**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: Гребенков Дмитрий Игоревич

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Спроектировать простейший «графический» векторный редактор.

Требование к функционалу редактора:

* создание нового документа
* импорт документа из файла
* экспорт документа в файл
* создание графического примитива (согласно варианту задания)
* 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;

Вариант задания:

Трапеция, ромб, 5-угольник.

1. Описание программы

Создан класс Figure, содержащий только виртуальные методы для вывода типа фигуры, вывода списка вершин фигуры и записи координат вершин фигуры в файл.

Созданы шаблонные классы для хранения трапеций, ромбов и пятиугольников. Они наследуются от класса Figure и содержат методы, раскрывающие методы класса Figure для каждой из заданных фигур.

Реализован требуемый по заданию класс Factory. Он содержит метод для создания фигур разных классов и метод для записи данных фигур в файл.

Также создан вспомогательный класс Action для хранения действий со списком.

Создан класс Redactor. Он хранит список различных фигур и стек действий, который необходим для реализации метода undo.

В классе Redactor находятся все необходимые по варианту задания методы для работы со списком: создание нового списка, вставка по индексу, удаление по индексу, печать списка, сохранение документа, удаление документа, отмена последнего действия.

Взаимодействие с пользователем осуществляется при помощи меню.

При выполнении некорректных операций происходит генерация исключений и вывод информации об ошибке в поток вывода std::cerr.

Примечание: Трапеция равнобедренная. Для ускорения вычислений и упрощения хранения на вводимые фигуры установлены некоторые ограничения. Во-первых, основания вводимых трапеций должны быть параллельны оси Ox. Во-вторых, большее основание трапеции должно быть снизу. В-третьих, диагонали вводимых ромбов должны быть параллельны осям Ox и Oy.

Трапеция задается координатами трех вершин.

Ромб задается координатами двух вершин.

Пятиугольник задается центром описанной окружности и радиусом.

1. Наборы тестов и результаты их выполнения

Демонстрация работы основных функций программы:

0. Show menu.

1. New document.

2. Add figure.

3. Delete figure.

4. Print list.

5. Save document.

6. Load document.

7. Undo.

8. Quit program.

Enter option: 2

Enter position: 0

1. Trapezoid.

2. Rhombus.

3. Pentagon.

Enter type of the figure: 2

Enter coordinates of left and bottom vertices:

0 1 1 0

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

Enter option: 2

Enter position: 1

1. Trapezoid.

2. Rhombus.

3. Pentagon.

Enter type of the figure: 1

Enter coordinates of left-top, left-bottom and right-bottom vertices:

1 2 0 0 4 0

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

Enter option: 2

Enter position: 1

1. Trapezoid.

2. Rhombus.

3. Pentagon.

Enter type of the figure: 3

Enter coordinates of center and radius:

1 1 1

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(0.0489438, 1.30902)(1, 2)(1.95106, 1.30902)(1.58778, 0.190983)(0.412215, 0.190983)

(1, 2)(0, 0)(4, 0)(3, 2)

Enter option: 7

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

Enter option: 2

Enter position: 2

1. Trapezoid.

2. Rhombus.

3. Pentagon.

Enter type of the figure: 3

Enter coordinates of center and radius:

2 2 1

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

(1.04894, 2.30902)(2, 3)(2.95106, 2.30902)(2.58778, 1.19098)(1.41222, 1.19098)

Enter option: 2

Enter position: 3

1. Trapezoid.

2. Rhombus.

3. Pentagon.

Enter type of the figure: 2

Enter coordinates of left and bottom vertices:

0 1 2 0

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

(1.04894, 2.30902)(2, 3)(2.95106, 2.30902)(2.58778, 1.19098)(1.41222, 1.19098)

(0, 1)(2, 0)(4, 1)(2, 2)

Enter option: 3

Enter position: 2

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

(0, 1)(2, 0)(4, 1)(2, 2)

Enter option: 5

Enter name of file: v.bin

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

(0, 1)(2, 0)(4, 1)(2, 2)

Enter option: 7

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

(1.04894, 2.30902)(2, 3)(2.95106, 2.30902)(2.58778, 1.19098)(1.41222, 1.19098)

(0, 1)(2, 0)(4, 1)(2, 2)

Enter option: 7

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

(1.04894, 2.30902)(2, 3)(2.95106, 2.30902)(2.58778, 1.19098)(1.41222, 1.19098)

Enter option: 7

Enter option: 7

Enter option: 7

Enter option: 7

No actions to undo.

Enter option: 4

List is empty.

Enter option: 6

Enter name of file: v.bin

Enter option: 4

(0, 1)(1, 0)(2, 1)(1, 2)

(1, 2)(0, 0)(4, 0)(3, 2)

(0, 1)(2, 0)(4, 1)(2, 2)

Enter option: 7

No actions to undo.

Enter option: 1

Enter option: 4

List is empty.

Enter option: 0

0. Show menu.

1. New document.

2. Add figure.

3. Delete figure.

4. Print list.

5. Save document.

6. Load document.

7. Undo.

8. Quit program.

Enter option: 8

Работа с некорректными командами:

0. Show menu.

1. New document.

2. Add figure.

3. Delete figure.

4. Print list.

5. Save document.

6. Load document.

7. Undo.

8. Quit program.

Enter option: 2

Enter position: 0

1. Trapezoid.

2. Rhombus.

3. Pentagon.

Enter type of the figure: 2

Enter coordinates of left and bottom vertices:

3 3 3 3

Enter option: 2

Enter position: -1

ERROR: incorrect index.

Enter option: 2

Enter position: 7

ERROR: incorrect index.

Enter option: 3

Enter position: -1

ERROR: incorrect index.

Enter option: 3

Enter position: 5

ERROR: incorrect index.

Enter option: 4

(3, 3)(3, 3)(3, 3)(3, 3)

Enter option: 7

Enter option: 4

List is empty.

Enter option: 7

No actions to undo.

Enter option: 2

Enter position: 0

1. Trapezoid.

2. Rhombus.

3. Pentagon.

Enter type of the figure: 4

ERROR: incorrect index.

Enter option: 4

List is empty.

Enter option: 2

Enter position: 0

1. Trapezoid.

2. Rhombus.

3. Pentagon.

Enter type of the figure: 2

Enter coordinates of left and bottom vertices:

4 4 4 4

Enter option: 5

Enter name of file: v.bin

Enter option: 4

(4, 4)(4, 4)(4, 4)(4, 4)

Enter option: 1

Enter option: 4

List is empty.

Enter option: 7

No actions to undo.

Enter option: 6

Enter name of file: v.bin

Enter option: 4

(4, 4)(4, 4)(4, 4)(4, 4)

Enter option: 7

No actions to undo.

Enter option: 3

Enter position: 1

ERROR: incorrect index.

Enter option: 3

Enter position: 0

Enter option: 4

List is empty.

Enter option: 7

Enter option: 4

(4, 4)(4, 4)(4, 4)(4, 4)

Enter option: 6

Enter name of file: v.bin

Enter option: 4

(4, 4)(4, 4)(4, 4)(4, 4)

Enter option: 4

(4, 4)(4, 4)(4, 4)(4, 4)

Enter option: 8

1. Листинг программы

/\*

Гребенков Д.И.

Группа М8О-207Б-19.

Вариант задания: трапеция, ромб, 5-угольник.

Спроектировать простейший <<графический>> векторный редактор.

Требование к функционалу редактора:

создание нового документа

импорт документа из файла

экспорт документа в файл

создание графического примитива (согласно варианту задания)

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;

\*/

#include <iostream>

#include <cmath>

#include <memory>

#include <list>

#include <stack>

#include <iterator>

#include <algorithm>

#include <string>

const double pi = 3.14159;

template <typename T>

class Figure {

public:

virtual int type () = 0;

virtual void vertices () = 0;

virtual void writeInFile (FILE \* fd) = 0;

virtual ~Figure () {};

};

template <typename T>

class Trapezoid: public Figure <T> {

public:

std::pair <T, T> left\_top, left\_bottom, right\_bottom;

Trapezoid () {}

Trapezoid (T ltx, T lty, T lbx, T lby, T rbx, T rby) {

left\_top.first = ltx;

left\_top.second = lty;

left\_bottom.first = lbx;

left\_bottom.second = lby;

right\_bottom.first = rbx;

right\_bottom.second = rby;

}

virtual int type () {

return 1;

}

virtual void vertices () {

std::cout << "(" << left\_top.first << ", " << left\_top.second << ")"

<< "(" << left\_bottom.first << ", " << left\_bottom.second << ")"

<< "(" << right\_bottom.first << ", " << right\_bottom.second << ")"

<< "(" << right\_bottom.first - left\_top.first + left\_bottom.first

<< ", " << left\_top.second << ")" << std::endl;

}

virtual void writeInFile (FILE \* fd) {

int type = this->type ();

fwrite (&type, sizeof (int), 1, fd);

fwrite (&left\_top.first, sizeof (T), 1, fd);

fwrite (&left\_top.second, sizeof (T), 1, fd);

fwrite (&left\_bottom.first, sizeof (T), 1, fd);

fwrite (&left\_bottom.second, sizeof (T), 1, fd);

fwrite (&right\_bottom.first, sizeof (T), 1, fd);

fwrite (&right\_bottom.second, sizeof (T), 1, fd);

}

};

template <typename T>

class Rhombus: public Figure <T> {

public:

std::pair <T, T> left, bottom;

Rhombus () {}

Rhombus (T lx, T ly, T bx, T by) {

left.first = lx;

left.second = ly;

bottom.first = bx;

bottom.second = by;

}

virtual int type () {

return 2;

}

virtual void vertices () {

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

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

std::cout << "(" << 2 \* bottom.first - left.first << ", " << left.second << ")";

std::cout << "(" << bottom.first << ", " << 2 \* left.second - bottom.second << ")"

<< std::endl;

}

virtual void writeInFile (FILE \* fd) {

int type = this->type ();

fwrite (&type, sizeof (int), 1, fd);

fwrite (&left.first, sizeof (T), 1, fd);

fwrite (&left.second, sizeof (T), 1, fd);

fwrite (&bottom.first, sizeof (T), 1, fd);

fwrite (&bottom.second, sizeof (T), 1, fd);

}

};

template <typename T>

class Pentagon: public Figure <T> {

public:

std::pair <T, T> center;

T radius;

Pentagon () {}

Pentagon (T cx, T cy, T r) {

center.first = cx;

center.second = cy;

radius = r;

}

virtual int type () {

return 3;

}

virtual void vertices () {

std::cout << "(" << center.first - radius \* sin (72 \* pi / 180) << ", " <<

center.second + radius \* cos (72 \* pi / 180) << ")";

std::cout << "(" << center.first << ", " << center.second + radius << ")";

std::cout << "(" << center.first + radius \* sin (72 \* pi / 180) << ", " <<

center.second + radius \* cos (72 \* pi / 180) << ")";

std::cout << "(" << center.first + radius \* sin (36 \* pi / 180) << ", " <<

center.second - radius \* cos (36 \* pi / 180) << ")";

std::cout << "(" << center.first - radius \* sin (36 \* pi / 180) << ", " <<

center.second - radius \* cos (36 \* pi / 180) << ")" << std::endl;

}

virtual void writeInFile (FILE \* fd) {

int type = this->type ();

fwrite (&type, sizeof (int), 1, fd);

fwrite (&center.first, sizeof (T), 1, fd);

fwrite (&center.second, sizeof (T), 1, fd);

fwrite (&radius, sizeof (T), 1, fd);

}

};

template <typename T>

class Factory {

public:

static std::shared\_ptr <Figure <T> > create () {

int type = 0;

std::cout << "1. Trapezoid." << std::endl;

std::cout << "2. Rhombus." << std::endl;

std::cout << "3. Pentagon." << std::endl;

std::cout << "Enter type of the figure: ";

std::cin >> type;

if ((type > 0) && (type < 4)) {

if (type == 1) {

T ltx, lty, lbx, lby, rbx, rby;

std::cout << "Enter coordinates of left-top, left-bottom and right-bottom vertices:" << std::endl;

std::cin >> ltx >> lty >> lbx >> lby >> rbx >> rby;

Trapezoid <T> \* temp (new Trapezoid <T> (ltx, lty, lbx, lby, rbx, rby));

return std::shared\_ptr <Figure <T> > (temp);

}

else if (type == 2) {

T lx, ly, bx, by;

std::cout << "Enter coordinates of left and bottom vertices: " <<

std::endl;

std::cin >> lx >> ly >> bx >> by;

Rhombus <T> \* temp (new Rhombus <T> (lx, ly, bx, by));

return std::shared\_ptr <Figure <T> > (temp);

}

else if (type == 3) {

T r, cx, cy;

std::cout << "Enter coordinates of center and radius: " <<

std::endl;

std::cin >> cx >> cy >> r;

Pentagon <T> \* temp (new Pentagon <T> (cx, cy, r));

return std::shared\_ptr <Figure <T> > (temp);

}

}

else {

throw "ERROR: incorrect index.";

}

std::shared\_ptr <Figure <T> > temp;

return (temp);

}

static std::shared\_ptr <Figure <T> > read (FILE \* fd) {

int type = 0;

fread (&type, sizeof (int), 1, fd);

if ((type > 0) && (type < 4)) {

if (type == 1) {

T ltx, lty, lbx, lby, rbx, rby;

fread (&ltx, sizeof (T), 1, fd);

fread (&lty, sizeof (T), 1, fd);

fread (&lbx, sizeof (T), 1, fd);

fread (&lby, sizeof (T), 1, fd);

fread (&rbx, sizeof (T), 1, fd);

fread (&rby, sizeof (T), 1, fd);

Trapezoid <T> \* temp (new Trapezoid <T> (ltx, lty, lbx, lby, rbx, rby));

return std::shared\_ptr <Figure <T> > (temp);

}

else if (type == 2) {

T lx, ly, bx, by;

fread (&lx, sizeof (T), 1, fd);

fread (&ly, sizeof (T), 1, fd);

fread (&bx, sizeof (T), 1, fd);

fread (&by, sizeof (T), 1, fd);

Rhombus <T> \* temp (new Rhombus <T> (lx, ly, bx, by));

return std::shared\_ptr <Figure <T> > (temp);

}

else if (type == 3) {

T r, cx, cy;

fread (&cx, sizeof (T), 1, fd);

fread (&cy, sizeof (T), 1, fd);

fread (&r, sizeof (T), 1, fd);

Pentagon <T> \* temp (new Pentagon <T> (cx, cy, r));

return std::shared\_ptr <Figure <T> > (temp);

}

}

else {

throw "ERROR: incorrect type of the figure in the file.";

}

std::shared\_ptr <Figure <T> > temp;

return (temp);

}

};

template <typename T>

class Action {

public:

bool act;

int pos;

std::shared\_ptr <Figure <T> > figure;

};

template <typename T>

class Redactor {

public:

std::list <std::shared\_ptr <Figure <T> > > listF;

std::stack <Action <T> > stackA;

void newDocument () {

listF.erase (listF.begin (), listF.end ());

while (!stackA.empty ()) {

stackA.pop ();

}

}

void insert () {

int p = -1;

std::cout << "Enter position: ";

std::cin >> p;

if ((p > -1) && (p < listF.size () + 1)) {

std::shared\_ptr <Figure <T> > temp = Factory <T> :: create ();

std::list <std::shared\_ptr <Figure <int> > > :: iterator iter = listF.begin ();

for (int i = 0; i < p; i ++) {

++ iter;

}

listF.insert (iter, temp);

Action <T> a;

a.act = false;

a.pos = p;

stackA.push (a);

}

else if (listF.size () == 0) {

std::shared\_ptr <Figure <T> > temp = Factory <T> :: create ();

listF.push\_back (temp);

Action <T> a;

a.act = false;

a.pos = 0;

stackA.push (a);

}

else {

throw "ERROR: incorrect index.";

}

}

void erase () {

int p = -1;

std::cout << "Enter position: ";

std::cin >> p;

if ((p > -1) && (p < listF.size ())) {

std::list <std::shared\_ptr <Figure <int> > > :: iterator iter = listF.begin ();

for (int i = 0; i < p; i ++) {

++ iter;

}

Action <T> a;

a.act = true;

a.pos = p;

a.figure = \*(iter);

stackA.push (a);

listF.erase (iter);

}

else {

throw "ERROR: incorrect index.";

}

}

void print () {

if (listF.size () != 0) {

std::list <std::shared\_ptr <Figure <int> > > :: iterator iter = listF.begin ();

for (int i = 0; i < listF.size (); i ++) {

(\*iter)->vertices ();

++ iter;

}

}

else {

std::cout << " List is empty." << std::endl;

}

}

void saveDocument () {

std::string name;

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

std::cin >> name;

FILE \* fd = fopen (name.c\_str (), "w");

int count = listF.size ();

fwrite (&count, sizeof (int), 1, fd);

if (listF.size () != 0) {

std::list <std::shared\_ptr <Figure <int> > > :: iterator iter = listF.begin ();

for (int i = 0; i < listF.size (); i ++) {

(\*iter)->writeInFile (fd);

++ iter;

}

}

fclose (fd);

}

void loadDocument () {

std::string name;

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

std::cin >> name;

FILE \* fd = fopen (name.c\_str (), "r");

if (fd == NULL) {

throw "ERROR: file with this name does not exist.";

}

newDocument ();

int count = -1;

fread (&count, sizeof (int), 1, fd);

if (count > 0) {

for (int i = 0; i < count; i ++) {

listF.push\_back (Factory <T> :: read (fd));

}

}

else if (count < 0) {

throw "ERROR: wrong file format.";

}

}

void undo () {

if (!stackA.empty ()) {

Action <T> a = stackA.top ();

if (a.act) {

if ((a.pos > -1) && (a.pos < listF.size () + 1)) {

std::list <std::shared\_ptr <Figure <int> > > :: iterator iter = listF.begin ();

for (int i = 0; i < a.pos; i ++) {

++ iter;

}

listF.insert (iter, a.figure);

stackA.pop ();

}

else if (listF.size () == 0) {

listF.push\_back (a.figure);

stackA.pop ();

}

}

else {

if ((a.pos > -1) && (a.pos < listF.size ())) {

std::list <std::shared\_ptr <Figure <int> > > :: iterator iter = listF.begin ();

for (int i = 0; i < a.pos; i ++) {

++ iter;

}

listF.erase (iter);

stackA.pop ();

}

}

}

else {

std::cout << " No actions to undo." << std::endl;

}

}

};

void showMenu () {

std::cout << "0. Show menu." << std::endl;

std::cout << "1. New document." << std::endl;

std::cout << "2. Add figure." << std::endl;

std::cout << "3. Delete figure." << std::endl;

std::cout << "4. Print list." << std::endl;

std::cout << "5. Save document." << std::endl;

std::cout << "6. Load document." << std::endl;

std::cout << "7. Undo." << std::endl;

std::cout << "8. Quit program." << std::endl;

}

int main () {

Redactor <int> red;

int option = 0;

showMenu ();

while (option != 8) {

std::cout << "Enter option: ";

std::cin >> option;

switch (option) {

case 0: {

showMenu ();

break;

}

case 1: {

try {

red.newDocument ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 2: {

try {

red.insert ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 3: {

try {

red.erase ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 4: {

try {

red.print ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 5: {

try {

red.saveDocument ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 6: {

try {

red.loadDocument ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 7: {

try {

red.undo ();

}

catch (const char\* exc) {

std::cerr << exc << std::endl;

}

break;

}

case 8: {

break;

}

default: {

std::cerr << "ERROR: incorrect option." << std::endl;

break;

}

}

}

return 0;

}

1. Вывод

Приобретены практические навыки проектирования структуры классов приложения.

Список литературы

1. Виртуальные функции и полиморфизм в C++ | Уроки C++ - Ravesli [Электронный ресурс] URL: <https://ravesli.com/urok-163-virtualnye-funktsii-i-polimorfizm/> (дата обращения 17.12.2020).
2. Абстрактные классы и интерфейсы в C++ | Уроки C++ - Ravesli [Электронный ресурс] URL: <https://ravesli.com/urok-168-chistye-virtualnye-funktsii-interfejsy-i-abstraktnye-klassy/> (дата обращения 17.12.2020).
3. std::list - cppreference.com [Электронный ресурс] URL: <https://en.cppreference.com/w/cpp/container/list> (дата обращения 17.12.2020).
4. std::stack - cppreference.com [Электронный ресурс] URL: <https://en.cppreference.com/w/cpp/container/stack> (дата обращения 17.12.2020).