Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Объектно-ориентированное программирование»
III Семестр

Задание 7
Вариант 8
Ассинхронное программирование

Студент:	Жерлыгин М.А
Группа:	М8О-208Б-18
Преподаватель:	Журавлев А.А.
Оценка:	
Дата:	
Подпись:	

Москва 2019

1. Код программы на языке C++

point.h

```
#ifndef D_POINT_H_
#define D_POINT_H_

#include <istream>
#include <ostream>

class Point {
    public:
        double x, y;
        Point();
        Point(double a, double b);
        Point& operator=(const Point& other);
        Point operator+(const Point& other);
        Point operator-(const Point& other);
        Point operator/(const double num);
        ~Point() = default;

        friend std::istream& operator>> (std::istream& is, Point& p);
        friend std::ostream& operator<< (std::ostream& os, const Point& p);
};


#endif //D_POINT_H_
```

point.cpp

```
#include "point.h"
#include <cmath>

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

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

Point& Point::operator=(const Point& other) {
  this->x = other.x;
  this->y = other.y;
  return *this;
}

Point Point::operator+(const Point& other) {
  Point result;
  result.x = this->x + other.x;
  result.y = this->y + other.y;
  return result;
}

Point Point::operator-(const Point& other) {
  Point result;
  result.x = this->x - other.x;
  result.y = this->y - other.y;
  return result;
}

Point Point::operator/(const double num) {
  Point result;
  result.x = this->x / num;
  result.y = this->y / num;
  return result;
}

std::istream& operator>> (std::istream& is, Point& p) {
  return is >> p.x >> p.y;
}

std::ostream& operator<< (std::ostream& os, const Point& p) {
  return os << "(" << p.x << ", " << p.y << ")" << std::endl;
}
```

figure.h

```
#ifndef FIGURE_H_
#define FIGURE_H_

#include <fstream>
#include <map>
#include <memory>
#include "point.h"

namespace figure {
    class Figure {
        public:
            virtual Point center() const = 0;
            virtual double area() const = 0;
            virtual void print(std::ostream& os) const = 0;
            virtual void save(std::ofstream& os) const = 0;
            virtual void load(std::ifstream& is) = 0;
            virtual uint32_t get_ID() const = 0;
            virtual ~Figure() = default;
            friend std::ostream& operator<< (std::ostream& os, const Figure& f);
    };
}

enum figure_t {
    OCTAGON,
    TRIANGLE,
    SQUARE
};

class Fact_Interface {
    public:
        virtual std::shared_ptr<figure::Figure> Create_figure() const = 0;
        virtual std::shared_ptr<figure::Figure> Create_figure(uint32_t id, std::istream& is) const = 0;
};



#endif // FIGURE_H_
```

figure.cpp

```
#include "figure.h"

std::ostream& operator<< (std::ostream& os, const figure::Figure& f) {
  f.print(os);
  return os;
}
```

octagon.h

```
#ifndef OCTAGON_H_
#define OCTAGON_H_

#include "figure.h"

namespace figure {

    class Octagon : public Figure {
        private:
            Point coordinate[8];
            uint32_t id_;
        public:
            Octagon();
            Octagon(uint32_t id, std::istream& is);
            Point center() const override;
            double area() const override;
            void print(std::ostream& os) const override;
            uint32_t get_ID() const override;
            void save(std::ofstream& os) const override;
            void load(std::ifstream& is) override;
    };
}
```

```
class Oct_factory: public Fact_Interface {
   public:
      std::shared_ptr<figure::Figure> Create_figure() const override;
      std::shared_ptr<figure::Figure> Create_figure(uint32_t id, std::istream& is) const override;
};

#endif // OCTAGON_H_
```

octagon.cpp

```
#include <iostream>
#include <cmath>
#include "octagon.h"

namespace figure {

Octagon::Octagon(): id_(0) {
   for(int i = 0; i < 8; i++) {
      coordinate[i].x = 0.0;
      coordinate[i].y = 0.0;
   }
}

Octagon::Octagon(uint32_t id, std::istream& is): id_(id) {
   for(int i = 0; i < 8; i++) {
      is >> coordinate[i];
   }
}

double Octagon::area() const {
   double result = 0;
   for(int i = 0; i < 7; i++) {
      result += (coordinate[i].x * coordinate[i+1].y) - (coordinate[i+1].x * coordinate[i].y);
   }
   result = std::abs(result + (coordinate[7].x * coordinate[0].y) - (coordinate[0].x * coordinate[7].y));
   return result / 2.0;
}

Point Octagon::center() const {
   Point result;
   for(int i = 0; i < 8; i++) {
      result = result + coordinate[i];
   }
   return result / 8.0;
}

void Octagon::print(std::ostream& os) const {
   os << "==================================\n";
   os << "id - " << id_ << "\nFigure - Octagon" << "\nArea: " << area() << "\nCenter: " << center();
   std::cout << "Octagon coordinates:" << std::endl;
   os << this->coordinate[0];
```

```
    os << this->coordinate[1];
    os << this->coordinate[2];
    os << this->coordinate[3];
    os << this->coordinate[4];
    os << this->coordinate[5];
    os << this->coordinate[6];
    os << this->coordinate[7];
}

uint32_t Octagon::get_ID() const {
    return id_;
}

void Octagon::save(std::ofstream& os) const {
    figure_t t = OCTAGON;
    os.write(reinterpret_cast<char*>(&t), sizeof(t));
    os.write((char*)(&id_), sizeof(id_));
    for(int i = 0; i <= 7 ; i++) {
        os << coordinate[i].x << ' ' << coordinate[i].y;
        if(i != 7) {
            os << "\t";
        }
    }
}

void Octagon::load(std::ifstream& is) {
    is.read((char*)(&id_), sizeof(id_));
    for(int i = 0; i <= 7; i++) {
        is >> coordinate[i].x >> coordinate[i].y;
    }
}

}// end of namespace

std::shared_ptr<figure::Figure> Oct_factory::Create_figure() const {
    return std::shared_ptr<figure::Figure>(new figure::Octagon());
}

std::shared_ptr<figure::Figure> Oct_factory::Create_figure(uint32_t id, std::istream& is) const {
    return std::shared_ptr<figure::Figure>(new figure::Octagon(id, is));
}
```

triangle.h

```
#ifndef D_TRIANGLE_H_
#define D_TRIANGLE_H_
```

```
#include "figure.h"

namespace figure {

class Triangle : public Figure {
   public:
      Point coordinate[3];
      uint32_t id_;
      Triangle();
      Triangle(uint32_t id, std::istream& is);
      Point center() const override;
      double area() const override;
      void print(std::ostream& os) const override;
      uint32_t get_ID() const override;
      void save(std::ofstream& os) const override;
      void load(std::ifstream& is) override;
};
} // end of namespace


class Tri_factory: public Fact_Interface {
   public:
   std::shared_ptr<figure::Figure> Create_figure() const override;
   std::shared_ptr<figure::Figure> Create_figure(uint32_t id, std::istream& is) const override;
};

#endif //D_TRIANGLE_H_
```

triangle.cpp

```
#include <iostream>
#include <cmath>
#include "triangle.h"

namespace figure {

Triangle::Triangle(): id_(0) {
   //coordinate = new Point[3];
   for(int i = 0; i < 3; i++) {
      coordinate[i].x = 0.0;
      coordinate[i].y = 0.0;
   }
}

Triangle::Triangle(uint32_t id, std::istream& is): id_(id) {
   //coordinate = new Point[3];
   for(int i = 0; i < 3; i++) {
      is >> coordinate[i];
   }
```

```
    double AB, BC, AC;
    AB = sqrt(pow(coordinate[1].x - coordinate[0].x, 2) + pow(coordinate[1].y - coordinate[0].y, 2));
    BC = sqrt(pow(coordinate[2].x - coordinate[1].x, 2) + pow(coordinate[2].y - coordinate[1].y, 2));
    AC = sqrt(pow(coordinate[2].x - coordinate[0].x, 2) + pow(coordinate[2].y - coordinate[0].y, 2));
    if(AB + BC <= AC || AB + AC <= BC || BC + AC <= AB) throw std::logic_error("This is not Triange");
}

Point Triangle::center() const {
    Point result;
    for(int i = 0; i < 3; i++) {
        result = result + coordinate[i];
    }
    return result / 3.0;
}

double Triangle::area() const {
    return fabs(((coordinate[0].x - coordinate[2].x) * (coordinate[1].y - coordinate[2].y) - (coordinate[1].x -
coordinate[2].x) * (coordinate[0].y - coordinate[2].y)) / 2);
}

void Triangle::print(std::ostream& os) const {
    os << "==================================\n";
    os << "id - " << id_ << "\nFigure - Triangle" << "\nArea: " << area() << "\nCenter: " << center();
    std::cout << "Triangle coordinates" << std::endl;
    os << Point(coordinate[0].x, coordinate[0].y) << "\n"
    << Point(coordinate[1].x, coordinate[1].y) << "\n"
    << Point(coordinate[2].x, coordinate[2].y) << std::endl;
}

uint32_t Triangle::get_ID() const {
    return id_;
}

void Triangle::load(std::ifstream& is) {
    is.read((char*)(&id_), sizeof(id_));
    for (int i = 0; i < 3; ++i) {
        is >> coordinate[i].x  >> coordinate[i].y;
    }
}

void Triangle::save(std::ofstream& os) const {
    figure_t t = TRIANGLE;
    os.write(reinterpret_cast<char*>(&t), sizeof(t));
    os.write((char*)(&id_), sizeof(id_));
    for (int i = 0; i <= 2; ++i) {
        os << coordinate[i].x << ' ' << coordinate[i].y;
        if (i != 2) os << '\t';
    }
}

}// end of namespace
```

```
std::shared_ptr<figure::Figure> Tri_factory::Create_figure() const {
   return std::shared_ptr<figure::Figure>(new figure::Triangle());
}

std::shared_ptr<figure::Figure> Tri_factory::Create_figure(uint32_t id, std::istream& is) const {
   return std::shared_ptr<figure::Figure>(new figure::Triangle(id, is));
}
```

square.h

```
#ifndef D_Square_H_
#define D_Square_H_

#include "figure.h"

namespace figure {
struct Square : public Figure {
  private:
    Point coordinate[4];
    uint32_t id_;
  public:
    Square();
    Square(uint32_t id, std::istream& is);
    Point center() const override;
    double area() const override;
    void print(std::ostream& os) const override;
    void save(std::ofstream& os) const override;
    void load(std::ifstream& is) override;
    uint32_t get_ID() const override;
};
}// end of namespace

class Squ_factory: public Fact_Interface {
  public:
    std::shared_ptr<figure::Figure> Create_figure() const override;
    std::shared_ptr<figure::Figure> Create_figure(uint32_t id, std::istream& is) const override;
};

#endif // D_Square_H_
```

square.cpp

```
#include <iostream>
#include "square.h"
#include <cmath>
#include <algorithm>

namespace figure {
```

```
Square::Square(): id_(0) {
  for(int i = 0; i < 4; i++) {
    coordinate[i].x = 0.0;
    coordinate[i].y = 0.0;
  }
}

Square::Square(uint32_t id, std::istream& is): id_(id) {
  double a, b, c, d;
  is >> coordinate[0];
  is >> coordinate[1];
  is >> coordinate[2];
  is >> coordinate[3];
  a = sqrt((coordinate[1].x - coordinate[0].x)*(coordinate[1].x - coordinate[0].x) + (coordinate[1].y -
coordinate[0].y)*(coordinate[1].y - coordinate[0].y));
  b = sqrt((coordinate[2].x - coordinate[1].x)*(coordinate[2].x - coordinate[1].x) + (coordinate[2].y -
coordinate[1].y)*(coordinate[2].y - coordinate[1].y));
  c = sqrt((coordinate[3].x - coordinate[2].x)*(coordinate[3].x - coordinate[2].x) + (coordinate[3].y -
coordinate[2].y)*(coordinate[3].y - coordinate[2].y));
  d = sqrt((coordinate[0].x - coordinate[3].x)*(coordinate[0].x - coordinate[3].x) + (coordinate[0].y -
coordinate[3].y)*(coordinate[0].y - coordinate[3].y));
  double d1, d2;
  d1 = sqrt((coordinate[1].x - coordinate[3].x)*(coordinate[1].x - coordinate[3].x) + (coordinate[1].y -
coordinate[3].y)*(coordinate[2].y - coordinate[3].y));
  d2 = sqrt((coordinate[2].x - coordinate[0].x)*(coordinate[2].x - coordinate[0].x) + (coordinate[2].y -
coordinate[0].y)*(coordinate[2].y - coordinate[0].y));
  double ABC = (a * a + b * b - d2 * d2) / (2 * a * b);
  double BCD = (b * b + c * c - d1 * d1) / (2 * b * c);
  double CDA = (c * c + d * d - d1 * d1) / (2 * c * d);
  double DAB = (d * d + a * a - d2 * d2) / (2 * d * a);

  if(ABC != BCD || ABC != CDA || ABC != DAB || a!=b || a!=c || a!=d) throw std::logic_error("It`s not a
square");
  //if((coordinate[1].x - coordinate[2].x != coordinate[1].y - coordinate[2].y) || (coordinate[1].x ==
coordinate[2].x && coordinate[1].y == coordinate[2].y)) throw std::logic_error("This are incorrect
coordinates");
  //if(coordinate[1].x - coordinate[2].x != coordinate[1].y - coordinate[2].y) throw std::logic_error("This is
not square");
}

Point Square::center() const {
  return Point((coordinate[0].x + coordinate[2].x) / 2, (coordinate[0].y + coordinate[2].y) / 2);
}

double Square::area() const {
  //const double dx = coordinate[1].x - coordinate[3].x;
  //const double dy = coordinate[1].y - coordinate[3].y;
  //return std::abs(dx * dy);
  return pow(sqrt((coordinate[0].x - coordinate[3].x)*(coordinate[0].x - coordinate[3].x) + (coordinate[0].y -
coordinate[3].y)*(coordinate[0].y - coordinate[3].y)), 2);
}
```

```
void Square::print(std::ostream& os) const {
  os << "================================\n";
  os << "id - " << id_ << "\nFigure - Square" << "\nArea: " << area() << "\nCenter: " << center();
  std::cout << "Square coordinates:" << std::endl;
  os << coordinate[0] << std::endl;
  os << coordinate[1] << std::endl;
  os << coordinate[2] << std::endl;
  os << coordinate[3] << std::endl;
}

void Square::save(std::ofstream& os) const {
  figure_t t = SQUARE;
  os.write(reinterpret_cast<char*>(&t), sizeof(t));
  os.write((char*)(&id_), sizeof(id_));
  for (int i = 0; i < 2; ++i) {
    os << coordinate[i].x << ' ' << coordinate[i].y;
    if (i != 1) os << '\t';
  }
}

void Square::load(std::ifstream& is) {
  is.read((char*)(&id_), sizeof(id_));
  for (int i = 0; i < 2; ++i) {
    is >> coordinate[i].x  >> coordinate[i].y;
  }
}

uint32_t Square::get_ID() const {
  return id_;
}
}// end of namespace

std::shared_ptr<figure::Figure> Squ_factory::Create_figure() const {
    return std::shared_ptr<figure::Figure>(new figure::Square());
}

std::shared_ptr<figure::Figure> Squ_factory::Create_figure(uint32_t id, std::istream& is) const {
    return std::shared_ptr<figure::Figure>(new figure::Square(id, is));
}
```

sub.h

```
#ifndef SUBSCRIBERS_H
#define SUBSCRIBERS_H

class Factory {
public:
    std::map<std::string, std::shared_ptr<Fact_Interface>> plants;
    Factory() {
        plants.emplace("triangle", std::make_shared<Tri_factory>());
        plants.emplace("square", std::make_shared<Squ_factory>());
        plants.emplace("octagon", std::make_shared<Oct_factory>());
    }
};

class Sub_Interface {
public:
    virtual void output(std::vector<std::shared_ptr<figure::Figure>>&) = 0;
    virtual ~Sub_Interface() = default;
};

class Console_Print : public Sub_Interface {
public:
    void output(std::vector<std::shared_ptr<figure::Figure>>& buffer) override {
        for (auto& figure : buffer) {
            figure->print(std::cout);
        }
    }
};

class DocumentPrint : public Sub_Interface {
private:
    int a;
public:
    DocumentPrint() : a(1) {}
    void output(std::vector<std::shared_ptr<figure::Figure>>& buffer) override {
        std::string file_name = std::to_string(a);
        file_name += ".txt";
        std::ofstream file;
        file.open(file_name);
        if(!file.is_open())
        {
            file.clear();
            file.open(file_name, std::ios::out);
            file.close();
            file.open(file_name);
        }
        for (auto &figure : buffer) {
            figure->print(file);
        }
        ++a;
    }
};

#endif // SUBSCRIBERS_H
```

main.cpp

```
#include <iostream>
#include <thread>
```

```
#include <mutex>
#include <condition_variable>
#include <vector>
#include <memory>
#include <string>
#include "triangle.h"
#include "square.h"
#include "octagon.h"
#include "sub.h"



int main(int args, char* argv[]) {
   if (args < 2) {
      std::cout << "Error, use ./[prog_name] [size of buffer]\n";
      return -1;
   }
   int a = 1;

   long buffer_size = strtol(argv[1], nullptr, 10);
   std::vector<std::shared_ptr<figure::Figure>> buffer;
   buffer.reserve(buffer_size);
   Factory factory;
   std::condition_variable cv;
   std::condition_variable cv2;
   std::string command;
   std::mutex mutex;
   bool done = false;
   std::vector<std::shared_ptr<Sub_Interface>> subs;

   subs.push_back(std::make_shared<Console_Print>());
   subs.push_back(std::make_shared<DocumentPrint>());

   std::thread sub([&]() {
      std::unique_lock<std::mutex> sub_lock(mutex);
      while(!done) {
         cv.wait(sub_lock);
         if (done) {
            cv2.notify_all();
            break;
         }
         for (unsigned int i = 0; i < subs.size(); ++i) {
            subs[i]->output(buffer);
         }
         buffer.resize(0);
         ++a;
         cv2.notify_all();
      }
   });

   while(command != "exit") {
      std::cin >> command;
```

```
        if (command == "exit") {

            done = true;
            cv.notify_all();
            break;

        } else if (command == "triangle" || command == "square" || command == "octagon") {

            auto temp = factory.plants[command]->Create_figure(std::cin);
            std::unique_lock<std::mutex> main_lock(mutex);
            buffer.push_back(temp);

            if (buffer.size() == buffer.capacity()) {
                cv.notify_all();
                cv2.wait(main_lock);
            }

        } else std::cout << "no such figure\n";
    }

    sub.join();
    return 0;
}
```

2. Ссылка на репозиторий на Github

https://github.com/mmaxim2710/oop_exercise_08

3.Набор testcases

1)

```
./a.out 2
triangle 0 0 2 2 0 2
triangle 0 0 2 2 0 2
```

2)

```
./a.out 3
triangle 0 0 0 3 3 3
square 0 0 0 3 3 3 3 0
octagon 1 0 1 4 2 5 5 5 6 3 3 3 0 3 0 1
```

4. Результат выполнения тестов

1)

```
./a.out 2
triangle 0 0 2 2 0 2
triangle 0 0 2 2 0 2
==================================

Figure - Triangle
Area: 2
Center: (0.666667, 1.33333)
Triangle coordinates
(0, 0)

(2, 2)

(0, 2)

==================================

Figure - Triangle
Area: 2
Center: (0.666667, 1.33333)
Triangle coordinates
(0, 0)

(2, 2)

(0, 2)
```

2)

```
==================================

Figure - Triangle
Area: 4.5
Center: (1, 2)
Triangle coordinates
(0, 0)

(0, 3)

(3, 3)
```

```
=================================

Figure - Square
Area: 9
Center: (1.5, 1.5)
Square coordinates:
(0, 0)

(0, 3)

(3, 3)

(3, 0)

=================================

Figure - Octagon
Area: 6
Center: (2.25, 3)
Octagon coordinates:
(1, 0)
(1, 4)
(2, 5)
(5, 5)
(6, 3)
(3, 3)
(0, 3)
(0, 1)
```

5. Объяснение результатов программы

Вследствие работы программа создает 2 потока: поток, считывающий команды и добавляющий фигуры в буфер: если буфер заполняется, этот поток посылает сигнал второму и ждёт его; и поток, вызывающий у подписчиков их методы: один подписчик создает файл и записывает буфер, содержимое которого выводит второй подписчик в консоль. После буфер отчищается, и второй поток посылает сигнал первому о том, что его работа окончена, и первый поток начинает работу сначала. Выход из программы — exit.

Вывод: Проделав данную работу я изучил основы ассинхронного программирования, о принципе — publish-subscibe.