**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 7**

Тема: Проектирование структуры классов

Студент: Ивенкова Любовь Васильевна

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

**Содержание**

[**1. Постановка задачи**](#_q3xiyr9tk7jn)3

[**2. Описание программы**](#_fsgxdsto112)3

[**3. Набор тестов**](#_2bvtr239grk9)6

[**4. Результат выполнения тестов**](#_ligwj0ite9tr)7

[**5. Листинг программы**](#_7vl5juvf0yhv)9

[**6. Вывод**](#_s13bec61cwdb)9

[**Список используемых источников**](#_y4bprx2szmjz)17

# 

# **1. Постановка задачи**

**Вариант:** 1

**Задача:**

Спроектировать простейший «графический» векторный редактор.

Требование к функционалу редактора:

* создание нового документа;
* импорт документа из файла;
* экспорт документа в файл;
* создание графического примитива (треугольника, квадрата и прямоугольника);
* удаление графического примитива;
* отображение документа на экране (печать перечня графических объектов и их характеристик в std::cout);
* реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

Требования к реализации:

* Создание графических примитивов необходимо вынести в отдельный класс – Factory.
* Сделать упор на использовании полиморфизма при работе с фигурами;
* Взаимодействие с пользователем (ввод команд) реализовать в функции main.

# **2. Описание программы**

Программа принимает в себя данные из консоли и из файла, при перенаправлении потока ввода вывода, и выполняет заданные действия. При запуске появляется меню выбора операций, после выбора которой вводятся данные. В список операций входит *добавление фигуры*, *удаление фигуры*, *вывод документа(вывод координат всех фигур), создание нового документа, загрузка и сохранение файлов*.

Программа состоит из четырех файлов: main.cpp, document.h, factory.h и figures.h:

* figures.h - описание классов фигур. В начале файла прописаны “id” фигур, которые будут использоваться при записи и считывании документов. Класс Figure является виртуальным родительским. От него наследуются 3 шаблонных класса фигур: класс треугольника, класс прямоугольника и класс квадрата. Публичные поля содержат функцию вывода координат фигур и сохранение фигуры в файл.
* factory.h - содержит класс Factory, который создает графические примитивы фигур с помощью функции CreateFigure().
* document.h - содержит класс Document, который работает с документом.

Класс TAction - абстрактный класс. Является родительским классом для AddAction и DeleteAction.

void Action() - удаление/добавление фигуры.

Также класс Document содержит стек действий, которые в дальнейшем будет использоваться для функции Undo, и список фигур, которые были созданы во время работы с документом.

void CreateNew() - создание нового документа.

void Save() - сохранение документа в файл.

void Load() - загрузка документа.

void AddElem() - добавление фигуры в список фигур и добавление действия в стек действий.

void Add() - добавление фигуры в список фигур без добавления действия в стек действий.

void Delete() - удаление фигуры из списка фигур без добавления действия в стек действий.

void DeleteElem() - удаление фигуры из списка фигур и добавление действия в стек действий.

void Undo() - отмена последнего действия.

* main.cpp - главный файл. В нем создается стек. Считывается выбор пользователя, исходя из которого выбираются дальнейшие действия.

void menu() - вывод меню.

void error() - вывод сообщения об ошибке.

**Переменные классов**

class Figure

нет переменных класса

class Triangle

using coords = std::pair<T, T>;

coords a, b, c - координаты треугольника;

size\_t side - сторона правильного треугольника;

class Rectangle

using coords = std::pair<T, T>;

coords a, b, c, d - координаты прямоугольника;

size\_t side, height - высота и ширина прямоугольника;

class Square

using coords = std::pair<T, T>;

coords a, b, c, d - координаты квадраты;

size\_t side - сторона квадрата;

class Factory

нет переменных

class Document

std::list<std::shared\_ptr<Figure>> figures - список всех фигур в документе;

std::stack<std::shared\_ptr<TAction>> actions - стек действий, производимых во время работы с документом;

class TAction

нет переменных

class AddAction

size\_t position - позиция элемента, на котором он стоял до удаления;

std::shared\_ptr<Figure> fig - указатель на объект, которых удалили;

class DeleteAction

size\_t position - позиция элемента, которых добавили.

# **3. Набор тестов**

Таблица 1. Тест 1

|  |  |
| --- | --- |
| 3  1 1 rec  6 7  3  1 1 tr 5  3  1 3 sq 7  3  4  3  0 | * вывод фигур в документе (ничего не выведет); * добавление прямоугольника на 1 позицию списка со сторонами 6 и 7; * вывод фигур в документе (выведет прямоугольник со сторонами 6 и 7); * добавление треугольника на 1 позицию списка со стороной 5; * вывод фигур в документе (выведет прямоугольник со сторонами 6 и 7 и треугольник со стороной 5); * добавление квадрата на 1 позицию списка со стороной 7; * вывод фигур в документе (выведет прямоугольник со сторонами 6 и 7, треугольник со стороной 5 и квадрат со стороной 7); * отмена последнего действия (добавление квадрата со стороной 7); * вывод фигур в документе (выведет прямоугольник со сторонами 6 и 7 и треугольник со стороной 5); * выход из программы; |

Таблица 2. Тест 2

|  |  |
| --- | --- |
| 4  7  fil  1 1 rectangle  7 4  3  6 file1  5  3  7 file1  3  0 | * отмена последнего действия (выведет ошибку: не было никаких действий до этого); * загрузка из файла “fil” (выведет ошибку: нет файла с таким именем); * добавление прямоугольника на 1 позицию списка со сторонами 7 и 4; * вывод фигур в документе (выведет прямоугольник со сторонами 7 и 4); * сохранение документа в файл “file1” * создание нового документа; * вывод фигур в документе (ничего не выведет); * загрузка из файла “file1” * вывод фигур в документе (выведет прямоугольник со сторонами 7 и 4); * выход из программы; |

# 

# **4. Результат выполнения тестов**

Тест 1:

Enter 0-7 to:

1 - add figure

2 - delete figure

3 - print file

4 - undo

5 - create new file

6 - save file

7 - load file

0 - exit

3

What's next?

>>1

Enter the position you want to place the new figure: 1

Enter name of figure: rec

Enter side: 6 7

Done!

What's next?

>>3

Rectangle: (0, 0), (6, 0), (6, 7), (0, 7)

What's next?

>>1

Enter the position you want to place the new figure: 1

Enter name of figure: tr

Enter side: 5

Done!

What's next?

>>3

Triangle: (0, 0), (5, 0), (2.5, 4.33013)

Rectangle: (0, 0), (6, 0), (6, 7), (0, 7)

What's next?

>>1

Enter the position you want to place the new figure: 3

Enter name of figure: sq

Enter side: 7

Done!

What's next?

>>3

Triangle: (0, 0), (5, 0), (2.5, 4.33013)

Rectangle: (0, 0), (6, 0), (6, 7), (0, 7)

Square: (0, 0), (7, 0), (7, 7), (0, 7)

What's next?

>>4

What's next?

>>3

Rectangle: (0, 0), (6, 0), (6, 7), (0, 7)

Square: (0, 0), (7, 0), (7, 7), (0, 7)

What's next?

>>0

The program is closed, goodbye!

Тест 2:

Enter 0-7 to:

1 - add figure

2 - delete figure

3 - print file

4 - undo

5 - create new file

6 - save file

7 - load file

0 - exit

4

There are no actions to cancel!

What's next?

>>7

Enter the name of file to load: fil

ERROR: can't open file

What's next?

>>1

Enter the position you want to place the new figure: 1

Enter name of figure: rectangle

Enter side: 7 4

Done!

What's next?

>>3

Rectangle: (0, 0), (7, 0), (7, 4), (0, 4)

What's next?

>>6

Enter the name of file to save: file1

Saved!

What's next?

>>5

What's next?

>>3

What's next?

>>7

Enter the name of file to load: file1

Loaded!

What's next?

>>3

Rectangle: (0, 0), (7, 0), (7, 4), (0, 4)

What's next?

>>0

The program is closed, goodbye!

# **5. Листинг программы**

main.cpp

|  |
| --- |
| /\* Ивенкова Любовь Васильевна, М8О-208Б-19  https://github.com/Li-Iven/OOP/tree/main/oop\_exercise\_07  \*/  #include<iostream>  #include"factory.h"  #include"document.h"  #include"figures.h"  #include<memory>  #include<algorithm>  void menu() {  std::cout << "Enter 0-7 to:" << std::endl;  std::cout << "1 - add figure" << std::endl;  std::cout << "2 - delete figure" << std::endl;  std::cout << "3 - print file" << std::endl;  std::cout << "4 - undo" << std::endl;  std::cout << "5 - create new file" << std::endl;  std::cout << "6 - save file" << std::endl;  std::cout << "7 - load file" << std::endl;  std::cout << "0 - exit" << std::endl;  }  int main()  {  std::ofstream fout;  std::ifstream fin;  Document doc;  int choice;  unsigned long side;  char r[30];  menu();  std::cout << "Your choice: ";  do {  std::cin >> choice;  switch (choice) {  case 1:  std::cout << "Enter the position you want to place the new figure: ";  std::cin >> side;  try {  if (side > doc.figures.size() + 1) {  throw side;  }  }  catch (const unsigned long a) {  std::cout << "There is no such position in the list!\n";  break;  }  doc.AddElem(side);  std::cout << "Done!\n";  break;  case 2:  std::cout << "Enter the position of the figure you want to delete: ";  try {  std::cin >> side;  if (side > doc.figures.size()) {  throw side;  }  }  catch (const unsigned long s) {  std::cout << "There is no such position in the list!\n";  break;  }  doc.DeleteElem(side);  std::cout << "Done!\n";  break;  case 3: {  auto print = [](std::shared\_ptr<Figure> elem) {  elem->Print();  };  std::for\_each(doc.figures.begin(), doc.figures.end(), print);  break;  }  case 4:  doc.Undo();  break;  case 5:  doc.CreateNew();  break;  case 6:  std::cout << "Enter the name of file to save: ";  std::cin >> r;  fout.open(r, std::ios::out);  doc.Save(fout);  std::cout << "Saved!\n";  fout.close();  break;  case 7:  std::cout << "Enter the name of file to load: ";  std::cin >> r;  fin.open(r, std::ios::in);  if (fin) {  doc.Load(fin);  std::cout << "Loaded!\n";  fin.close();  }  else {  std::cout << "ERROR: can't open file\n";  }  break;  case 0:  std::cout << "The program is closed, goodbye!\n";  return false;  break;  default:  std::cout << "Incorrect values! Try again.\n";  break;  }  std::cout << "What's next?\n";  std::cout << ">>";  } while (choice);  } |

factory.h

|  |
| --- |
| #pragma once  #include<memory>  #include"figures.h"  template<typename T>  class Factory {  public:  static std::shared\_ptr<Figure> CreateFigure(char\* type) {  std::shared\_ptr<Figure> fig;  T side;  if (type[0] == 's') {  std::cout << "Enter side: ";  std::cin >> side;  fig = std::make\_shared<Square<T>>(side);  }  if (type[0] == 't') {  std::cout << "Enter side: ";  std::cin >> side;  fig = std::make\_shared<Triangle<T>>(side);  }  if (type[0] == 'r') {  std::cout << "Enter side and height: ";  T height;  std::cin >> side >> height;  fig = std::make\_shared<Rectangle<T>>(side, height);  }  return fig;  }  }; |

figures.h

#pragma once

#include<iostream>

#include<fstream>

#include<math.h>

const int ID\_Square = 1;

const int ID\_Triangle = 2;

const int ID\_Rectangle = 3;

class Figure {

public:

virtual void Print() = 0;

virtual void Save(std::ostream& file) = 0;

virtual ~Figure() {}

};

template<typename T>

class Square : public Figure {

private:

using coords = std::pair<T, T>;

coords a, b, c, d;

size\_t side;

public:

Square(T s) :side(s) {

b.first = c.second = c.first = d.second = s;

a.first = a.second = b.second = d.first = 0;

}

Square() {}

~Square() {

side = 0;

}

void Print() override {

std::cout << "Square: ";

std::cout << "(" << a.first << ", " << a.second << "), ";

std::cout << "(" << b.first << ", " << b.second << "), ";

std::cout << "(" << c.first << ", " << c.second << "), ";

std::cout << "(" << d.first << ", " << d.second << ")" << std::endl;

}

void Save(std::ostream& file) override {

file.write((char\*)&ID\_Square, sizeof(int));

file.write((char\*)&side, sizeof(size\_t));

}

};

template<typename T>

class Triangle : public Figure {

private:

using coords = std::pair<T, T>;

coords a, b, c;

size\_t side;

public:

Triangle(size\_t s) : side(s) {

a.first = a.second = b.second = 0;

b.first = s;

c.second = s \* sqrt(3) / 2;

c.first = (double)s / 2;

}

Triangle() {}

~Triangle() {

side = 0;

}

void Print() override {

std::cout << "Triangle: ";

std::cout << "(" << a.first << ", " << a.second << "), ";

std::cout << "(" << b.first << ", " << b.second << "), ";

std::cout << "(" << c.first << ", " << c.second << ")" << std::endl;

}

void Save(std::ostream& file) override {

file.write((char\*)&ID\_Triangle, sizeof(int));

file.write((char\*)&side, sizeof(size\_t));

}

};

template<typename T>

class Rectangle : public Figure {

private:

using coords = std::pair<T, T>;

coords a, b, c, d;

size\_t side, height;

public:

Rectangle(T s, T h) :side(s), height(h) {

a.first = a.second = b.second = d.first = 0;

b.first = c.first = s;

c.second = d.second = h;

}

Rectangle() {}

~Rectangle() {

side = 0;

}

void Print() override {

std::cout << "Rectangle: ";

std::cout << "(" << a.first << ", " << a.second << "), ";

std::cout << "(" << b.first << ", " << b.second << "), ";

std::cout << "(" << c.first << ", " << c.second << "), ";

std::cout << "(" << d.first << ", " << d.second << ")" << std::endl;

}

void Save(std::ostream& file) override {

file.write((char\*)&ID\_Rectangle, sizeof(int));

file.write((char\*)&side, sizeof(size\_t));

file.write((char\*)&height, sizeof(size\_t));

}

};

document.h

#pragma once

#include<fstream>

#include"factory.h"

#include<list>

#include<stack>

class Document {

class TAction {

public:

virtual void Action(Document& doc) = 0;

virtual ~TAction() {}

};

class AddAction : public TAction {

private:

size\_t position;

std::shared\_ptr<Figure> fig;

public:

AddAction(size\_t n, std::shared\_ptr<Figure> f) : position(n), fig(f) {}

void Action(Document& doc) override {

doc.Add(position, fig);

doc.actions.pop(); //??????

}

};

class DeleteAction : public TAction {

private:

size\_t position;

public:

DeleteAction(size\_t n) : position(n) {}

void Action(Document& doc) override {

doc.Delete(position);

}

};

public:

void CreateNew() {

figures.clear();

while (!actions.empty()) {

actions.pop();

}

}

void Save(std::ostream& file) {

if (figures.size() == 0) {

std::cout << "There are no figures to save!\n";

}

size\_t size = figures.size();

file.write((char\*)&size, sizeof(size\_t));

for (auto i = figures.begin(); i != figures.end(); ++i) {

(\*i)->Save(file);

}

}

void Load(std::istream& file) {

CreateNew();

size\_t size, id, side, height;

file.read((char\*)&size, sizeof(size\_t));

for (unsigned long i = 0; i < size; ++i) {

file.read((char\*)&id, sizeof(int));

if (id == 1) {

file.read((char\*)&side, sizeof(size\_t));

figures.push\_back(std::shared\_ptr<Figure>(new Square<double>(side)));

}

if (id == 2) {

file.read((char\*)&side, sizeof(size\_t));

figures.push\_back(std::shared\_ptr<Figure>(new Triangle<double>(side)));

}

if (id == 3) {

file.read((char\*)&side, sizeof(size\_t));

file.read((char\*)&height, sizeof(size\_t));

figures.push\_back(std::shared\_ptr<Figure>(new Rectangle<double>(side, height)));

}

}

}

void AddElem(size\_t pos) {

char r[10];

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

std::cin >> r;

Add(pos, Factory<double>::CreateFigure(r));

TAction\* act = new DeleteAction(pos);

actions.push(std::shared\_ptr<TAction>(act));

}

void Add(size\_t pos, std::shared\_ptr<Figure> fig) {

if (fig == nullptr) {

return;

}

if (pos == figures.size() + 1) {

figures.push\_back(fig);

}

else {

auto it = figures.begin();

for (size\_t i = 1; i < pos; ++i) {

++it;

}

figures.insert(it, fig);

}

}

void Delete(size\_t pos) {

if (pos == figures.size()) {

figures.pop\_back();

}

else {

auto it = figures.begin();

for (size\_t i = 1; i < pos; ++i) {

++it;

}

figures.erase(it);

}

}

void DeleteElem(size\_t pos) {

if (pos == figures.size()) {

TAction\* act = new AddAction(pos, figures.back());

actions.push(std::shared\_ptr<TAction>(act));

figures.pop\_back();

}

else {

auto it = figures.begin();

for (size\_t i = 1; i < pos; ++i) {

++it;

}

TAction\* act = new AddAction(pos, \*it);

actions.push(std::shared\_ptr<TAction>(act));

figures.erase(it);

}

}

void Undo() {

if (actions.empty()) {

std::cout << "There are no actions to cancel!\n";

}

else {

actions.top()->Action(\*this);

actions.pop();

}

}

std::list<std::shared\_ptr<Figure>> figures;

std::stack<std::shared\_ptr<TAction>> actions;

};

# **6. Вывод**

Благодаря данной лабораторной работе я научилась работать с файлами. Смогла реализовать функцию отмены действия Undo и закрепила знания в работе с полиморфизмом.

# **Список используемых источников**

1. Реализация Undo/Redo модели для сложного документа [Электронный ресурс]. URL: <https://habr.com/ru/post/313654/> (Дата обращения: 17.104.2021)
2. Undo и Redo - анализ и реализации [Электронный ресурс]. URL: <https://habr.com/ru/post/306398/> (Дата обращения: 17.104.2021)
3. Виртуальные функции и полиморфизм [Электронный ресурс]. URL: <https://ravesli.com/urok-163-virtualnye-funktsii-i-polimorfizm/> (Дата обращения: 17.104.2021)
4. std::size\_t [Электронный ресурс]. <https://en.cppreference.com/w/cpp/types/size_t> (Дата обращения: 17.104.2021)