**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: «Поддержка обработки исключительных ситуаций**»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 7307 | Шалугин Е.Д. Васильев А.В. |  |
| Преподаватель | Колинько П.Г. |  |

Санкт-Петербург

2019

Оглавление

[Цель работы 3](#__RefHeading___Toc188_4150899979)

[Задание 3](#__RefHeading___Toc190_4150899979)

[Набор и вид классов для фиксации исключительных ситуаций 3](#__RefHeading___Toc302_2376616454)

[Тестирование исключительных ситуаций 4](#__RefHeading___Toc304_2376616454)

[Добавленный код 5](#__RefHeading___Toc306_2376616454)

[Вывод 7](#__RefHeading___Toc308_2376616454)

# **Цель работы**

Изучить механизм исключительных ситуаций на примере программы, рисующей фигуры.

# **Задание**

Переработать программу работы с библиотекой фигур, дополнив её

механизмом контроля исключительных ситуаций.

# **Набор и вид классов для фиксации исключительных ситуаций**

Для обработки исключительных ситуаций были добавлены 5 классов:

* 1. 1) PointOutOfScreenException
  2. 2) XOfPointOutOfScreenException
  3. 3)YOfPointOutOfScreenException
  4. 4)ShapeOutOfScreenException
  5. 5)WrongShapeException

Классы не наследуются от exception, но имеют функцию what(), которая выводит сообщения об ошибке.

Исключение 1) – точка по координатам X и Y находится вне экрана.

Исключение 2) – точка по координате X выходит за границы.

Исключение 3) – точка по координате Y выходит за границы.

Исключение 4) – фигура находится вне экрана.

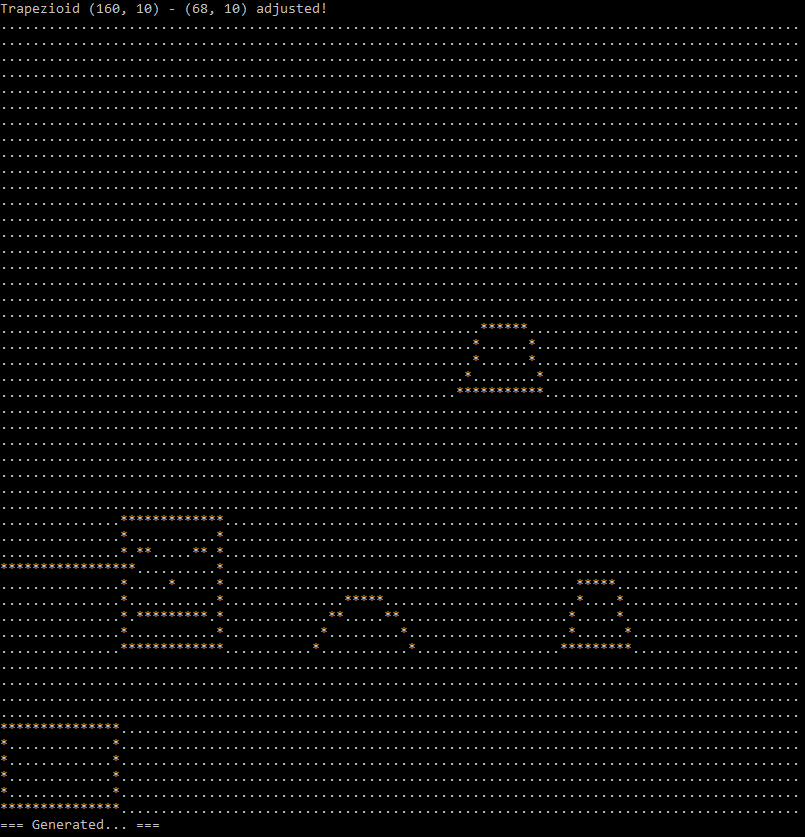
Исключение 5) – фигура задана неправильно.

Изменена функция put\_point – обработка исключений непопадания точки на экран.

Так же в классах описана проверка исключений для фигур.

# **Тестирование исключительных ситуаций**

# **Фигура была задана с неправильными размерами.**

1. 
2. 

# **Добавленный код**

1. void put\_point(int a, int b)
2. {
3. try
4. {
5. if ((b > YMAX || b < 0) && (a > XMAX || a < 0)) throw PointOutOfScreenException();
6. if (b > YMAX || b < 0) throw YOfPointOutOfScreenException();
7. if (a > XMAX || a < 0) throw XOfPointOutOfScreenException();
8. screen[b][a] = black;
9. }
10. catch (XOfPointOutOfScreenException& e)
11. {
12. cout << e.what() << endl;
13. }
14. catch (YOfPointOutOfScreenException& e)
15. {
16. cout << e.what() << endl;
17. }
18. catch (PointOutOfScreenException& e)
19. {
20. cout << e.what() << endl;
21. }

# }

class rectangle : public rotatable {

/\* nw ------ n ------ ne

| |

| |

w c e

| |

| |

sw ------ s ------ se \*/

protected:

point sw, ne;

public:

//NNNNNNNNNNNNNNNAAAAAAAAAAAAAAAAAAAAAAAAA

rectangle(point a, point b)

{

try

{

if (a.x > XMAX || a.x < 0 ||

a.y > YMAX || a.y < 0 ||

b.x > XMAX || b.x < 0 ||

b.y > YMAX || b.y < 0) {

throw ShapeOutOfScreenException();

flag = true;

}

else

if (sw.x > ne.x || sw.y > ne.y) {

throw WrongShapeException();

flag = true;

}

sw = a;

ne = b;

}

catch (ShapeOutOfScreenException)

{

cout << "Rectangle (" << a.x << ", " << a.y << ") - (" << b.x << ", " << b.y << ") is out of screen!" << endl;

sw = point::zero();

ne = point::zero();

}

catch (WrongShapeException)

{

cout << "Rectangle (" << a.x << ", " << a.y << ") - (" << b.x << ", " << b.y << ") adjusted!" << endl;

sw.x = a.x < b.x ? a.x : b.x;

sw.y = a.y < b.y ? a.y : b.y;

ne.x = a.x > b.x ? a.x : b.x;

ne.y = a.y > b.y ? a.y : b.y;

}

}

//NNNNNNNNNNNNNNNAAAAAAAAAAAAAAAAAAAAAAAAA

//rectangle(point a, point b) : sw(a), ne(b) { }

point north() const { return point((sw.x + ne.x) / 2, ne.y); }

point south() const { return point((sw.x + ne.x) / 2, sw.y); }

point east() const { return point(ne.x, (sw.y + ne.y) / 2); }

point west() const { return point(sw.x, (sw.y + ne.y) / 2); }

point neast() const { return ne; }

point seast() const { return point(ne.x, sw.y); }

point nwest() const { return point(sw.x, ne.y); }

point swest() const { return sw; }

void rotate\_right() // Поворот вправо относительно se

{

int w = ne.x - sw.x, h = ne.y - sw.y;

sw.x = ne.x - h \* 2; ne.y = sw.y + w / 2;

}

void rotate\_left() // Поворот влево относительно sw

{

int w = ne.x - sw.x, h = ne.y - sw.y;

ne.x = sw.x + h \* 2; ne.y = sw.y + w / 2;

}

void move(int a, int b)

{

sw.x += a; sw.y += b; ne.x += a; ne.y += b;

}

void resize(int d)

{

ne.x += (ne.x - sw.x) \* (d - 1); ne.y += (ne.y - sw.y) \* (d - 1);

}

void draw();

};

void rectangle::draw()

{

put\_line(nwest(), ne); put\_line(ne, seast());

put\_line(seast(), sw); put\_line(sw, nwest());

# }

class trapezioid : public rotatable, public reflectable {

/\* nw ------ n ------ ne

/ \

/ \

e c w

/ \

/ \

sw ----------- s ----------- se \*/

protected:

point sw, se, nw, ne;

public:

trapezioid(point a, point b, point c, point d) : sw(a), se(b), nw(c), ne(d)

{

try

{

if (b.x > XMAX || a.x < 0 ||

c.y > YMAX || a.y < 0 ||

d.x > XMAX || c.x < 0) {

flag = true;

throw ShapeOutOfScreenException();

}

else

if (a.y - c.y != b.y - d.y || a.x > b.x || c.x > d.x || c.y < a.y) {

flag = true;

throw WrongShapeException();

}

//sw = a;

//ne = b;

}

catch (ShapeOutOfScreenException)

{

cout << "Trapezioid (" << a.x << ", " << a.y << ") - (" << b.x << ", " << b.y << ") is out of screen!" << endl;

//sw = point::zero();

//ne = point::zero();

}

catch (WrongShapeException)

{

cout << "Trapezioid (" << a.x << ", " << a.y << ") - (" << b.x << ", " << b.y << ") adjusted!" << endl;

//sw.x = a.x < b.x ? a.x : b.x;

//sw.y = a.y < b.y ? a.y : b.y;

//ne.x = a.x > b.x ? a.x : b.x;

//ne.y = a.y > b.y ? a.y : b.y;

}

}

//trapezioid(point a, point b, point c, point d) : sw(a), se(b), nw(c), ne(d) { }

point north() const { return point((nw.x + ne.x) / 2, ne.y); }

point south() const { return point((sw.x + se.x) / 2, sw.y); }

point east() const { return point((ne.x + se.x) / 2, (se.y + ne.y) / 2); }

point west() const { return point((sw.x + nw.x) / 2, (sw.y + nw.y) / 2); }

point neast() const { return ne; }

point seast() const { return se; }

point nwest() const { return nw; }

point swest() const { return sw; }

void rotate\_right() // Поворот вправо относительно se

{

int l1 = nw.x - sw.x;

int h = nw.y - sw.y;

int l2 = se.x - ne.x;

se.y = sw.y - (se.x - sw.x);

se.x = sw.x;

nw.x = se.x + h;

nw.y = sw.y - l1;

ne.x = nw.x;

ne.x = nw.x;

ne.y = se.y + l2;

//int w = ne.x - sw.x, h = ne.y - sw.y;

//sw.x = ne.x - h \* 2; ne.y = sw.y + w / 2;

}

void rotate\_left() // Поворот влево относительно sw

{

int l1 = nw.x - sw.x;

int h = nw.y - sw.y;

int l2 = se.x - ne.x;

se.y = sw.y + (se.x - sw.x);

se.x = sw.x;

nw.x = se.x - h;

nw.y = sw.y + l1;

ne.x = nw.x;

ne.y = se.y - l2;

//int w = ne.x - sw.x, h = ne.y - sw.y;

//ne.x = sw.x + h \* 2; ne.y = sw.y + w / 2;

}

void move(int a, int b)

{

sw.x += a; sw.y += b; ne.x += a; ne.y += b;

nw.x += a; nw.y += b; se.x += a; se.y += b;

}

void flip\_horisontally() {

int p1 = nw.y;

nw.y = se.y;

ne.y = se.y;

se.y = p1;

sw.y = p1;

}

void flip\_vertically() {

int p1 = nw.x;

nw.x = ne.x;

ne.x = p1;

p1 = sw.x;

sw.x = se.x;

se.x = p1;

}

void resize(int blabla){

}

void draw();

};

void trapezioid::draw()

{

put\_line(nwest(), ne); put\_line(ne, seast());

put\_line(seast(), sw); put\_line(sw, nwest());

}

# **Вывод**

1. В данной работе мы изучили механизм исключительных ситуаций. Внедрение в программу данного механизма повысило устойчивость программы, позволило исправлять ошибки пользователя. Данная программа берет идеи предыдущей, но не является ее расширением, а лишь демонстрирует возможности механизма перехвата исключений. В базовом классе shape создан flag, который хранит знание о том правильно ли нарисована фигура или нет. В случае если фигура нарисована неверно, она в дальнейшем не выводится на экран, что позволяет работать с остальными фигурами дальше и не завершать работу программы.