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 8
Вариант 27
Асинхронное программирование

Студент:	Чурсина Н. А.
Группа:	М8О-208Б-18
Преподаватель:	Журавлев А.А.
Оценка:	
Дата:	

Москва 2019

1. Код программы на языке C++

Main.cpp

```
// //  Created by kosha on 2/01/2020.
// //  Copyright © 2019 kosha. All rights reserved.
// //
#include <condition_variable>
#include <fstream>
#include <iostream>
#include <memory>
#include <string>
#include <vector>
#include <thread>
#include <cstdlib>
#include "factory.h"
#include "figures.h"
#include "subscriber.h"// класс, для передачи в поток как функтора, который необходим для выполнения обработки на отдельном потоке

void menu() {
   std::cout  << "M - Print menu\n";
   std::cout << "A - Add figure\n";
   std::cout << "E - Exit\n\n";
}


int main(int argc, char *argv[]){
   if (argc != 2) {
      std::cout << "Enter size \n";
      return 1;
   }

   size_t vector_size = std::atoi(argv[1]); //передаем размер буффера
   Factory factory;

   Subscriber subscriber;
   subscriber.buffer.reserve(vector_size);
   subscriber.processes.push_back(std::make_shared<Console_process>());
   subscriber.processes.push_back(std::make_shared<File_process>());
   std::thread subscriber_thread(std::ref(subscriber));

   menu();
   std::string cmd;
   while(std::cin >> cmd) {
      std::unique_lock<std::mutex> main_lock(subscriber.mtx);
      if (cmd =="M") {
         menu();
         continue;
      } else if (cmd == "E") {
         subscriber.end = true;
         subscriber.cv.notify_all();
         break;
      } else if (cmd == "A"){
```

```
        std::string figure_type;

        for (size_t id = 0; id < vector_size; id++) {
            std::cout << "Enter name of figure: Rectangle[R], Trapezoid[T], Rhombus[Rh]\n";
            std::cin >> figure_type;
            if (figure_type == "R"){
                Vertex *vertices = new Vertex[4];
                std::cout << "Enter the coordinates separated by a space\n";
                for (int i = 0; i < 4; i++) {
                    std::cin >> vertices[i];
                }
                try {
                    subscriber.buffer.push_back(factory.FigureCreate(Rec, vertices, id));
                } catch (std::logic_error &e) {
                    std::cout << e.what() << "\n";
                    id--;
                }
            } else if (figure_type == "T") {
                Vertex *vertices = new Vertex[4];
                std::cout << "Enter the coordinates separated by a space\n";
                for (int i = 0; i < 4; i++) {
                    std::cin >> vertices[i];
                }
                try {
                    subscriber.buffer.push_back(factory.FigureCreate(Trapeze, vertices, id));
                } catch (std::logic_error &e) {
                    std::cout << e.what() << "\n";
                    id--;
                }
            } else if (figure_type == "Rh") {
                Vertex *vertices = new Vertex[4];
                std::cout << "Enter the coordinates separated by a space\n";
                for (int i = 0; i < 4; i++) {
                    std::cin >> vertices[i];
                }
                try {
                    subscriber.buffer.push_back(factory.FigureCreate(Rhomb, vertices, id));
                } catch (std::logic_error &e){
                    std::cout << e.what() << "\n";
                    id--;
                }
            } else { std::cout << "Wrong input. Try another words." << "\n";}
        }
        if (subscriber.buffer.size() == vector_size) {
            subscriber.cv.notify_all(); //уведомляет все ожидающие потоки
            subscriber.cv.wait(main_lock, [&subscriber]() { //блокирует текущий поток до тех пор, пока переменная не
будет пробужена
                return subscriber.success;
            });
            subscriber.success = false;
        }
    }
}
subscriber_thread.join();
```

```
    return 0;
}
```

subscriber.h

```
#ifndef SUBSCRIBER_H
#define SUBSCRIBER_H

#include <string>

struct Subscriber;

struct Subscribers_process {
  virtual void Process(std::vector<std::shared_ptr<Figure>> &buffer) = 0;
  virtual ~Subscribers_process() = default;
};

struct Console_process : Subscribers_process {
  void Process(std::vector<std::shared_ptr<Figure>> &buffer) override {
    for (const auto figure : buffer) {
      figure->Print(std::cout);
    }
  }
};

struct File_process : Subscribers_process {
  size_t name = 0;

  void Process(std::vector<std::shared_ptr<Figure>> &buffer) override {
    std::ofstream os(std::to_string(name));
    for (const auto figure : buffer) {
      figure->Print(os);
    }
    name++;
  }
};

struct Subscriber {
  std::mutex mtx;
  std::condition_variable cv;
  std::vector<std::shared_ptr<Figure>> buffer;
  std::vector<std::shared_ptr<Subscribers_process>> processes;
  std::condition_variable sv;
  bool end=0;
  bool success=0;

  void operator()() {
    for(;;) {
      std::unique_lock<std::mutex> guard(mtx); // lock mutex
      cv.wait(guard, [&](){              // вошедший поток на этом месте ждет пока не выполнится условия
        return buffer.size() == buffer.capacity() || end;
      });
      if (end) {
        break;
```

```
        }
        for (size_t i = 0; i < processes.size(); i++) {
            processes[i]->Process(buffer);
        }
        buffer.clear();
        success = true;
        cv.notify_all();
    }
  }
};

#endif //SUBSCRIBER_H
```

factory.h

```
#ifndef FACTORY_H
#define FACTORY_H 1

#include "figures.h"

class Factory {
public:
  std::shared_ptr<Figure> FigureCreate(FigureType type) const {
    std::shared_ptr<Figure> res;
    if (type == Rec) {
      res = std::make_shared<Rectangle>();
    } else if (type == Rhomb) {
      res = std::make_shared<Rhombus>();
    } else if (type == Trapeze) {
      res = std::make_shared<Trapezoid>();
    }
    return res;
  }

  std::shared_ptr<Figure> FigureCreate(FigureType type, Vertex *vertices, int id) const {
    std::shared_ptr<Figure> res;
    if (type == Rec) {
      res = std::make_shared<Rectangle>(vertices[0], vertices[1], vertices[2], vertices[3], id);
    } else if (type == Rhomb) {
      res = std::make_shared<Rhombus>(vertices[0], vertices[1], vertices[2], vertices[3], id);
    } else if (type == Trapeze) {
      res = std::make_shared<Trapezoid>(vertices[0], vertices[1], vertices[2], vertices[3], id);
    }

    return res;
  }
};

#endif //FACTORY_H
```

figures.h

```
#ifndef FIGURES_H
#define FIGURES_H 1
```

```
#include <iostream>
#include <fstream>
#include <utility> // for pair
#include <memory>
#include <cmath>
#include <stdexcept>

enum FigureType {
  Rec,
  Rhomb,
  Trapeze
};

using Vertex = std::pair<double, double>;


class Figure {
public:
  virtual double Area() const = 0;
  virtual Vertex Center() const = 0;
  virtual std::ostream &Print(std::ostream &out) const = 0;
  virtual void Serialize(std::ofstream &os) const = 0;
  virtual void Deserialize(std::ifstream &is) = 0;
  virtual int getId() const = 0;
  virtual ~Figure() = default;
};



Vertex Get_Center(const Vertex *vertices, int n) {
  double x = 0, y = 0;
  for (int i = 0; i < n; i++) {
     x += vertices[i].first;
     y += vertices[i].second;
  }
  return std::make_pair(x / n, y / n);
}

Vertex operator-(const Vertex &p1, const Vertex &p2) {
  return {p1.first - p2.first, p1.second - p2.second};
}

bool oneline(const Vertex &a, const Vertex &b, const Vertex &c) {
  return ( (c.first - a.first) / (b.first - a.first) == (c.second - a.second) / (b.second - a.second) );
}

bool collinear(const Vertex &a, const Vertex &b, const Vertex &c, const Vertex &d){
  return (b.second-a.second)*(d.first-c.first) - (d.second-c.second)*(b.first-a.first) <= 1e-9;
}

bool perpendic(const Vertex &a, const Vertex &b, const Vertex &c, const Vertex &d){
  using vect = std::pair<double, double>;
  vect AC = c-a;
```

```
  vect BD = d-b;
  double dotProduct = AC.first*BD.first + AC.second*BD.second;
  if(dotProduct <= 1e-9 && dotProduct >= -1e-9) return true;
  else return false;
}


double dist(const Vertex &a, const Vertex &b){
  return sqrt( ((b.first - a.first) * (b.first - a.first)) + ((b.second - a.second) * (b.second - a.second)));
}

bool operator==(const Vertex &a, const Vertex &b){
  return (a.first == b.first) && (a.second == b.second);
}


std::ostream& operator<<(std::ostream &o, const Vertex &p){
  o << "<" << p.first << ", " << p.second << ">";
  return o;
}


bool isNumber(const std::string& s){
  return !s.empty() && s.find_first_not_of("-.0123456789") == std::string::npos;
}


std::istream& operator>>(std::istream &is, Vertex &p){
  std::string checker;

  is >> checker;
  if(isNumber(checker) == false){
     throw std::overflow_error("Is not a number");
  }
  p.first = static_cast<double>(std::stod(checker));

  is >> checker;
  if(isNumber(checker) == false){
     throw std::overflow_error("Is not a number");
  }
  p.second = static_cast<double>(std::stod(checker));

  return is;
}




class Rectangle : public Figure {
  int Id;
  Vertex *vertices;
public:
  Rectangle() : Id{0}, vertices{new Vertex[4]} {
     for (int i = 0; i < 4; i++){
```

```
        vertices[i] = std::make_pair(0,0);
    }
}

Rectangle(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :
        Id{id}, vertices{new Vertex[4]} {
    if (a == b || a == c || b == c || a == d || oneline(a,b,c) || oneline(a,b,d) || oneline(b,c,d) ||
        !(perpendic(a, b, a, d)) || !collinear(a, d, c, b)
        || !collinear(a, b, d, c)) {
        throw std::logic_error("The entered coordinates of the vertices do not belong to the rectangle.");
    } else {
        vertices[0] = a;
        vertices[1] = b;
        vertices[2] = c;
        vertices[3] = d;
    }
}

~Rectangle() override {
    delete [] vertices;
    vertices = nullptr;
}

Vertex Center() const override {
    return Get_Center(vertices,4);
}

double Area() const override {
    auto AB = dist(vertices[0], vertices[1]);
    auto AD = dist(vertices[0],vertices[3]);
    return AD*AB;
}

std::ostream &Print(std::ostream &out) const override{
    out << "Id: " << Id << "\n";
    out << "Figure: Rectangle\n";
    out << "Coords:\n";
    for (int i = 0; i < 4; i++) {
        out << vertices[i] << "\n";
    }
    return out;
}

void Serialize(std::ofstream &os) const override{
    FigureType type = Rec;
    os.write((char *) &type, sizeof(type));
    os.write((char *) &Id, sizeof(Id));
    for (int i = 0; i < 4; i++) {
        os.write((char*) &(vertices[i].first), sizeof(vertices[i].first));
        os.write((char*) &(vertices[i].second), sizeof(vertices[i].second));
    }
}

void Deserialize(std::ifstream &is) override {
```

```
    is.read((char *) &Id, sizeof(Id));
    for (int i = 0; i<4; i++) {
      is.read((char *) &(vertices[i].first), sizeof(vertices[i].first));
      is.read((char *) &(vertices[i].second), sizeof(vertices[i].second));
    }
  }

  int getId() const override {
    return Id;
  }
};

class Trapezoid : public Figure {
  int Id;
  Vertex *vertices;
public:
  Trapezoid() : Id{0}, vertices{new Vertex[4]} {
    for (int i = 0; i < 4; i++){
      vertices[i] = std::make_pair(0,0);
    }
  }

  Trapezoid(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :
      Id{id}, vertices{new Vertex[4]} {
    auto AB = dist(a, b);
    auto AD = dist(a, d);
    if (a == b || a == c || b == c || a == d || b == d || c == d || oneline(a,b,c) ||
      oneline(a,b,d) || oneline(b,c,d) ||
      collinear(a, b, c, a) || collinear(a, b, d, a) || collinear(a, c, d, a)
      || collinear(b, c, d, b)) {
      throw std::logic_error("The entered coordinates of the vertices do not belong to the trapezoid.");
    } else {
      vertices[0] = a;
      vertices[1] = b;
      vertices[2] = c;
      vertices[3] = d;
    }
  }

  ~Trapezoid() override {
    delete [] vertices;
    vertices = nullptr;
  }

  Vertex Center() const override {
    return Get_Center(vertices,4);
  }

  double Area() const override {
    double x1,x2,x3,x4, y1,y2,y3,y4;
    x1 = vertices[0].first; y1 = vertices[0].second;
    x2 = vertices[1].first; y2 = vertices[1].second;
    x3 = vertices[2].first; y3 = vertices[2].second;
    x4 = vertices[3].first; y4 = vertices[3].second;
```

```
    auto area = ( (x1*y2-x2*y1)+(x2*y3-x3*y2)+(x3*y4-x4*y3) ) / 2;
    return std::abs(area);
  }

  std::ostream &Print(std::ostream &out) const override{
    out << "Id: " << Id << "\n";
    out << "Figure: Trapezoid\n";
    out << "Coords:\n";
    for (int i = 0; i < 4; i++) {
      out << vertices[i] << "\n";
    }
    return out;
  }

  void Serialize(std::ofstream &os) const override{
    FigureType type = Trapeze;
    os.write((char *) &type, sizeof(type));
    os.write((char *) &Id, sizeof(Id));
    for (int i = 0; i < 4; i++) {
      os.write((char *) &(vertices[i].first),sizeof(vertices[i].first));
      os.write((char *) &(vertices[i].second),sizeof(vertices[i].second));
    }
  }

  void Deserialize(std::ifstream &is) override {
    is.read((char *) &Id, sizeof(Id));
    for (int i = 0; i < 4; i++) {
      is.read((char *) &(vertices[i].first),sizeof(vertices[i].first));
      is.read((char *) &(vertices[i].second),sizeof(vertices[i].second));
    }
  }

  int getId() const override {
    return Id;
  }
};

class Rhombus: public Figure {
  int Id;
  Vertex *vertices;
public:
  Rhombus() : Id{0}, vertices{new Vertex[4]} {
    for (int i = 0; i < 4; i++){
      vertices[i] = std::make_pair(0,0);
    }
  }
  Rhombus(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :
      Id{id}, vertices{new Vertex[4]} {
    auto AB = dist(a, b);
    auto AD = dist(a, d);
    auto BC = dist(b,c);
    auto CD = dist(c,d);
    if (a == b || a == c || b == c || a == d || b == d || c == d ||
```

```
        !(AB == AD) || !(CD == BC) || !(AB == CD) ||
        oneline(a,b,c) || oneline(a,b,d) || oneline(b,c,d) ||
        !perpendic(a,b,c,d) || !collinear(a, b, c, d)
        || !collinear(a, d, c, b) ) {
        throw std::logic_error("The entered coordinates of the vertices do not belong to the rhombus.");
    } else {
        vertices[0] = a;
        vertices[1] = b;
        vertices[2] = c;
        vertices[3] = d;
    }
}

~Rhombus() override {
    delete [] vertices;
    vertices = nullptr;
}

Vertex Center() const override {
    return Get_Center(vertices,4);
}

double Area() const override {
    double AC = dist(vertices[0],vertices[2]);
    double DB = dist(vertices[3],vertices[1]);
    return (AC*DB) /2;
}

std::ostream &Print(std::ostream &out) const override{
    out << "Id: " << Id << "\n";
    out << "Figure: Trapezoid\n";
    out << "Coords:\n";
    for (int i = 0; i < 4; i++) {
        out << vertices[i] << "\n";
    }
    return out;
}

void Serialize(std::ofstream &os) const override{
    FigureType type = Rhomb;
    os.write((char *) &type, sizeof(type));
    os.write((char *) &Id, sizeof(Id));
    for (int i = 0; i < 4; i++) {
        os.write((char *) &(vertices[i].first),sizeof(vertices[i].first));
        os.write((char *) &(vertices[i].second),sizeof(vertices[i].second));
    }
}

void Deserialize(std::ifstream &is) override {
    is.read((char *) &Id, sizeof(Id));
    for (int i = 0; i < 4; i++) {
        is.read((char *) &(vertices[i].first),
                sizeof(vertices[i].first));
        is.read((char *) &(vertices[i].second),
```

```
            sizeof(vertices[i].second));
    }
  }

  int getId() const override {
    return Id;
  }
};


#endif //FIGURES_H
```

2. Ссылка на репозиторий на Github

https://github.com/kwk18/oop_exercise_08

3.Набор testcases

1. **Входные данные:**

```
./oop_exercise_08 3
Add
Rec  0 0  3 0  3 5  0 5
Trapeze 0 0  4 0  3 5  1 5
Rhomb -2 0  0 -5  2 0 0 5

Menu
Add
Rec 5 5  10 5  10 10  5 10
Rhomb 0 0  2 -3  4 0  2 3
Rec 0 0 3 0 3 2 0 2
Exit
ls
cat 0
cat 1
```

4. Результат выполнения тестов

1.

```
Menu
Add figure
Exit

Add
Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]
Rec
Enter the coordinates separated by a space
0 0 3 0 3 5 0 5
Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]
```

```
Trapeze
Enter the coordinates separated by a space
0 0  4 0  3 5  1 5
Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]
Rhomb
Enter the coordinates separated by a space
-2 0  0 -5  2 0  0 5
Id: 0
Figure: Rectangle
Coords:
<0, 0>
<3, 0>
<3, 5>
<0, 5>
Id: 1
Figure: Trapezoid
Coords:
<0, 0>
<4, 0>
<3, 5>
<1, 5>
Id: 2
Figure: Trapezoid
Coords:
<-2, 0>
<0, -5>
<2, 0>
<0, 5>
Menu

Menu
Add figure
Exit

Add
Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]
Rec
Enter the coordinates separated by a space
5 5 10 5  10 10 5 10
Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]
Rhomb
Enter the coordinates separated by a space
0 0  2 -3  4 0  2 3
Enter name of figure: Rectangle[Rec], Trapezoid[Trapeze], Rhombus[Rhomb]
Rec
Enter the coordinates separated by a space
0 0 3 0 3 2 0 2
Id: 0
Figure: Rectangle
Coords:
<5, 5>
<10, 5>
<10, 10>
<5, 10>
Id: 1
Figure: Trapezoid
Coords:
<0, 0>
```

<2, -3>
<4, 0>
<2, 3>
Id: 2
Figure: Rectangle
Coords:
<0, 0>
<3, 0>
<3, 2>
<0, 2>

Exit

5. Объяснение результатов программы

Программа состоит из 4 файлов:
1) figures.h - содержит реализацию фигур и все операции связанные с ними.
2) factory.h - содержит класс для создания графических примитиве фигур.
3) subscriber.h - реализация класса, необходимого для передачи в поток как функтора, который необходим для выполнения обработки на отдельном потоке.
4) main.cpp - файл с взаимодействием с пользователем.
Пользователь при запуске указывает размер буфера фигур, то есть количество фигур, которое можно вбить буфер. В программе пользователь может полностью заполнить буфер различными фигурами, после чего будет показано содержимое буфера на экран и экспорт буфера в файл с уникальным именем. После экспорта буфер очищается и пользователь может вновь его заполнить различными фигурами. В программе имеются два потока, в поток subscriber_thread передаем функтор класса Subscriber, который после заполнения буфера будет выводить информацию о фигурах из буфера в файл и на экран.

При неверном вводе параметров фигуры будет происходить исключения и пользователю нужно вновь выбрать фигуру с данными.

Вывод: Проделав данную работу я познакомилась с потоками и их блокировкой с помощью мьютекса. Научилась работать с многопоточностью в C++ с асинхронной обработкой данных.