Министерство образования Республики Беларусь  
Учреждение образования  
«Брестский государственный технический университет»  
Кафедра ИИТ

Лабораторная работа №3

«Иерархии наследования, динамический полиморфизм в приложениях на языках С++, С#  
По дисциплине: «Объектное моделирование интеллектуальных систем»

Выполнил:  
Студент 3 курса  
Группы ИИ-21  
Литвинюк Т. В.

Проверила:  
Демидович А.Г.

Брест 2023

**Цель работы:**

1. Изучить особенности организации и реализации множественного наследования классов.

2. Изучить варианты организации иерархии классов с использованием отношений агрегации, композиции, использования.

3. Изучить проектирование иерархий классов с использованием динамического полиморфизма.

Множественное наследование

Анализировать предметную область и разработать классы для демонстрации всех видов наследования. Для этого:

- описать исходный класс. Состав минимальный, включая конструкторы, деструктор, перегруженный оператор. Включить в классы члены-данные, требующие динамического распределения памяти;

- для заданной предметной области разработать дополнительные классы для демонстрации всех видов наследования;

- описать классы на UML (язык - русский);

- описать диаграмму классов

(уровень ассоциаций) на UML (язык - русский) без раскрытия классов;

- уточнить диаграмму классов (типы отношений) на UML (язык - русский) без раскрытия классов;

- описать уточненную диаграмму классов UML с раскрытием классов;

- специфицировать, прокомментировать при необходимости данные и методы класса.

Организация иерархий классов

Для указанной предметной области:

– реализовать иерархии классов, используя наследование и используя агрегацию (состав минимальный, включая конструкторы, деструктор, перегруженный оператор):

- для каждого описать диаграмму UML (язык - русский);

- описать диаграмму UML (язык - английский);

- специфицировать данные и методы класса;

- разработать диаграмму классов (уровень ассоциаций);

- уточнить диаграмму классов (типы отношений). Программировать.

Реализовать приложение для обработки объектов предметной области MIX (целое и вещественное), использовав агрегацию и наследование для организации иерархии классов.

Динамический полиморфизм

1. Продемонстрировать для своей предметной области (без применения виртуальных функций) использование собственных указателей классов и указателей базовых классов для доступа к членам классов иерархии со стороны объектов.
2. Повторить предыдущий пункт с применением виртуальных функций.
3. Реализовать приложение с использованием динамического полиморфизма. Вариант реализации – консольный с управлением через простейшее меню.

**1 (Множественное наследование).**

#include <iostream>

#include <stdio.h>

#include <string.h>

class Shape {

public:

    double x, y;

    char \* name;

    Shape(){

        x = y = 0;

        name = nullptr;

    }

    Shape(double x, double y, const char \*name) : x(x), y(y) {

        this->x = x; this->y = y; this->name = new char[strlen(name)];

        for (int i = 0; i < strlen(name); i++){

            this->name[i] = name[i];

        }

    }

    Shape &operator=(const Shape& obj){

        this->x = obj.x; this->y = obj.y;

        delete [] this->name;

        size\_t len = strlen(obj.name);

        this->name = new char[len];

        for (int i = 0; i < strlen(name); i++){

            this->name[i] = obj.name[i];

        }

    }

    ~Shape() {

        delete [] name;

    }

};

#include <cmath>

class Circle : public Shape {

public:

    double radius;

    Circle() : Shape(), radius(0) {}

    Circle(double x, double y, const char\* name, double radius) : Shape(x, y, name), radius(radius) {}

    double area() const {

        return 3.141592 \* radius \* radius;

    }

    ~Circle() {}

};

class ColoredShape : public Circle {

public:

    std::string color;

    ColoredShape() : Circle(), color("") {}

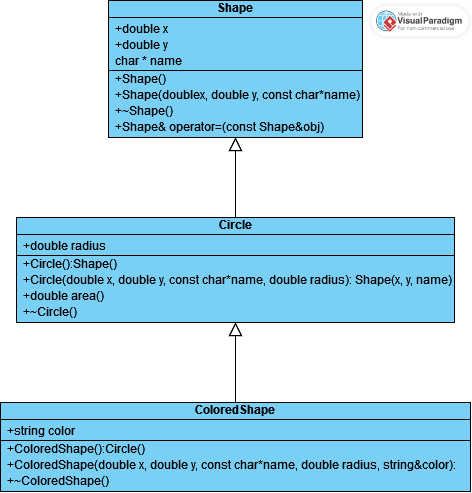
    ColoredShape(double x, double y, const char\* name, double radius, const std::string& color)

        : Circle(x, y, name, radius), color(color) {}

    ~ColoredShape() {}

};

**UML-диаграмма, описывающая классы и наследования**



**2 (Организация иерархий классов).**

Так как структура всех UML будет идентична предыдущему пункту, нет нужды повторно их повторно рисовать.

**Код программы:**

#include <iostream>

#include <vector>

// Базовый класс для точек

class Point {

public:

    double x, y;

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

    void print() const {

        std::cout << "Point at (" << x << ", " << y << ")" << std::endl;

    }

};

// Класс для линии, состоящей из двух точек

class Line {

public:

    Point start, end;

    Line(const Point& start, const Point& end) : start(start), end(end) {}

    void print() const {

        std::cout << "Line from (" << start.x << ", " << start.y << ") to (" << end.x << ", " << end.y << ")" << std::endl;

    }

};

// Класс для прямоугольника, заданного двумя диагональными угловыми точками

class Rectangle {

public:

    Point topLeft, bottomRight;

    Rectangle(const Point& topLeft, const Point& bottomRight) : topLeft(topLeft), bottomRight(bottomRight) {}

    void print() const {

        std::cout << "Rectangle from (" << topLeft.x << ", " << topLeft.y << ") to ("

                  << bottomRight.x << ", " << bottomRight.y << ")" << std::endl;

    }

};

// Класс, представляющий собой коллекцию фигур

class Drawing {

public:

    std::vector<Point> points;

    std::vector<Line> lines;

    std::vector<Rectangle> rectangles;

    void print() const {

        std::cout << "Drawing contains:" << std::endl;

        for (const auto& point : points) {

            point.print();

        }

        for (const auto& line : lines) {

            line.print();

        }

        for (const auto& rectangle : rectangles) {

            rectangle.print();

        }

    }

};

int main() {

    // Пример использования иерархии классов

    Point p1(1.0, 2.0);

    Point p2(3.0, 4.0);

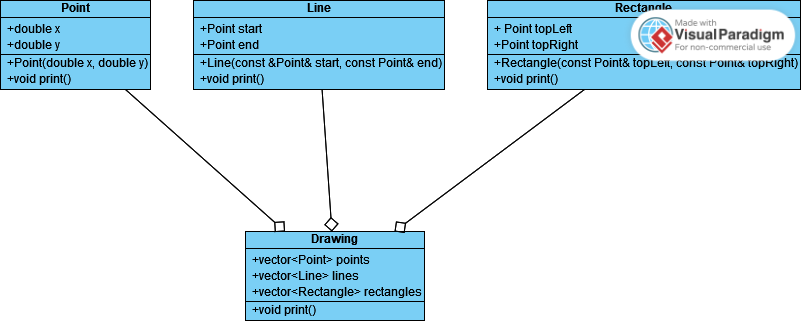
    Line line(p1, p2);

    Rectangle rect(p1, p2);

    // Пример использования агрегации (Drawing содержит векторы фигур)

    Drawing drawing;

    drawing.points.push\_back(p1);

    drawing.points.push\_back(p2);

    drawing.lines.push\_back(line);

    drawing.rectangles.push\_back(rect);

    // Выводим содержимое рисунка

    drawing.print();

    return 0;

}

**3 (Динамический полиморфизм).**

1. Демонстрация использования собственных указателей классов и указателей базовых классов для доступа к членам классов иерархии со стороны объектов (без применения виртуальных функций):

**Код программы:**

#include <iostream>

class Shape {

public:

double x, y;

Shape(double x, double y) : x(x), y(y) {}

void print() {

std::cout << "Shape at (" << x << ", " << y << ")" << std::endl;

}

};

class Circle : public Shape {

public:

double radius;

Circle(double x, double y, double radius) : Shape(x, y), radius(radius) {}

void print() {

std::cout << "Circle at (" << x << ", " << y << ") with radius " << radius << std::endl;

}

};

int main() {

Shape shape(1.0, 2.0);

Circle circle(3.0, 4.0, 5.0);

Shape\* shapePtr = &shape;

Shape\* circlePtr = &circle;

shapePtr->print();

circlePtr->print();

return 0;

}

2. Демонстрация использования виртуальных функций:

**Код программы:**

#include <iostream>

class Shape {

public:

double x, y;

Shape(double x, double y) : x(x), y(y) {}

virtual void print() {

std::cout << "Shape at (" << x << ", " << y << ")" << std::endl;

}

virtual ~Shape() {}

};

class Circle : public Shape {

public:

double radius;

Circle(double x, double y, double radius) : Shape(x, y), radius(radius) {}

void print() override {

std::cout << "Circle at (" << x << ", " << y << ") with radius " << radius << std::endl;

}

};

int main() {

Shape shape(1.0, 2.0);

Circle circle(3.0, 4.0, 5.0);

Shape\* shapePtr = &shape;

Shape\* circlePtr = &circle;

shapePtr->print();

circlePtr->print();

return 0;

}

3. Реализовать приложение с использованием динамического полиморфизма через консольное меню:

**Код программы:**

#include <iostream>

#include <vector>

#include <cstring>

class Shape {

public:

double x, y;

char\* name;

Shape(double x, double y, const char\* name) : x(x), y(y) {

this->name = new char[strlen(name) + 1];

strcpy(this->name, name);

}

void print() const {

std::cout << "Shape at (" << x << ", " << y << ") with name: " << name << std::endl;

}

~Shape() {

delete[] name;

}

};

class Circle : public Shape {

public:

double radius;

Circle(double x, double y, const char\* name, double radius) : Shape(x, y, name), radius(radius) {}

void print() const {

std::cout << "Circle at (" << x << ", " << y << ") with name: " << name << " and radius " << radius << std::endl;

}

};

int main() {

std::vector<Shape\*> shapes;

char choice;

do {

std::cout << "Menu:" << std::endl;

std::cout << "1. Add a Shape" << std::endl;

std::cout << "2. Add a Circle" << std::endl;

std::cout << "3. Print all shapes" << std::endl;

std::cout << "4. Exit" << std::endl;

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

std::cin >> choice;

switch (choice) {

case '1': {

double x, y;

char name[100];

std::cout << "Enter x and y coordinates: ";

std::cin >> x >> y;

std::cout << "Enter name: ";

std::cin >> name;

shapes.push\_back(new Shape(x, y, name));

break;

}

case '2': {

double x, y, radius;

char name[100];

std::cout << "Enter x and y coordinates: ";

std::cin >> x >> y;

std::cout << "Enter name: ";

std::cin >> name;

std::cout << "Enter radius: ";

std::cin >> radius;

shapes.push\_back(new Circle(x, y, name, radius));

break;

}

case '3':

for (const auto& shape : shapes) {

shape->print();

}

break;

case '4':

// Clean up allocated memory

for (const auto& shape : shapes) {

delete shape;

}

break;

default:

std::cout << "Invalid choice. Try again." << std::endl;

}

} while (choice != '4');

return 0;

}

**Вывод:**

1. Изучил особенности организации и реализации множественного наследования классов.

2. Изучил варианты организации иерархии классов с использованием отношений агрегации, композиции, использования.

3. Изучил проектирования иерархий классов с использованием динамического полиморфизма.