Лабораторная работа: Иерархия геометрических объектов

Цель:

Реализовать иерархию геометрических фигур с использованием механизмов ООП: абстракции, полиморфизма, инкапсуляции и наследования.

Структура программы:

- Абстрактный класс Point содержит координаты и виртуальные методы: draw(), move(), rotate(), hide().
- Класс Line реализует линию с начальной и конечной точками.
- Класс Square реализует базовый квадрат с виртуальным наследованием.
- Классы Rectangle, Rhombus, Parallelogram наследуются от Square и реализуют свои особенности отрисовки.
- Parallelogram использует виртуальное наследование от Square.
- Метод render демонстрирует позднее связывание (runtime polymorphism) при работе с указателями на базовый тип Point.

Проверка доступа:

- Переменные `x`, `y` защищённые (protected) — доступны в наследниках.
- Прямой доступ из main невозможен — обеспечивается инкапсуляция.

Результат:

Программа успешно компилируется, демонстрирует поведение всех геометрических объектов. Использованы виртуальные функции, абстрактные классы, виртуальное наследование и полиморфизм.

```
#include <iostream>
#include <cmath>
using namespace std;

const double PI = 3.141592653589793;

// 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 move(double dx, double dy) {
      x += dx;
      y += dy;
   }

   virtual void rotate(double angle) {
      double rad = angle * PI / 180.0;
      double newX = x * cos(rad) - y * sin(rad);
      double newY = x * sin(rad) + y * cos(rad);
      x = newX;
      y = newY;
   }

   virtual void hide() const = 0;
   virtual ~Point() {}
};

// 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 << "Drawing Line from (" << x << ", " << y << ") to (" << x2 << ", " << y2 << ")\n";
   }

   void hide() const override {
      cout << "Hiding Line\n";
   }

   void move(double dx, double dy) override {
      Point::move(dx, dy);
```

```
        x2 += dx;
        y2 += dy;
    }

    void rotate(double angle) override {
        Point::rotate(angle);
        double rad = angle * PI / 180.0;
        double newX2 = x2 * cos(rad) - y2 * sin(rad);
        double newY2 = x2 * sin(rad) + y2 * cos(rad);
        x2 = newX2;
        y2 = newY2;
    }
};

// Виртуальное наследование от Базового Квадрата
class Square : virtual public Point {
protected:
    double side;

public:
    Square(double x, double y, double side)
        : Point(x, y), side(side) {}

    void draw() const override {
        cout << "Drawing Square with side " << side << " at (" << x << ", " << y << ")\n";
    }

    void hide() const override {
        cout << "Hiding Square\n";
    }

    void rotate(double angle) override {
        Point::rotate(angle);
        cout << "Rotating Square by " << angle << " degrees\n";
    }
};

// Класс Прямоугольник
class Rectangle : public Square {
protected:
    double height;

public:
    Rectangle(double x, double y, double width, double height)
        : Square(x, y, width), height(height) {}

    void draw() const override {
        cout << "Drawing Rectangle at (" << x << ", " << y << ") with width " << side << " and height " <<
height << "\n";
    }
```

```
    void hide() const override {
        cout << "Hiding Rectangle\n";
    }

    void rotate(double angle) override {
        Point::rotate(angle);
        cout << "Rotating Rectangle by " << angle << " degrees\n";
    }
};

// Класс Ромб
class Rhombus : public Square {
public:
    Rhombus(double x, double y, double side)
        : Square(x, y, side) {}

    void draw() const override {
        cout << "Drawing Rhombus at (" << x << ", " << y << ") with side " << side << "\n";
    }

    void hide() const override {
        cout << "Hiding Rhombus\n";
    }
};

// Класс Параллелограмм (с виртуальным наследованием от Square)
class Parallelogram : public virtual Square {
protected:
    double height;

public:
    Parallelogram(double x, double y, double side, double height)
        : Point(x, y), Square(x, y, side), height(height) {}

    void draw() const override {
        cout << "Drawing Parallelogram at (" << x << ", " << y << ") with side " << side << " and height "
<< height << "\n";
    }

    void hide() const override {
        cout << "Hiding Parallelogram\n";
    }
};

// Демонстрация позднего связывания
void render(Point* shape) {
    shape->draw();
    shape->move(5, 5);
    shape->rotate(45);
    shape->hide();
}
```

```
int main() {
    Line l(0, 0, 10, 10);
    Rectangle r(1, 1, 4, 6);
    Rhombus rh(2, 2, 5);
    Parallelogram p(3, 3, 6, 4);

    render(&l);
    render(&r);
    render(&rh);
    render(&p);

    return 0;
}
```

Вывод:
Работа выполнена в полном объёме, цели достигнуты. Программа легко масштабируется для добавления новых фигур и операций.