**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**»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7307 | Торопов В.А. |  |
| 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) ExceptionPoint
  2. 2) ExceptionX
  3. 3) ExceptionY
  4. 4) ExceptionShape

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.

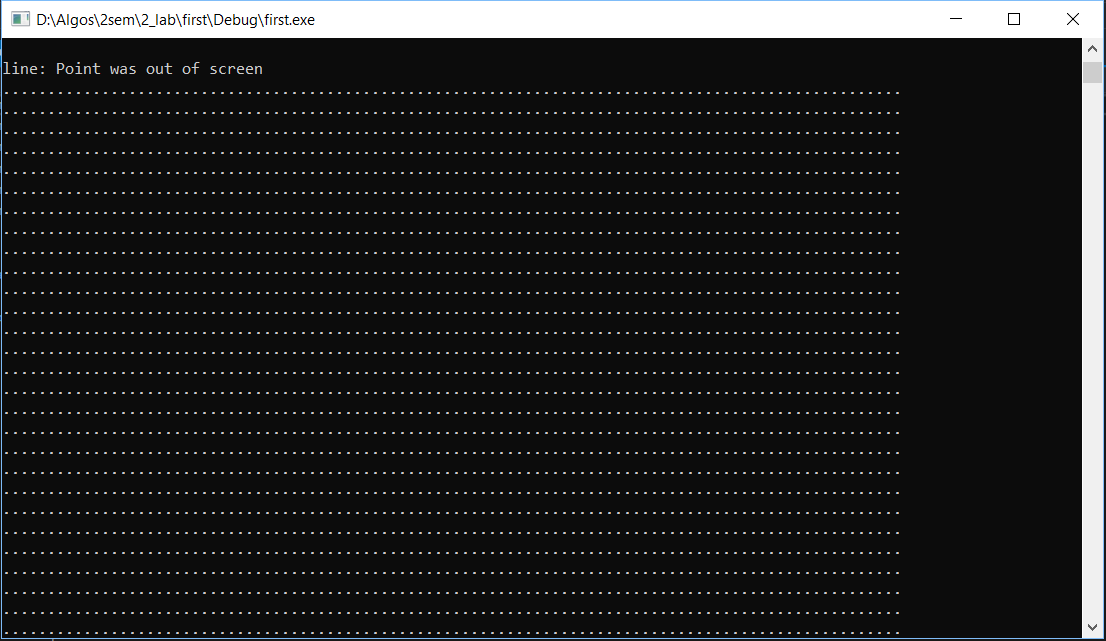
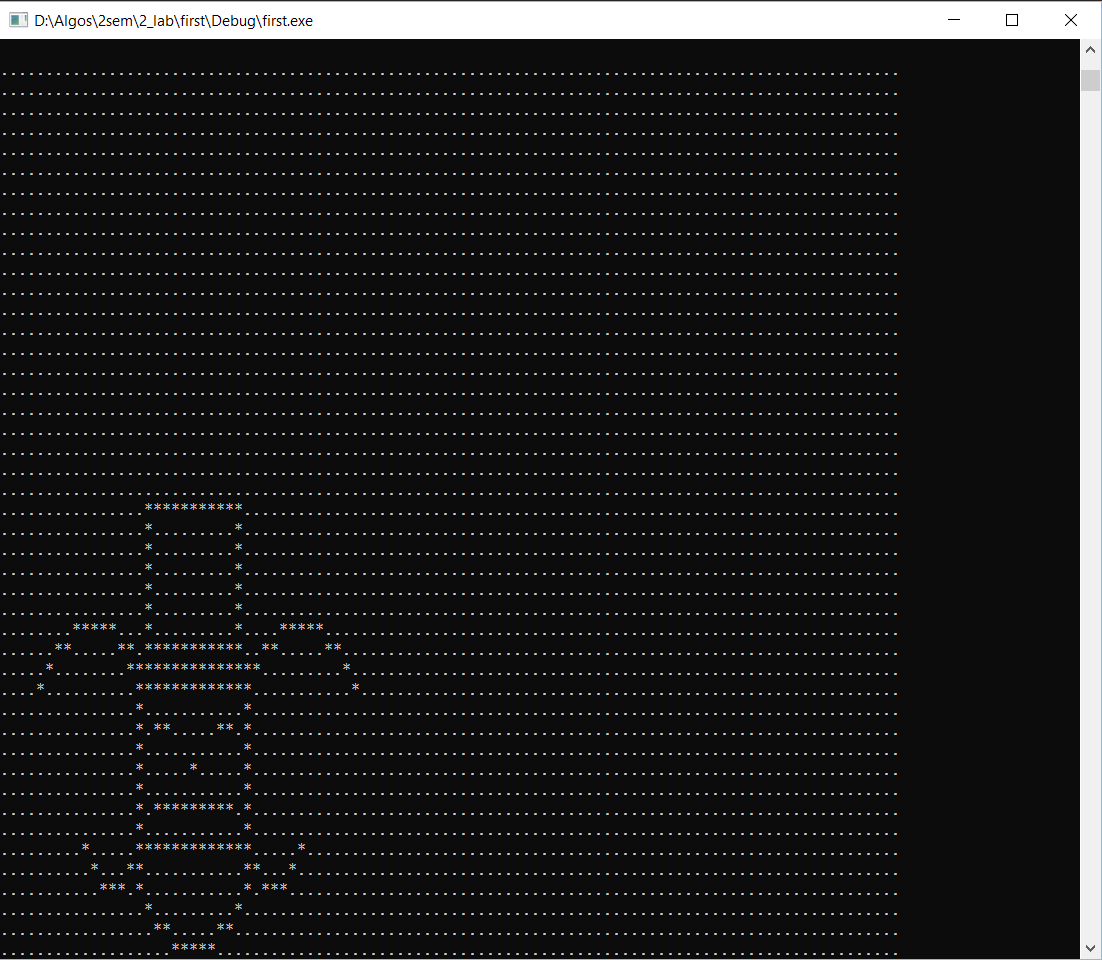
Исключение 4) – фигура задана неправильно.

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)  
   {  
    **try** {  
    **if** (!on\_screen(a, b)) **throw** ExceptionPoint();  
    **if** (!on\_screen(a, 0)) **throw** ExceptionX();  
    **if** (!on\_screen(0, b)) **throw** ExceptionY();  
    screen[b][a] = black;  
    }  
    **catch** (ExceptionX &a) {  
    std::cout << a.what() << std::endl;  
    }  
    **catch** (ExceptionY &a) {  
    std::cout << a.what() << std::endl;  
    }  
    **catch** (ExceptionPoint &a) {  
    std::cout << a.what() << std::endl;  
    }  
   }  
     
   **void** put\_line(**int** x0, **int** y0, **int** x1, **int** y1)  
   /\*  
   Рисование отрезка прямой от (x0,y0) до (x1,y1).  
   Уравнение прямой: b(x-x0) + a(y-y0) = 0.  
   Минимизируется величина abs(eps),  
   где eps = 2\*(b(x-x0)) + a(y-y0) (алгоритм Брезенхэма для прямой).  
   \*/  
   {  
    **try**{  
    **if** (!on\_screen(x0, y0) || !on\_screen(x1, y1)) **throw** ExceptionPoint();  
    **if** (!on\_screen(x0, 0) || !on\_screen(x1, 0)) **throw** ExceptionX();  
    **if** (!on\_screen(0, y0) || !on\_screen(0, y1)) **throw** ExceptionY();  
    **int** dx = 1;  
    **int** a = x1 - x0;  
    **if** (a < 0) dx = -1, a = -a;  
    **int** dy = 1;  
    **int** b = y1 - y0;  
    **if** (b < 0) dy = -1, b = -b;  
    **int** two\_a = 2 \* a;  
    **int** two\_b = 2 \* b;  
    **int** xcrit = -b + two\_a;  
    **int** eps = 0;  
     
    **for** (;;) { //Формирование прямой линии по точкам  
    put\_point(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;  
    }  
    }  
    **catch** (ExceptionPoint &a){  
    std::cout << "line: " << a.what() << std::endl;  
    }  
    **catch** (ExceptionX &a) {  
    std::cout << "line: " << a.what() << std::endl;  
    }  
    **catch** (ExceptionY &a) {  
    std::cout << "line: " << a.what() << std::endl;  
    }  
   }

**class** rectangle : **public** rotatable {  
 /\* nw ------ n ------ ne  
 | |  
 | |  
 w c e  
 | |  
 | |  
 sw ------ s ------ se \*/  
**protected**:  
 point sw, ne;  
**public**:  
 rectangle(point a, point b)   
 {  
 **try** {  
 **if** (a.x > b.x || a.y > b.y) **throw** ExceptionShape();  
 **if** (!on\_screen(a.x, a.y) || !on\_screen(b.x, b.y)) **throw** ExceptionPoint();  
 **if** (!on\_screen(a.x, 0) || !on\_screen(b.x, 0)) **throw** ExceptionX();  
 **if** (!on\_screen(0, a.y) || !on\_screen(0, b.y)) **throw** ExceptionY();  
 sw = a;  
 ne = b;  
 }  
 **catch** (ExceptionShape &buf) {  
 std::cout << "rectangle: " << buf.what() << std::endl;  
 }  
 **catch** (ExceptionPoint &buf) {  
 std::cout << "rectangle: " << buf.what() << std::endl;  
 }  
 **catch** (ExceptionX &buf) {  
 std::cout << "rectangle: " << buf.what() << std::endl;  
 }  
 **catch** (ExceptionY &buf) {  
 std::cout << "rectangle: " << buf.what() << std::endl;  
 }  
 }

**class** h\_circle : **public** rectangle, **public** reflectable{  
 **bool** reflected;  
**public**:  
 h\_circle(point a, point b, **bool** r = **true**): rectangle(a,b)  
 {  
 **try** {  
 **if** (a.x > b.x || a.y > b.y) **throw** ExceptionShape();  
 **if** (!on\_screen(a.x, a.y) || !on\_screen(b.x, b.y)) **throw** ExceptionPoint();  
 **if** (!on\_screen(a.x, 0) || !on\_screen(b.x, 0)) **throw** ExceptionX();  
 **if** (!on\_screen(0, a.y) || !on\_screen(0, b.y)) **throw** ExceptionY();  
 reflected = r;  
 }  
 **catch** (ExceptionShape &buf) {  
 std::cout << "h\_circle: " << buf.what() << std::endl;  
 }  
 **catch** (ExceptionPoint &buf) {  
 std::cout << "h\_circle: " << buf.what() << std::endl;  
 }  
 **catch** (ExceptionX &buf) {  
 std::cout << "h\_circle: " << buf.what() << std::endl;  
 }  
 **catch** (ExceptionY &buf) {  
 std::cout << "h\_circle: " << buf.what() << std::endl;  
 }  
 }

**void** h\_circle::draw() //Алгоритм Брезенхэма для окружностей  
{  
 **try** {  
 **int** x0 = (sw.x + ne.x) / 2;  
 **int** y0 = reflected ? sw.y : ne.y;  
 **int** radius = (ne.x - sw.x) / 2;  
 **int** x = 0;  
 **int** y = radius;  
 **int** delta = 2 - 2 \* radius;  
 **int** error = 0;  
 **while** (y >= 0) {  
 **if** (reflected) {   
 put\_point(x0 + x, y0 + y \* 0.7);  
 put\_point(x0 - x, y0 + y \* 0.7);   
 }  
 **else**{   
 put\_point(x0 + x, y0 - y \* 0.7);   
 put\_point(x0 - x, y0 - y \* 0.7);  
 }  
 error = 2 \* (delta + y) - 1;  
 **if** (delta < 0 && error <= 0) { ++x; delta += 2 \* x + 1; **continue**; }  
 error = 2 \* (delta - x) - 1;  
 **if** (delta > 0 && error > 0) { --y; delta += 1 - 2 \* y; **continue**; }  
 ++x; delta += 2 \* (x - y); --y;  
 }  
 }  
 **catch** (**char** &buf) {  
 std::cout << "h\_circle: " << buf << std::endl;  
 }  
}

# **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.