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 |  |  |  | С.А. Гришин |
|  |  |  | 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

#include "screen.hpp"

#include <string>

class Shape {

public:

static int idIdx;

Shape() { id = idIdx++; }

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

virtual void move(Point p) = 0;

virtual Point getLeftTop() const = 0;

virtual std::string getName() const = 0;

virtual Point getRightTop() const = 0;

virtual Point getLeftBottom() const = 0;

virtual Point getRightBottom() const = 0;

int getID() const { return id; };

virtual ~Shape() {}

private:

int id;

};

int Shape::idIdx = 0;

#pragma once

#include "shape.hpp"

class Circle : public virtual Shape {

public:

Circle(uint32\_t x, uint32\_t y, uint32\_t radius) : \_x(x), \_y(y), \_r(radius) {}

virtual void draw(Screen \*screen) const {

int y = 0;

int x = \_r;

int delta = 1 - 2 \* \_r;

int error = 0;

while (x >= 0) {

try {

screen->putPoint(\_y + y, \_x + x);

} catch (...) {

std::cerr << Shape::getID() << ": Out of Range screen in " << getName()

<< '\n';

}

try {

screen->putPoint(\_y + y, \_x - x);

} catch (...) {

std::cerr << Shape::getID() << ": Out of Range screen in " << getName()

<< '\n';

}

try {

screen->putPoint(\_y - y, \_x + x);

} catch (...) {

std::cerr << Shape::getID() << ": Out of Range screen in " << getName()

<< '\n';

}

try {

screen->putPoint(\_y - y, \_x - x);

} catch (...) {

std::cerr << Shape::getID() << ": Out of Range screen in " << getName()

<< '\n';

}

error = 2 \* (delta + x) - 1;

if (delta < 0 && error <= 0) {

++y;

delta += 2 \* y + 1;

continue;

}

error = 2 \* (delta - y) - 1;

if (delta > 0 && error > 0) {

--x;

delta += 1 - 2 \* x;

continue;

}

++y;

delta += 2 \* (y - x);

--x;

}

}

virtual void move(Point p) {

\_x = \_x + p.getX();

\_y = \_y + p.getY();

}

std::string getName() const { return "Circle"; }

Point getLeftTop() const { return Point(\_x - \_r, \_y - \_r); }

Point getRightTop() const { return Point(\_x + \_r, \_y - \_r); }

Point getLeftBottom() const { return Point(\_x - \_r, \_y + \_r); }

Point getRightBottom() const { return Point(\_x + \_r, \_y + \_r); }

virtual ~Circle() {}

protected:

uint32\_t \_x, \_y, \_r;

};

#pragma once

#include "screen.hpp"

#include "shape.hpp"

#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 {

try {

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

} catch (...) {

std::cerr << Shape::getID() << ": Out of Range screen in " << getName()

<< '\n';

}

}

std::string getName() const { return "Line"; }

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; }

Point getSecondPoint() const { return \_b; }

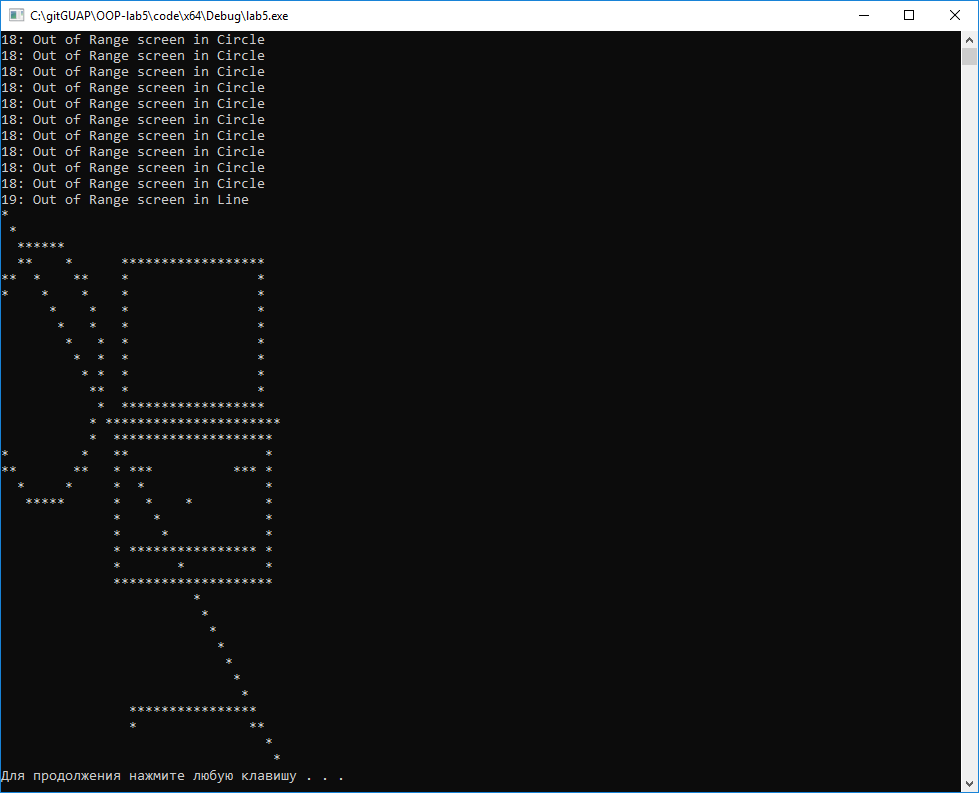
protected:

Point \_a;

Point \_b;

};

**3. Пример работы**



**4. Вывод**

Изучил принципы построения консольных приложений, применив на практике знания базовых синтаксических конструкций языка C++ и объектно-ориентированного программирования.