**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 7**

Тема: Проектирование структуры классов

Студент: Трофимов М.А.

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:\_\_\_\_\_\_\_\_\_

Оценка:\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_

Москва, 2019

1. **Постановка задачи**

Написать простейший редактор фигур для квадратов прямоугольников и трапеций с возможность сохранения и загрузки фигур, а также с операциями undo и redo.

1. **Репозиторий github**

<https://github.com/student31415/oop_exercise_07>

1. **Описание программы**

Программа принимает на вход команды на вставку и удаление фигур. Фигуры : квадрат, прямоугольник и трапеция задаются по двум, трём и четырём точкам соответственно. команды undo и redo откатывают изменения на вставку и удаление фигур. Также предусмотрена возможность сохранения в файл и загрузки из него. (файл бинарный)

1. **Набор testcases**

**test\_01.txt**

5

0

1

0 0 2 2

5

1

2

0 0 0 4 5 4

5

2

3

0 -5 0 -3 3 0 5 0

9 -1 2

7 7 7

9 -1 2

8 8 8

9 -1 2

0

действия теста:

вставка квадрата треугольника трапеции

печать их

откат этих действий с помощью undo

попытка распечатать пустой буфер

возврат фигур и их распечатка на экран

1. **Результаты выполнения тестов**

schizo@VivoBook:~/labs/2nd\_kurs/OOP/oop\_exercise\_07$ ./oop\_exercise\_07

1. Print menu

2. Save document

3. Load documet

4. New document (clear current document)

5. Insert figure

6. Remove figure

7. Undo

8. Redo

9. Do action with figures

0. Exit

>>5

input position for inserting figure

>>0

Chose figure for insert:

1. Square (2 vertex)

2. Rectangle (3 vertex)

3. Trapeze (4 vertex)

>>1

0 0 2 2

>>5

input position for inserting figure

>>1

Chose figure for insert:

1. Square (2 vertex)

2. Rectangle (3 vertex)

3. Trapeze (4 vertex)

>>2

0 0 0 4 5 4

>>5

input position for inserting figure

>>2

Chose figure for insert:

1. Square (2 vertex)

2. Rectangle (3 vertex)

3. Trapeze (4 vertex)

>>3

0 -5 0 -3 3 0 5 0

>>9

Choose input figure number for action or -1 for all figure

>>-1

Choose action:

0. print all squares

1. print all centers

2. print full information about figure

>>2

0. Figure - Square

vertexes:

( 0 , 0 )

( 0 , 2 )

( 2 , 2 )

( 2 , 0 )

square = 4

center = ( 1 , 1 )

1. Figure - Rectangle

vertexes:

( 0 , 0 )

( 0 , 4 )

( 5 , 4 )

( 5 , 0 )

square = 20

center = ( 2 , 2 )

2. Figure - Trapeze

vertexes:

( 0 , -5 )

( 0 , -3 )

( 3 , 0 )

( 5 , 0 )

square = 8

center = ( 2 , -2 )

>>7

Removed figure from pos 2

>>>>7

Removed figure from pos 1

>>>>7

Removed figure from pos 0

>>>>9

Choose input figure number for action or -1 for all figure

>>-1

Choose action:

0. print all squares

1. print all centers

2. print full information about figure

>>2

logic error: buffer of figures empty

>>8

Insert figure in pos 0

>>>>8

Insert figure in pos 1

>>>>8

Insert figure in pos 2

>>>>9

Choose input figure number for action or -1 for all figure

>>-1

Choose action:

0. print all squares

1. print all centers

2. print full information about figure

>>2

0. Figure - Square

vertexes:

( 0 , 0 )

( 0 , 2 )

( 2 , 2 )

( 2 , 0 )

square = 4

center = ( 1 , 1 )

1. Figure - Rectangle

vertexes:

( 0 , 0 )

( 0 , 4 )

( 5 , 4 )

( 5 , 0 )

square = 20

center = ( 2 , 2 )

2. Figure - Trapeze

vertexes:

( 0 , -5 )

( 0 , -3 )

( 3 , 0 )

( 5 , 0 )

square = 8

center = ( 2 , -2 )

>>0

1. **Листинг программы**

**main.cpp**

#include<iostream>

#include<vector>

#include "figures"

#include "factory"

#include "redactor"

/\*

Трофимов М.А

вариант 10

редактор фигур:

квадрат прямоугольник трапеция

\*/

void print\_menu(){

std::cout

<<"1. Print menu"<<std::endl

<<"2. Save document"<<std::endl

<<"3. Load documet"<<std::endl

<<"4. New document (clear current document)"<<std::endl

<<"5. Insert figure"<<std::endl

<<"6. Remove figure"<<std::endl

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

<<"8. Redo"<<std::endl

<<"9. Do action with figures"<<std::endl

<<"0. Exit"<<std::endl;

}

bool is\_num( std::string& str){

for(char c: str)

if(!( ((c>='0') && (c<='9')) || (c=='-') || (c=='.') ) )

return false;

return true;

}

typedef int work\_type;

work\_type get\_num(){

std::string buf;

std::cin>>buf;

if(is\_num(buf))

return atoi(buf.c\_str());

else

throw std::logic\_error("wrong input");

}

int main(){

print\_menu();

Redactor<work\_type> redactor;

while(true){

try{

std::cout<<">>"<<std::flush;

switch(get\_num()){

case 0:

return 0;

case 1:

print\_menu();

break;

case 2:{

std::string file\_path;

std::cout<<"Input file name/path"<<std::endl<<">>"<<std::flush;

std::cin>>file\_path;

redactor.save(file\_path);

break;

}

case 3:{

std::string file\_path;

std::cout<<"Input file name/path"<<std::endl<<">>"<<std::flush;

std::cin>>file\_path;

redactor.load(file\_path);

break;

}

case 4:{

redactor.New();

break;

}

case 5:{

std::cout<<"input position for inserting figure"<<std::endl<<">>"<<std::flush;

unsigned long long pos=get\_num();

std::cout

<<"Chose figure for insert: "<<std::endl

<<"1. Square (2 vertex)"<<std::endl

<<"2. Rectangle (3 vertex)"<<std::endl

<<"3. Trapeze (4 vertex)"<<std::endl<<">>"<<std::flush;

switch(get\_num()){

case 1:

redactor.insert(pos,

std::vector<std::pair<work\_type, work\_type>>{

std::pair<work\_type, work\_type>{get\_num(), get\_num()},

std::pair<work\_type, work\_type>{get\_num(), get\_num()}

});

break;

case 2:

redactor.insert( pos,

std::vector< std::pair<work\_type, work\_type> > {

std::pair<work\_type, work\_type>{get\_num(), get\_num()},

std::pair<work\_type, work\_type>{get\_num(), get\_num()},

std::pair<work\_type, work\_type>{get\_num(), get\_num()}

});

break;

case 3:

redactor.insert(pos,

std::vector<std::pair<work\_type, work\_type>>{

std::pair<work\_type, work\_type>{get\_num(), get\_num()},

std::pair<work\_type, work\_type>{get\_num(), get\_num()},

std::pair<work\_type, work\_type>{get\_num(), get\_num()},

std::pair<work\_type, work\_type>{get\_num(), get\_num()}

});

break;

}

break;

}

case 6:{

std::cout<<"input figure position for removing"<<std::endl<<">>"<<std::flush;

redactor.remove(get\_num());

break;

}

case 7:

redactor.undo();

break;

case 8:

redactor.redo();

break;

case 9:{

std::cout<<"Choose input figure number for action or -1 for all figure "<<std::endl<<">>"<<std::flush;

int pos =get\_num();

std::cout<<"Choose action:"<<std::endl

<<"0. print all squares"<<std::endl

<<"1. print all centers"<<std::endl

<<"2. print full information about figure"<<std::endl

<<">>"<<std::flush;

if(pos == -1)

redactor.do\_for\_all(get\_num());

else

redactor.do\_for\_one(pos, get\_num());

break;

}

}

}catch(std::logic\_error& e){

std::cerr<<"logic error: "<<e.what()<<std::endl;

}catch(std::runtime\_error& e){

std::cerr<<"runtime error: "<<e.what()<<std::endl;

}

}

//return 0;

}

**redactor**

#pragma once

#include<string>

#include<fstream>

#include<exception>

#include<deque>

#include<vector>

#include<memory>

#include"factory"

#include"figures"

template<class T>

class Redactor{

private:

class Action{

public:

Action(bool c, unsigned long long p, std::shared\_ptr<Figure<T>> o\_p): cmd(c), pos(p), obj\_ptr(o\_p) {}

bool cmd; //insert = true ; remove = false

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

unsigned long long pos;

};

std::deque<Action> done, canceled;

std::vector<std::shared\_ptr<Figure<T>>> figures;

public:

void New(){

done.clear();

canceled.clear();

figures.clear();

return;

}

void save(const std::string& file\_name){

std::ofstream file(file\_name, std::ios::binary);

if(!file.good())

throw std::runtime\_error("can't open file");

std::size\_t size=figures.size();

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

for(std::shared\_ptr<Figure<T>>& elem : figures){

int type=elem->get\_type();

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

std::vector<std::pair<T, T>> points=elem->get\_vertex();

switch(type){

case 1:{

file.write((char\*) &(points[0].first), sizeof(points[0].first));

file.write((char\*) &(points[0].second), sizeof(points[0].second));

file.write((char\*) &(points[2].first), sizeof(points[2].first));

file.write((char\*) &(points[2].second), sizeof(points[2].second));

break;

}

case 2:{

file.write((char\*) &(points[0].first), sizeof(points[0].first));

file.write((char\*) &(points[0].second), sizeof(points[0].second));

file.write((char\*) &(points[1].first), sizeof(points[1].first));

file.write((char\*) &(points[1].second), sizeof(points[1].second));

file.write((char\*) &(points[2].first), sizeof(points[2].first));

file.write((char\*) &(points[2].second), sizeof(points[2].second));

break;

}

case 3:{

file.write((char\*) &(points[0].first), sizeof(points[0].first));

file.write((char\*) &(points[0].second), sizeof(points[0].second));

file.write((char\*) &(points[1].first), sizeof(points[1].first));

file.write((char\*) &(points[1].second), sizeof(points[1].second));

file.write((char\*) &(points[2].first), sizeof(points[2].first));

file.write((char\*) &(points[2].second), sizeof(points[2].second));

file.write((char\*) &(points[3].first), sizeof(points[3].first));

file.write((char\*) &(points[3].second), sizeof(points[3].second));

break;

}

}

}

file.close();

}

void load(const std::string& file\_name){

std::ifstream file(file\_name, std::ios::binary);

if(!file.good())

throw std::runtime\_error("can't open file");

New();

std::size\_t size=0;

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

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

int type;

T ar[4];

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

switch(type){

case 1:{

file.read((char\*)&ar, sizeof(T)\*4);

std::vector<std::pair<T, T>>buf{

std::pair<T, T>{ar[0], ar[1]},

std::pair<T, T>{ar[2], ar[3]}

} ;

figures.push\_back( Factory<T>::create(buf));

break;

}

case 2:{

file.read((char\*)&ar, sizeof(T)\*6);

std::vector<std::pair<T, T>>buf{

std::pair<T, T>{ar[0], ar[1]},

std::pair<T, T>{ar[2], ar[3]},

std::pair<T, T>{ar[4], ar[5]}

};

figures.push\_back( Factory<T>::create(buf));

break;

}

case 3:{

file.read((char\*)&ar, sizeof(T)\*8);

std::vector<std::pair<T, T>>buf{

std::pair<T, T>{ar[0], ar[1]},

std::pair<T, T>{ar[2], ar[3]},

std::pair<T, T>{ar[4], ar[5]},

std::pair<T, T>{ar[6], ar[7]}

};

figures.push\_back( Factory<T>::create( buf));

break;

}

}

}

file.close();

}

void insert(unsigned long long pos, const std::vector<std::pair<T, T>>& points){

if(pos>figures.size())

throw std::logic\_error("Position for inserting figure is too big");

canceled.clear();

figures.insert(std::next(figures.begin(), pos) , Factory<T>::create(points));

done.push\_back(Action{true, pos, figures[pos]});

return;

}

void remove(unsigned long long pos){

if(pos>figures.size())

throw std::logic\_error("Position for removing figure is too big");

canceled.clear();

done.push\_back(Action{false, pos, figures[pos]});

figures.erase(std::next(figures.begin(), pos));

return;

}

void undo(){

if(done.empty())

throw std::logic\_error("No action to undo");

Action cur=done.back();

done.pop\_back();

if(cur.cmd){

figures.erase(std::next(figures.begin(), cur.pos));

std::cout<<"Removed figure from pos "<<cur.pos<<std::endl<<">>"<<std::flush;

}else{

figures.insert(std::next(figures.begin(), cur.pos) , cur.obj\_ptr);

std::cout<<"Insert figure in pos "<<cur.pos<<std::endl<<">>"<<std::flush;

}

canceled.push\_back(Action{!cur.cmd, cur.pos, cur.obj\_ptr});

}

void redo(){

if(canceled.empty())

throw std::logic\_error("No action to redo");

Action cur=canceled.back();

canceled.pop\_back();

if(cur.cmd){

std::cout<<"Removed figure from pos "<<cur.pos<<std::endl<<">>"<<std::flush;

figures.erase(std::next(figures.begin(), cur.pos));

}else{

std::cout<<"Insert figure in pos "<<cur.pos<<std::endl<<">>"<<std::flush;

figures.insert(std::next(figures.begin(), cur.pos) , cur.obj\_ptr);

}

done.push\_back(Action{(!cur.cmd), cur.pos, cur.obj\_ptr});

}

void do\_for\_one(unsigned long long pos, int cmd){

if(pos>figures.size())

throw std::logic\_error("Figure position for action is too big");

switch(cmd){

case 0:

std::cout<<"figure "<<pos<<" has square = "<<figures[pos]->square()<<std::endl;

break;

case 1:

std::cout<<"figure "<<pos<<" has center = "<<figures[pos]->center()<<std::endl;

break;

case 2:

figures[pos]->print();

break;

default:

throw std::logic\_error("Wrong command for action");

}

}

void do\_for\_all(int cmd){

if(figures.size()==0)

throw std::logic\_error("buffer of figures empty");

switch(cmd){

case 0:

for(unsigned long long pos=0; pos<figures.size(); ++pos)

std::cout<<"figure "<<pos<<" has square = "<<figures[pos]->square()<<std::endl;

break;

case 1:

for(unsigned long long pos=0; pos<figures.size(); ++pos)

std::cout<<"figure "<<pos<<" has center = "<<figures[pos]->center()<<std::endl;

break;

case 2:

for(unsigned long long pos=0; pos<figures.size(); ++pos){

std::cout<<pos<<". "; figures[pos]->print();

}

break;

default:

throw std::logic\_error("Wrong command for action");

}

}

};

**figures**

#pragma once

#include<iostream>

#include<cmath>

#include<exception>

#include<vector>

#include<tuple>

template <class T>

double length(std::pair<T, T> p1, std::pair<T, T> p2){

return sqrt((p2.first-p1.first)\*(p2.first-p1.first) + (p2.second-p1.second)\*(p2.second-p1.second));

}

template <class T>

double cos(std::pair<T, T> p1, std::pair<T, T> p2, std::pair<T, T> p3, std::pair<T, T> p4){

return ( (p2.first-p1.first) \* (p4.first-p3.first)

+ (p2.second-p1.second) \* (p4.second-p3.second) )

/ ( length(p1, p2) \* length( p3, p4) );

}

template <class T>

std::ostream& operator <<(std::ostream &o, const std::pair<T, T>& a){

o<<"( "<<a.first<<" , "<<a.second<<" )";

return o;

}

template <class T>

std::ostream& operator <<(std::ostream &o, const std::pair<T, T>&& a){

o<<"( "<<a.first<<" , "<<a.second<<" )";

return o;

}

template <class T>

std::ostream& operator <<(std::ostream &o, std::pair<T, T>& a){

o<<"( "<<a.first<<" , "<<a.second<<" )";

return o;

}

template <class T>

std::ostream& operator <<(std::ostream &o, std::pair<T, T>&& a){

o<<"( "<<a.first<<" , "<<a.second<<" )";

return o;

}

template<class T>

class Figure{

protected:

using type = T;

using vertex = std::pair<type, type>;

using vertexes = std::vector<vertex>;

vertexes points;

int figure\_type=0;

double squar=0;

vertex centr{0,0};

public:

void print(){

switch(figure\_type){

case 0:

throw std::logic\_error("Figure is not created");

case 1:

std::cout<<"Figure - Square"<<std::endl;

break;

case 2:

std::cout<<"Figure - Rectangle"<<std::endl;

break;

case 3:

std::cout<<"Figure - Trapeze"<<std::endl;

break;

default:

throw std::logic\_error("Figure type is unknown");

}

std::cout<<"vertexes: "<<std::endl;

for(const vertex& ver: points)

std::cout<<'\t'<<ver<<std::endl;

std::cout<<"square = "<<squar<<std::endl;

std::cout<<"center = "<<centr<<std::endl;

}

double square(){

return squar;

}

vertex center(){

return centr;

}

int get\_type(){

return figure\_type;

}

vertexes get\_vertex(){

return points;

}

};

template <class T>

class Square: public Figure<T>{

public:

using typename Figure<T>::type;

using typename Figure<T>::vertex;

using typename Figure<T>::vertexes;

using Figure<T>::squar;

using Figure<T>::centr;

using Figure<T>::points;

using Figure<T>::figure\_type;

Square(const vertexes& p){

if( (p.size() == 2) &&

(p[0].first!=p[1].first) || (p[0].second!=p[1].second) ){

points = vertexes{

p[0],

vertex{

-p[1].second + (p[1].second + p[0].second + p[1].first + p[0].first)/2,

p[1].first + (p[1].second + p[0].second - p[1].first - p[0].first)/2

},

p[1],

vertex{

p[1].second + (p[1].first + p[0].first - p[1].second - p[0].second)/2,

-p[1].first + (p[1].second + p[0].second + p[1].first + p[0].first)/2

}

};

std::pair<T, T> c {

-p[1].second + (p[1].second + p[0].second + p[1].first + p[0].first)/2,

p[1].first + (p[1].second + p[0].second - p[1].first - p[0].first)/2

};

squar = (double) length( p[0] , c )

\* length( p[1] , c );

centr = std::pair{

( p[0].first + p[1].first ) / 2,

( p[0].second + p[1].second ) / 2

};

figure\_type=1;

}else

throw std::logic\_error("it's not a square");

}

};

template <class T>

class Rectangle: public Figure<T>{

public:

using typename Figure<T>::type;

using typename Figure<T>::vertex;

using typename Figure<T>::vertexes;

using Figure<T>::squar;

using Figure<T>::centr;

using Figure<T>::points;

using Figure<T>::figure\_type;

Rectangle(const vertexes& p){

if( (p.size()==3) &&

(p[0].first!=p[1].first || p[0].second!=p[1].second)

&& (p[2].first!=p[1].first || p[2].second!=p[1].second)

&& (p[0].first!=p[2].first || p[0].second!=p[2].second)

&& (cos(p[0], p[1], p[1], p[2])<=1e-9 && cos(p[0], p[1], p[1], p[2])>=-1e-9) ) {

points=std::vector{p[0], p[1], p[2], std::pair<T, T>{

p[2].first-p[1].first+p[0].first,

p[2].second-p[1].second+p[0].second

}

};

squar=(double) length( p[0], p[1] )

\* length( p[1], p[2] );

centr=std::pair{

( p[0].first + p[2].first ) / 2,

( p[0].second + p[2].second ) / 2

};

figure\_type=2;

}else

throw std::logic\_error("it's not a rectangle");

}

};

template <class T>

class Trapeze: public Figure<T>{

public:

using typename Figure<T>::type;

using typename Figure<T>::vertex;

using typename Figure<T>::vertexes;

using Figure<T>::squar;

using Figure<T>::centr;

using Figure<T>::points;

using Figure<T>::figure\_type;

Trapeze(const vertexes& p){

if( p.size()==4 &&

(( length(p[0], p[1]) - length(p[2], p[3])<=1e-9) && ( length(p[0], p[1]) - length(p[2], p[3])>=-1e-9 ) )

&&!( (length(p[0], p[1]) <= 1e-9) && (length(p[0], p[1]) >= -1e-9) )

&& ( ( (cos(p[0], p[3], p[2], p[1]) <= 1+1e-9) && (cos(p[0], p[3], p[2], p[1]) >= 1-1e-9) )

|| ( (cos(p[0], p[3], p[2], p[1])<=-1+1e-9) && (cos(p[0], p[3], p[2], p[1])>=-1-1e-9) ) )

) {

points=p;

squar = (double)

(length(p[0], p[3])+length(p[1], p[2]))/2

\* length(p[1], std::pair<T, T> {

p[0].first + ( length(p[0], p[3]) - length(p[1], p[2]) ) / 2

\* ( length(p[0], std::pair{ p[3].first, p[0].second} ) / length(p[0], p[3]) ),

p[0].second + ( length(p[0], p[3])-length(p[1], p[2]) )/2

\* ( length(p[3], std::pair{p[3].first, p[0].second}) / length(p[0], p[3]) )

}

);

centr = std::pair{

( p[0].first + p[1].first + p[2].first + p[3].first ) / 4,

( p[0].second + p[1].second + p[2].second + p[3].second ) / 4

};

figure\_type=3;

}else

throw std::logic\_error("it's not a trapeze");

}

};

**Factory**

#pragma once

#include<memory>

#include "figures"

template<class T> class Factory{

public:

static std::shared\_ptr<Figure<T>> create(const std::vector<std::pair<T, T>>& points){

switch(points.size()){

case 2:

return std::shared\_ptr<Figure<T>>(new Square<T>(points));

case 3:

return std::shared\_ptr<Figure<T>>(new Rectangle<T>(points));

case 4:

return std::shared\_ptr<Figure<T>>(new Trapeze<T>(points));

}

return 0;

}

};

1. **Вывод**

Научился строить структуру классов на языке С++.

**Список литературы**

1. Справочник по языку С++ [Электронный ресурс]. URL:

<https://en.cppreference.com/w/> (дата обращения: 25.12.2019).