape* shape3 = new Shape("yellow", "Dynamic Shape");

    std::cout << "\n--- Конструктор копирования ---" << std::endl;
    Shape shape4 = shape2;  // копирование

    // Вызов методов
    shape1.draw();
    shape2.draw();
    shape3->draw();
    shape4.draw();

    // Очистка динамической памяти
    delete shape3;
}

// Функция для демонстрации наследования
void demonstrateInheritance() {
    std::cout << "\n=== 2. Демонстрация наследования ===" << std::endl;

    std::cout << "\n--- Создание Circle ---" << std::endl;
    Circle circle1;  // конструктор без параметров
    Circle circle2("purple", 2.5);  // конструктор с параметрами

    std::cout << "\n--- Создание Rectangle ---" << std::endl;
    Rectangle rect1;
    Rectangle rect2("orange", 3.0, 4.0);

    // Вызов методов
    circle1.draw();
    circle2.draw();
    rect1.draw();
    rect2.draw();

    std::cout << "\n--- Копирование объектов-наследников ---" << std::endl;
    Circle circle3 = circle2;  // конструктор копирования
    circle3.draw();
}

// Функция для демонстрации работы с указателями
void demonstratePointers() {
```

```
    std::cout << "\n=== 3. Демонстрация работы с указателями ===" <<
std::endl;

    std::cout << "\n--- Указатели базового класса на объекты наследников ---
" << std::endl;
    Shape* shapePtr1 = new Circle("pink", 3.0);
    Shape* shapePtr2 = new Rectangle("brown", 2.0, 5.0);

    // Вызов виртуальных методов
    shapePtr1->draw();  // вызовет Circle::draw()
    shapePtr2->draw();  // вызовет Rectangle::draw()

    // Вызов невиртуальных методов
    std::cout << "Shape1 color: " << shapePtr1->getColor() << std::endl;
    std::cout << "Shape2 color: " << shapePtr2->getColor() << std::endl;

    // Очистка
    delete shapePtr1;
    delete shapePtr2;
}

// Circle с прямой композицией Point
class CircleWithPoint : public Shape {
private:
    double radius;
    Point center;  // Прямой объект (не указатель!)

public:
    // Конструктор с инициализацией в списке
    CircleWithPoint(const std::string& color, double radius, double x, double y)
        : Shape(color, "CircleWithPoint"), radius(radius), center(x, y) {
        std::cout << "CircleWithPoint constructor complete" << std::endl;
    }

    ~CircleWithPoint() {
        std::cout << "CircleWithPoint destructor" << std::endl;
    }

    void draw() const override {
        std::cout << "CircleWithPoint: " << color << " at ("
                  << center.getX() << ", " << center.getY()
                  << ") radius " << radius << std::endl;
    }
};
```

```
// Класс с композицией через указатель
class ShapeWithPointer {
private:
    Shape* shapePtr;
    std::string containerName;

public:
    ShapeWithPointer(const std::string& name, Shape* shape)
        : containerName(name), shapePtr(shape) {
        std::cout << "ShapeWithPointer constructor: " << name << std::endl;
    }

    ~ShapeWithPointer() {
        std::cout << "ShapeWithPointer destructor: " << containerName <<
std::endl;
        delete shapePtr;
    }

    void display() const {
        std::cout << "Pointer container '" << containerName << "' contains: ";
        shapePtr->draw();
    }
};

void demonstrateCompositionDifference() {
    std::cout << "\n=== Демонстрация РАЗНИЦЫ композиции ===" <<
std::endl;

    std::cout << "\n--- 1. Композиция с ПРЯМЫМ объектом ---" << std::endl;
    {
        CircleWithPoint circle("red", 2.0, 10.0, 20.0);
        circle.draw();
    } // Point уничтожится автоматически!

    std::cout << "\n--- 2. Композиция с УКАЗАТЕЛЕМ ---" << std::endl;
    {
        ShapeWithPointer container("PointerComp", new Circle("blue", 3.0));
        container.display();
    } // Нужно самим delete в деструкторе!
}

// Дополнительные демонстрации
void demonstrateAdvancedConcepts() {
    std::cout << "\n=== 5. Дополнительные демонстрации ===" << std::endl;
```

```
    std::cout << "\n--- Массив указателей на базовый класс ---" << std::endl;
    Shape* shapes[3];
    shapes[0] = new Circle("red", 1.0);
    shapes[1] = new Rectangle("blue", 2.0, 3.0);
    shapes[2] = new Circle("green", 1.5);

    for (int i = 0; i < 3; ++i) {
        shapes[i]->draw();
        std::cout << "Area: " << shapes[i]->area() << std::endl;
    }

    // Очистка
    for (int i = 0; i < 3; ++i) {
        delete shapes[i];
    }

    std::cout << "\n--- Демонстрация присваивания ---" << std::endl;
    Circle circle1("yellow", 2.0);
    Circle circle2("black", 1.0);

    std::cout << "Before assignment:" << std::endl;
    circle1.draw();
    circle2.draw();

    circle2 = circle1;  // присваивание

    std::cout << "After assignment:" << std::endl;
    circle1.draw();
    circle2.draw();

    circle2.setRadius(3.0);  // изменяем только circle2

    std::cout << "After modifying circle2:" << std::endl;
    circle1.draw();
    circle2.draw();
}

int main() {
    std::cout << "=== Лабораторная работа 2: Объекты и Классы ===" <<
std::endl;

    demonstrateObjectCreation();
    demonstrateInheritance();
    demonstratePointers();
    demonstrateCompositionDifference();
```

```
    demonstrateAdvancedConcepts();

    std::cout << "\n=== Завершение программы ===" << std::endl;
    std::cout << "Все автоматические объекты будут уничтожены..." <<
std::endl;

    return 0;
}
```