Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
I I семестр
Задание 3: «Наследование, полиморфизм»

Группа:	М8О-208Б-18, №2
Студент:	Алексеева Мария Алексеевна
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	27.10.2019

Москва, 2019

1. **Тема**: Нсаледование. Полиморфизм

2. **Цель работы**: Изучение механизмов работы с наследованием C++.

3. **Задание** (*вариант № 2*):

Разработать классы согласно варианту задания. Классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать общий набор методов:

- Вычисление геометрического центра фигуры
- Вывод в стандартный поток std::cout координат вершин фигуры
- Вычисление площади фигуры

Создать программу, которая позволяет:
Вводить из стандартного вода std::cin фигуры, согласно варианту задания
Сохранять созданные фигуры в динамический массив std::vector<Figure*>
Вызывыть для всего массива общие функции (1 — 3)
Необходимо уметь вычислять общую площадь фигур в массиве
Удалять из массива фигуру по индексу
Фигуры (Вариант 2):
Квадрат, прямоугольник,трапеция.

4. **Адрес репозитория на GitHub** https://github.com/PowerMasha/oop_exercise_03

5. **Код программы на C++**

main.cpp

```
#include <iostream>
#include "figure.h"
#include <vector>
#include <string>

void read_typeF(std::vector<Figure *>& fig)
{
  int typeF;
  Rectangle *re = nullptr;
  Trapeze *t = nullptr;
  Square *s = nullptr;

  std::cout << "Выберете фигуру: 1-квадрат; 2-прямоугольник; 3-трапеция. \n";
  std::cin >> typeF;
  switch (typeF) {
  case 1:
 try{
     s = new Square(std::cin);
    } catch(std::logic_error& err){
     std::cout << err.what() << std::endl;
     break;
    }
    fig.push_back(dynamic_cast<Figure*>(s));
    break;
  case 2:
```

```
      try {
           re = new Rectangle(std::cin);
      } catch(std::logic_error& err){
      std::cout << err.what() << std::endl;
      break;
    }
    fig.push_back(dynamic_cast<Figure*>(re));
    break;
  case 3:
 try{
      t = new Trapeze(std::cin);
    }
      catch(std::logic_error& err){
      std::cout << err.what() << std::endl;
      break;
    }
    fig.push_back(dynamic_cast<Figure*>(t));
    break;
  default:
    std::cout << "Ошибка\n";  }
}

int main(){
  unsigned int index;
  double Tarea = 0;
  std::string operation;
  std::vector<Figure*> fig;
  std::cout << "Operations: add / delete / out / quit\n";

  while (std::cin >> operation) {
    if (operation == "add") {
      read_typeF(fig);
    }
    else if (operation == "delete") {
      std::cin >> index;
      delete fig[index];
      for (; index < fig.size() - 1; ++index) {
      fig[index] = fig[index + 1];
      }
      fig.pop_back();
    }
    else if (operation == "out") {
      Tarea = 0;
      for (unsigned int i = 0; i < fig.size(); i++) {
      std::cout << i << ":\n";
      std::cout << "Area: " << fig[i]->area() << std::endl;
      std::cout << "Center: " << fig[i]->center() << std::endl;
      std::cout << "Coordinates: ";
      fig[i]->print(std::cout);
      std::cout << std::endl;
      Tarea += fig[i]->area();
      }
```

```
      std::cout << "Total area: " << Tarea << std::endl;
    }
    else if (operation == "quit"){
      for (unsigned int i = 0; i < fig.size(); ++i) {
      delete fig[i];
      }
      return 0;
    }
    else {
      std::cout << "Ошибка\n";
    }
  }
}

figure.cpp
#include "figure.h"
#include <cmath>
//Точка
Point::Point() : x{0}, y{0} {}

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

double Point::X() const {
     return x;
}

double Point::Y() const {
     return y;
}

std::ostream& operator<< (std::ostream& out, const Point& p) {
     out << "(" << p.X() << ";" << p.Y() << ")";
     return out;
}


std::istream& operator>> (std::istream& in, Point& p) {
       in >> p.x >> p.y;
     return in;
}
//Квадрат
Square::Square() : A1{0, 0}, B2{0, 0}, C3{0, 0}, D4{0, 0} {}

Square::Square(std::istream& in) {
  in >> A1 >> B2 >> C3 >> D4;
  double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB;
  a = sqrt((B2.X()- A1.X()) * (B2.X() - A1.X()) + (B2.Y() - A1.Y()) * (B2.Y() - A1.Y()));
  b = sqrt((C3.X()- B2.X()) * (C3.X() - B2.X()) + (C3.Y() - B2.Y()) * (C3.Y() - B2.Y()));
  c = sqrt((C3.X()- D4.X()) * (C3.X() - D4.X()) + (C3.Y() - D4.Y()) * (C3.Y() - D4.Y()));
  d = sqrt((D4.X()- A1.X()) * (D4.X() - A1.X()) + (D4.Y() - A1.Y()) * (D4.Y() - A1.Y()));
  d1 = sqrt((B2.X()- D4.X()) * (B2.X() - D4.X()) + (B2.Y() - D4.Y()) * (B2.Y() - D4.Y()));
  d2 = sqrt((C3.X()- A1.X()) * (C3.X() - A1.X()) + (C3.Y() - A1.Y()) * (C3.Y() - A1.Y()));
```

```
  ABC = (a * a + b * b - d2 * d2) / 2 * a * b;
  BCD = (b * b + c * c - d1 * d1) / 2 * b * c;
  CDA = (d * d + c * c - d2 * d2) / 2 * d * c;
  DAB = (a * a + d * d - d1 * d1) / 2 * a * d;
  if(ABC != BCD || ABC != CDA || ABC != DAB || a!=b || a!=c || a!=d )
    throw std::logic_error("Это не квадрат!");
}

double Square::area() const{
double p = abs(A1.X()*B2.Y()+B2.X()*C3.Y()+C3.X()*D4.Y()+D4.X()*A1.Y()-
B2.X()*A1.Y()-C3.X()*B2.Y()-D4.X()*C3.Y()-A1.X()*D4.Y())/2;
  return p;
}

Point Square::center() const{
  return Point{(A1.X() + B2.X() + C3.X() + D4.X()) / 4, (A1.Y() + B2.Y() + C3.Y() + D4.Y()) /
4};
}

std::ostream& Square::print(std::ostream& out) const{
  out << A1 << " " << B2 << " " << C3 << " " << D4;
  return out;
}
//Прямоугольник
Rectangle::Rectangle() : A1{0, 0}, B2{0, 0}, C3{0, 0}, D4{0, 0} {}

Rectangle::Rectangle(std::istream& in) {
  in >> A1 >> B2 >> C3 >> D4;
  double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB;
  a = sqrt((B2.X()- A1.X()) * (B2.X() - A1.X()) + (B2.Y() - A1.Y()) * (B2.Y() - A1.Y()));
  b = sqrt((C3.X()- B2.X()) * (C3.X() - B2.X()) + (C3.Y() - B2.Y()) * (C3.Y() - B2.Y()));
  c = sqrt((C3.X()- D4.X()) * (C3.X() - D4.X()) + (C3.Y() - D4.Y()) * (C3.Y() - D4.Y()));
  d = sqrt((D4.X()- A1.X()) * (D4.X() - A1.X()) + (D4.Y() - A1.Y()) * (D4.Y() - A1.Y()));
  d1 = sqrt((B2.X()- D4.X()) * (B2.X() - D4.X()) + (B2.Y() - D4.Y()) * (B2.Y() - D4.Y()));
  d2 = sqrt((C3.X()- A1.X()) * (C3.X() - A1.X()) + (C3.Y() - A1.Y()) * (C3.Y() - A1.Y()));
  ABC = (a * a + b * b - d2 * d2) / 2 * a * b;
  BCD = (b * b + c * c - d1 * d1) / 2 * b * c;
  CDA = (d * d + c * c - d2 * d2) / 2 * d * c;
  DAB = (a * a + d * d - d1 * d1) / 2 * a * d;
  if(ABC != BCD || ABC != CDA || ABC != DAB)
    throw std::logic_error("Это не прямоугольник!");
}

double Rectangle::area() const{
double p = abs(A1.X()*B2.Y()+B2.X()*C3.Y()+C3.X()*D4.Y()+D4.X()*A1.Y()-
B2.X()*A1.Y()-C3.X()*B2.Y()-D4.X()*C3.Y()-A1.X()*D4.Y())/2;
  return p;
}

Point Rectangle::center() const{
  return Point{(A1.X() + B2.X() + C3.X() + D4.X()) / 4, (A1.Y() + B2.Y() + C3.Y() + D4.Y()) /
4};
```

```
}

std::ostream& Rectangle::print(std::ostream& out) const{
  out << A1 << " " << B2 << " " << C3 << " " << D4;
  return out;
}
//Трапеция
Trapeze::Trapeze() : A1{0, 0}, B2{0, 0}, C3{0, 0}, D4{0,0} {}

Trapeze::Trapeze(std::istream& in) {
  in >> A1 >> B2 >> C3 >> D4;
  double a, c;
  a = sqrt(pow((B2.X()- A1.X()),2) + pow((B2.Y() - A1.Y()),2));
  c = sqrt(pow((C3.X()- D4.X()),2) + pow((C3.Y() - D4.Y()),2));
  if(a != c || (C3.Y()-B2.Y())/(C3.X()-B2.X())!=(D4.Y()-A1.Y())/(D4.X()-A1.X()))
    throw std::logic_error("Это не трапеция!");
}


double Trapeze::area() const{
double p = abs(A1.X()*B2.Y()+B2.X()*C3.Y()+C3.X()*D4.Y()+D4.X()*A1.Y()-
B2.X()*A1.Y()-C3.X()*B2.Y()-D4.X()*C3.Y()-A1.X()*D4.Y())/2;
  return p;

}

Point Trapeze::center() const
{
  double a = sqrt((C3.X() - B2.X()) * (C3.X() - B2.X()) + (B2.Y() - C3.Y()) * (B2.Y() - C3.Y()));
  double b = sqrt((B2.X() - A1.X()) * (B2.X() - A1.X()) + (B2.Y() - A1.Y()) * (B2.Y() -
A1.Y()));
  double l = sqrt((D4.X() - A1.X()) * (D4.X() - A1.X()) + (A1.Y() - D4.Y()) * (A1.Y() -
D4.Y()));
  double c = (l - a) / 2;
  double h = sqrt((b * b) - (c * c));
  double y_ = (2 * l + a) * h / (a + l) / 3;
  if (B2.X() == C3.X() && D4.X() < C3.X())
    return Point{D4.X() + h - y_, (A1.Y() + B2.Y() + C3.Y() + D4.Y()) / 4};
  if (B2.X() == C3.X() && C3.X() < D4.X())
    return Point{C3.X() + h - y_, (A1.Y() + B2.Y() + C3.Y() + D4.Y()) / 4};
  return Point{(A1.X() + B2.X() + C3.X() + D4.X()) / 4, (B2.Y() + C3.Y()) / 2 - ((B2.Y() +
C3.Y()) / 2 - (D4.Y() + A1.Y()) / 2) * y_ / h};
}

std::ostream& Trapeze::print(std::ostream& out) const
{
  out << A1 << " " << B2 << " " << C3 << " " << D4;
  return out;
}
```

Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>

class Point {
public:
    Point();
    Point(double x, double y);
    double X() const;
    double Y() const;
    friend std::ostream& operator<< (std::ostream& out, const Point& p);
    friend std::istream& operator>> (std::istream& in, Point& p);
private:
    double x;
    double y;
};

class Figure {
public:
    virtual double area() const = 0;
    virtual Point center() const = 0;
    virtual std::ostream& print(std::ostream& out) const = 0;
    virtual ~Figure() = default;
};


class Square : public Figure {
public:
    Square();
    Square(std::istream& in);
    double area() const override;
    Point center() const override;
    std::ostream& print(std::ostream& out) const override;
private:
    Point A1;
    Point B2;
    Point C3;
    Point D4;
};

class Rectangle : public Figure {
public:
    Rectangle();
    Rectangle(std::istream& in);
    double area() const override;
    Point center() const override;
    std::ostream& print(std::ostream& out) const override;
private:
    Point A1;
```

```
    Point B2;
    Point C3;
    Point D4;
};

class Trapeze : public Figure {
public:
     Trapeze();
    Trapeze(std::istream& in);
    double area() const override;
    Point center() const override;
    std::ostream& print(std::ostream& out) const override;
private:
    Point A1;
    Point B2;
    Point C3;
    Point D4;
};


#endif
```

CmakeLists.txt

```
project(lab3)

add_executable(oop_exercise_03
  main.cpp
  figure.cpp
  )

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra")

set_target_properties(oop_exercise_03 PROPERTIES CXX_STANDART 14
CXX_STANDART_REQUIRED ON)
```

6. Набор testcases

test_01.txt

```
add
1
1 1
1 5
5 5
5 1
out
add
2
2 2
```

2 5
4 5
4 2
out
delete
1
out

test_02.txt
add
3
0 0
1 5
4 7
9 6
add
2
0 0
0 4
2 4
2 0
add
1
1 1
1 6
6 6
6 1
out
delete
2
out

test_03.txt

add
2
0 0
0 2
1 2
1 0
out
add
3
2 2
3 4
5 4
6 2
out
delete
0
out

7. Результаты выполнения тестов

```
masha@masha-VirtualBox:~/2kurs/oop_exercise_03/tmp$ ./oop_exercise_03 <
~/2kurs/oop_exercise_03/test_01.txt
Operations: add / delete / out / quit
Выберете фигуру: 1-квадрат; 2-прямоугольник; 3-трапеция.
0:
Area: 16
Center: (3;3)
Coordinates: (1;1) (1;5) (5;5) (5;1)
Total area: 16
Выберете фигуру: 1-квадрат; 2-прямоугольник; 3-трапеция.
0:
Area: 16
Center: (3;3)
Coordinates: (1;1) (1;5) (5;5) (5;1)
1:
Area: 6
Center: (3;3.5)
Coordinates: (2;2) (2;5) (4;5) (4;2)
Total area: 22
0:
Area: 16
Center: (3;3)
Coordinates: (1;1) (1;5) (5;5) (5;1)
Total area: 16
masha@masha-VirtualBox:~/2kurs/oop_exercise_03/tmp$ ./oop_exercise_03 <
~/2kurs/oop_exercise_03/test_02.txt
Operations: add / delete / out / quit
Выберете фигуру: 1-квадрат; 2-прямоугольник; 3-трапеция.
Выберете фигуру: 1-квадрат; 2-прямоугольник; 3-трапеция.
Выберете фигуру: 1-квадрат; 2-прямоугольник; 3-трапеция.
0:
Area: 26
Center: (3.5;4.25)
Coordinates: (0;0) (1;5) (4;7) (9;6)
1:
Area: 8
Center: (1;2)
Coordinates: (0;0) (0;4) (2;4) (2;0)
2:
Area: 25
Center: (3.5;3.5)
Coordinates: (1;1) (1;6) (6;6) (6;1)
Total area: 59
0:
Area: 26
Center: (3.5;4.25)
Coordinates: (0;0) (1;5) (4;7) (9;6)
1:
Area: 8
Center: (1;2)
```

Coordinates: (0;0) (0;4) (2;4) (2;0)
Total area: 34
masha@masha-VirtualBox:~/2kurs/oop_exercise_03/tmp$ ./oop_exercise_03 < ~/2kurs/oop_exercise_03/test_03.txt
Operations: add / delete / out / quit
Выберете фигуру: 1-квадрат; 2-прямоугольник; 3-трапеция.
0:
Area: 2
Center: (0.5;1)
Coordinates: (0;0) (0;2) (1;2) (1;0)
Total area: 2
Выберете фигуру: 1-квадрат; 2-прямоугольник; 3-трапеция.
0:
Area: 2
Center: (0.5;1)
Coordinates: (0;0) (0;2) (1;2) (1;0)
1:
Area: 6
Center: (4;2.88889)
Coordinates: (2;2) (3;4) (5;4) (6;2)
Total area: 8
0:
Area: 6
Center: (4;2.88889)
Coordinates: (2;2) (3;4) (5;4) (6;2)
Total area: 6

8. Объяснение результатов работы программы - вывод

в figure.h задаётся базовый класс Figure задающий общий принцип структуры для классов — наследников — Square, Rectangle, Trapeze**.**

Наследование является одним из основополагающих принципов ООП. В соответствии с ним, класс может использовать переменные и методы другого класса как свои собственные. Класс, который наследует данные, называется подклассом , производным классом или дочерним классом. Класс, от которого наследуются данные или методы, называется суперклассом, базовым классом или родительским классом .

Полиморфизм позволяет использовать одно и то же имя для различных действий, похожих, но технически отличающихся. В данной лабораторной работе он осуществляется посредством виртуальных функций.