ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ­­

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | А. В. Устинов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5 |
| ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4631 |  |  |  | Д.В.Килин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2018

**1. Цель работы**

Изучить принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Закрепить знания по теме: Обработка исключительных ситуаций.

**2. Листинг программы**

#pragma once

// Файл shape.h

#ifndef SHAPE\_H

#define SHAPE\_H

#include "screen.h"

class Shape

{

public:

virtual void draw(Screen\* screen) const = 0;

virtual void move(Point p) = 0;

virtual Point getLeftTop() const = 0;

virtual Point getRightTop() const = 0;

virtual Point getLeftBottom() const = 0;

virtual Point getRightBottom() const = 0;

virtual ~Shape() {}

};

#endif // SHAPE\_H

#pragma once

// Файл line.h

#ifndef LINE\_H

#define LINE\_H

#include "shape.h"

#include "screen.h"

#include <algorithm>

class Line : public Shape

{

public:

Line(const Point& a, const Point& b)

{

\_a = a;

\_b = b;

}

Line(unsigned int x0, unsigned int y0, unsigned int x1, unsigned int y1)

{

\_a = Point(x0, y0);

\_b = Point(x1, y1);

}

Line(const Line& source)

{

\_a = source.\_a;

\_b = source.\_b;

}

void draw(Screen\* screen) const

{

screen->putLine(\_a.getY(), \_a.getX(), \_b.getY(), \_b.getX());

}

void move(Point p)

{

\_a.setX(\_a.getX() + p.getX()); \_a.setY(\_a.getY() + p.getY());

\_b.setX(\_b.getX() + p.getX()); \_b.setY(\_b.getY() + p.getY());

}

Point getLeftTop() const

{

return Point(std::min(\_a.getX(), \_b.getX()), std::min(\_a.getY(), \_b.getY()));

}

Point getRightTop() const

{

return Point(std::max(\_a.getX(), \_b.getX()), std::min(\_a.getY(), \_b.getY()));

}

Point getLeftBottom() const

{

return Point(std::min(\_a.getX(), \_b.getX()), std::max(\_a.getY(), \_b.getY()));

}

Point getRightBottom() const

{

return Point(std::max(\_a.getX(), \_b.getX()), std::max(\_a.getY(), \_b.getY()));

}

Point getFirstPoint() const

{

return \_a;

}

void EditLine(const Point& a, const Point& b)

{

\_a = a;

\_b = b;

}

Point getSecondPoint() const

{

return \_b;

}

protected:

Point \_a;

Point \_b;

};

#endif // LINE\_H

#pragma once

//Файл square.h

#ifndef RHOMB\_H

#define RHOMB\_H

#include "line.h"

#include "Error.h"

using namespace std;

class Rhomb : public virtual Shape

{

public:

/\*Rhomb(const Point& left, const Point& top,const Point& right, const Point& bottom)\*/

Rhomb(const Point& dot)

{

\_leftRight = new Line(Point(dot.getX() -4, dot.getY()), Point(dot.getX()+4, dot.getY()));

\_leftTop = new Line(Point(dot.getX() - 4, dot.getY()), Point(dot.getX() , dot.getY()-4));

\_topRight = new Line(Point(dot.getX(), dot.getY()-4), Point(dot.getX() + 4, dot.getY()));

\_leftBot = new Line(Point(dot.getX() - 4, dot.getY()), Point(dot.getX(), dot.getY()+4));

\_rightBot = new Line(Point(dot.getX() + 4, dot.getY()), Point(dot.getX(), dot.getY()+4));

\_topBot = new Line(Point(dot.getX(), dot.getY()-4), Point(dot.getX(), dot.getY()+4));

}

void DrawExeptions(Screen\* screen)const {

\_leftRight->draw(screen);

\_leftTop->draw(screen);

\_topRight->draw(screen);

\_leftBot->draw(screen);

\_rightBot->draw(screen);

\_topBot->draw(screen);

}

virtual void draw(Screen\* screen) const

{

try {

if (((\_leftRight->getFirstPoint().getX()) < 0) || (\_leftRight->getFirstPoint().getX()) >100) {

cout << ((\_leftRight->getFirstPoint()).getX());

Point p = \_leftRight->getFirstPoint();

cout << p.getX();

std::cout << "err" << endl;

throw MyError("out of screen range", "rhomb");

}

DrawExeptions(screen);

}

catch (MyError err) {

int newX, newY;

cout << err.message << endl;

cout << err.TypeOfShape << endl;

do{

cout << "enter new dot coordinates " << endl;

cout << "X: " << endl;

cin >> newX;

cout << "Y: " << endl;

cin >> newY;

}

while (((newX - 4) < 0) || ((newX + 4) > 100) || ((newY - 4) < 0) || ((newY + 4) > 100));

Point newdot(newX, newY);

{\_leftRight->EditLine(Point(newdot.getX() - 4, newdot.getY()), Point(newdot.getX() + 4, newdot.getY()));

\_leftRight->EditLine(Point(newdot.getX() - 4, newdot.getY()), Point(newdot.getX() + 4, newdot.getY()));

\_leftTop->EditLine(Point(newdot.getX() - 4, newdot.getY()), Point(newdot.getX(), newdot.getY() - 4));

\_topRight->EditLine(Point(newdot.getX(), newdot.getY() - 4), Point(newdot.getX() + 4, newdot.getY()));

\_leftBot->EditLine(Point(newdot.getX() - 4, newdot.getY()), Point(newdot.getX(), newdot.getY() + 4));

\_rightBot->EditLine(Point(newdot.getX() + 4, newdot.getY()), Point(newdot.getX(), newdot.getY() + 4));

\_topBot->EditLine(Point(newdot.getX(), newdot.getY() - 4), Point(newdot.getX(), newdot.getY() + 4));

}

DrawExeptions(screen);

// throw MyError("out of screen range", "rhomb");

}

}

virtual void move(Point p)

{

\_leftRight->move(p);

}

Point getLeftTop() const

{

return \_leftRight->getLeftTop();

}

Point getRightTop() const

{

return \_leftRight->getLeftTop();

}

Point getLeftBottom() const

{

return \_leftRight->getLeftTop();

//return \_left->getLeftBottom();

}

Point getRightBottom() const

{

return \_leftRight->getLeftTop();//return \_right->getRightBottom();

}

virtual ~Rhomb()

{

delete \_leftRight;

}

protected:

Line \* \_leftRight, \*\_leftTop,\*\_topRight, \*\_leftBot, \*\_rightBot,\*\_topBot;

};

#endif //RHOMB\_H

#pragma once

//Файл screen.h

#ifndef SCREEN\_H

#define SCREEN\_H

#include "point.h"

#include <vector>

#include <iostream>

class Screen

{

public:

enum class Pixel : char

{

WHITE = ' ',

BLACK = '\*'

};

Screen(std::uint32\_t xSize, std::uint32\_t ySize)

{

\_xSize = xSize;

\_ySize = ySize;

\_screen.reserve(ySize);

for (size\_t i = 0; i < ySize; i++)

\_screen.emplace\_back(std::vector<Pixel>(xSize, Pixel::WHITE));

}

void putLine(const Point& a, const Point& b)

{

putLine(a.getX(), a.getY(), b.getX(), b.getY());

}

void putLine(std::uint32\_t x0, std::uint32\_t y0, std::uint32\_t x1, std::uint32\_t y1)

{

std::int32\_t dx = 1;

std::int32\_t a = x1 - x0;

if (a < 0) { dx = -1; a = -a; }

std::int32\_t dy = 1;

std::int32\_t b = y1 - y0;

if (b < 0) { dy = -1; b = -b; }

std::int32\_t two\_a = 2 \* a;

std::int32\_t two\_b = 2 \* b;

std::int32\_t xcrit = -b + two\_a;

std::int32\_t eps = 0;

while (true)

{

putPoint(x0, y0);

if (x0 == x1 && y0 == y1) break;

if (eps <= xcrit)

{

x0 += dx;

eps += two\_b;

}

if (eps >= a || a < b)

{

y0 += dy;

eps -= two\_a;

}

}

}

void putPoint(const Point& p)

{

\_screen[p.getX()][p.getY()] = Pixel::BLACK;

}

void putPoint(const std::uint32\_t x, const std::uint32\_t y)

{

/\*\_screen[x][y] = Pixel::BLACK;\*/

\_screen.at(x).at(y) = Pixel::BLACK;

}

void clear()

{

for (size\_t i = 0; i < \_ySize; ++i)

for (size\_t j = 0; j < \_xSize; ++j)

\_screen[i][j] = Pixel::WHITE;

}

void draw() const

{

for (size\_t i = 0; i < \_ySize; ++i)

{

for (size\_t j = 0; j < \_xSize; ++j)

std::cout << (char)\_screen[i][j];

std::cout << std::endl;

}

}

private:

std::uint32\_t \_xSize;

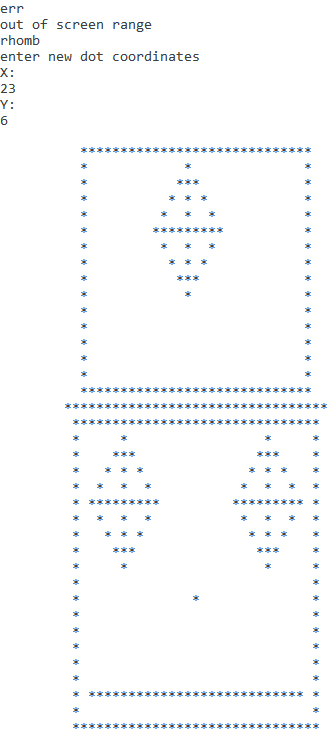
std::uint32\_t \_ySize;

std::vector<std::vector<Pixel> > \_screen;

};

#endif // SCREEN\_H

**3. Пример работы**

****

**4. Вывод**

Изучил принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования. Добавил в программу обработчик ошибок, с возможностью ввода новых координат фигур.