**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**ОТЧЕТ**

**по лабораторной работе № 1**

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

**Тема: «Работа с иерархией объектов: наследование и полиморфизм»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 3311 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Сапронов К. Д.  Землякова С. А. |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Преподаватель |  | Манирагена Валенс |

Санкт-Петербург

2025

**Текст индивидуального задания (вариант 40)**

Переработать программу работы с библиотекой фигур, дополнив её

механизмом контроля исключительных ситуаций. Например, возможно

выявление следующих ошибок:

— непопадание точки на экран;

— некорректные параметры при формировании фигуры;

— нехватка места на экране для размещения фигуры в одной из позиций

(исходной, повёрнутой, отражённой, перемещённой);

— повторный поворот/отражение уже повёрнутой/отражённой фигуры и др.;

— присоединение несимметричной фигуры к картинке неправильной

стороной.

1. Введенные классы исключений

В файле exceptions.h реализована иерархия классов исключений:

class ShapeError : public std::exception { /\*...\*/ };

Конкретные типы исключений:

class PointOutOfScreen : public ShapeError { /\*...\*/ }; // Точка вне экрана

class InvalidShapeParameters : public ShapeError { /\*...\*/ }; // Некорректные параметры

class NotEnoughSpace : public ShapeError { /\*...\*/ }; // Нехватка места

class AlreadyTransformed : public ShapeError { /\*...\*/ }; // Повторная трансформация

class AttachmentError : public ShapeError { /\*...\*/ }; // Ошибка присоединения

2. Реализованные проверки и места генерации исключений

2.1. Базовые проверки (в shape2.cpp):

* Проверка координат точек:

void put\_point(int a, int b) {

if (!on\_screen(a, b))

throw PointOutOfScreen(a, b);

// ...

}

2.2. Проверки в конструкторах фигур:

* Для линии:

line::line(point a, int L) {

if (!on\_screen(w.x, w.y) || !on\_screen(e.x, e.y))

throw NotEnoughSpace("line");

if (L <= 0)

throw InvalidShapeParameters("line length must be positive");

}

2.3. Проверки трансформаций:

* Повторный поворот:

void rotatable::rotate\_left() {

if (state == rotated::left)

throw AlreadyTransformed("left rotation");

// ...

}

3. Демонстрация обработки исключений (в main.cpp)

3.1. Пример перехвата ошибки создания фигуры:

try {

rectangle bad\_rect(point(-10,-10), point(300,300));

}

catch (const NotEnoughSpace& e) {

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

// Можно подменить фигуру на "знак ошибки"

}

3.2. Пример перехвата ошибки трансформации:

try {

hat.rotate\_left();

hat.rotate\_left(); // Повторный поворот

}

catch (const AlreadyTransformed& e) {

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

}

4. Архитектурные решения

4.1. Расположение throw:

* В конструкторах фигур
* В методах трансформации
* В функциях отрисовки (put\_point, put\_line)

4.2. Стратегия обработки:

* Глобальный перехват в shape\_refresh():

void shape\_refresh() {

try {

// Отрисовка всех фигур

}

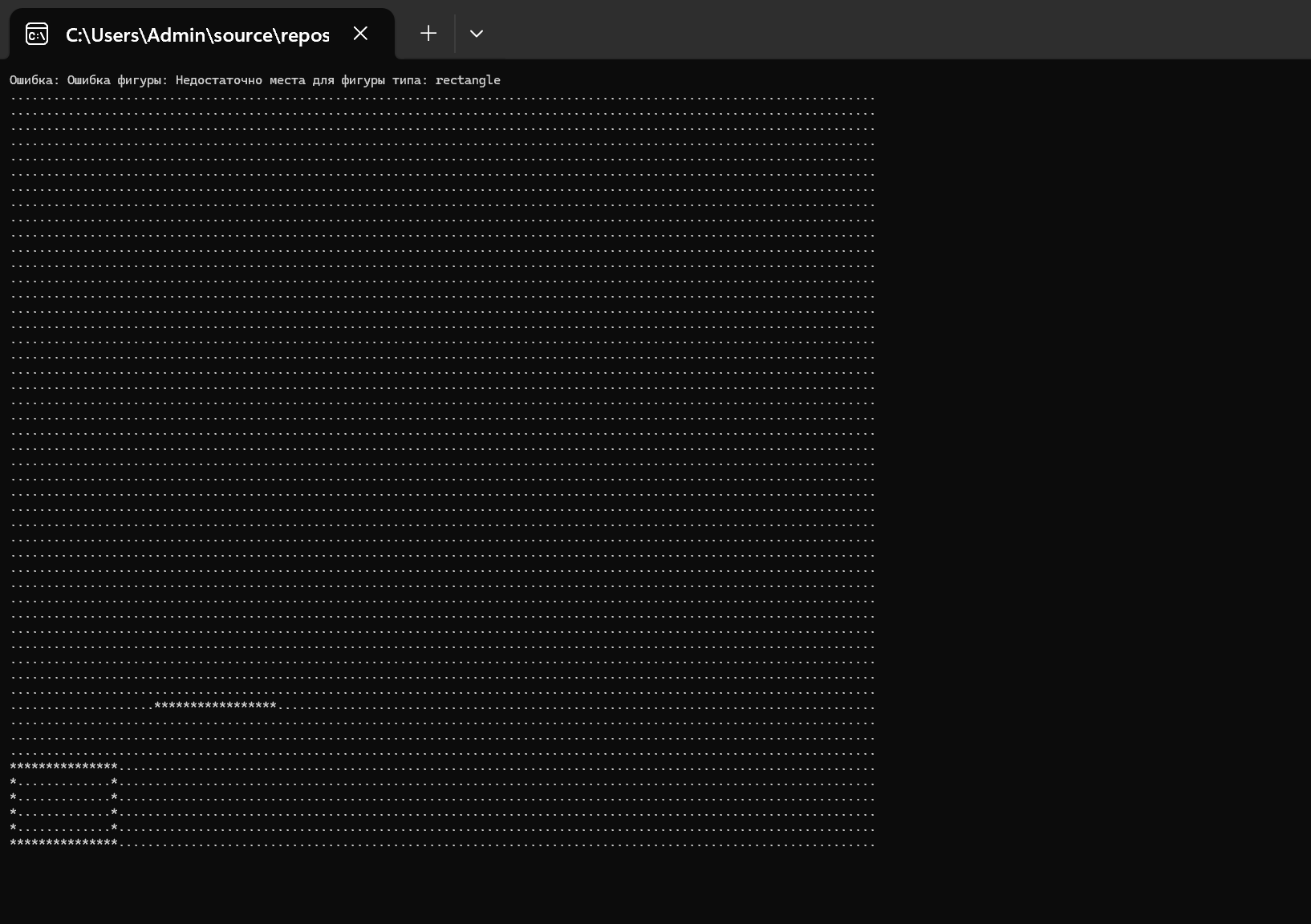
catch (const ShapeError& e) {

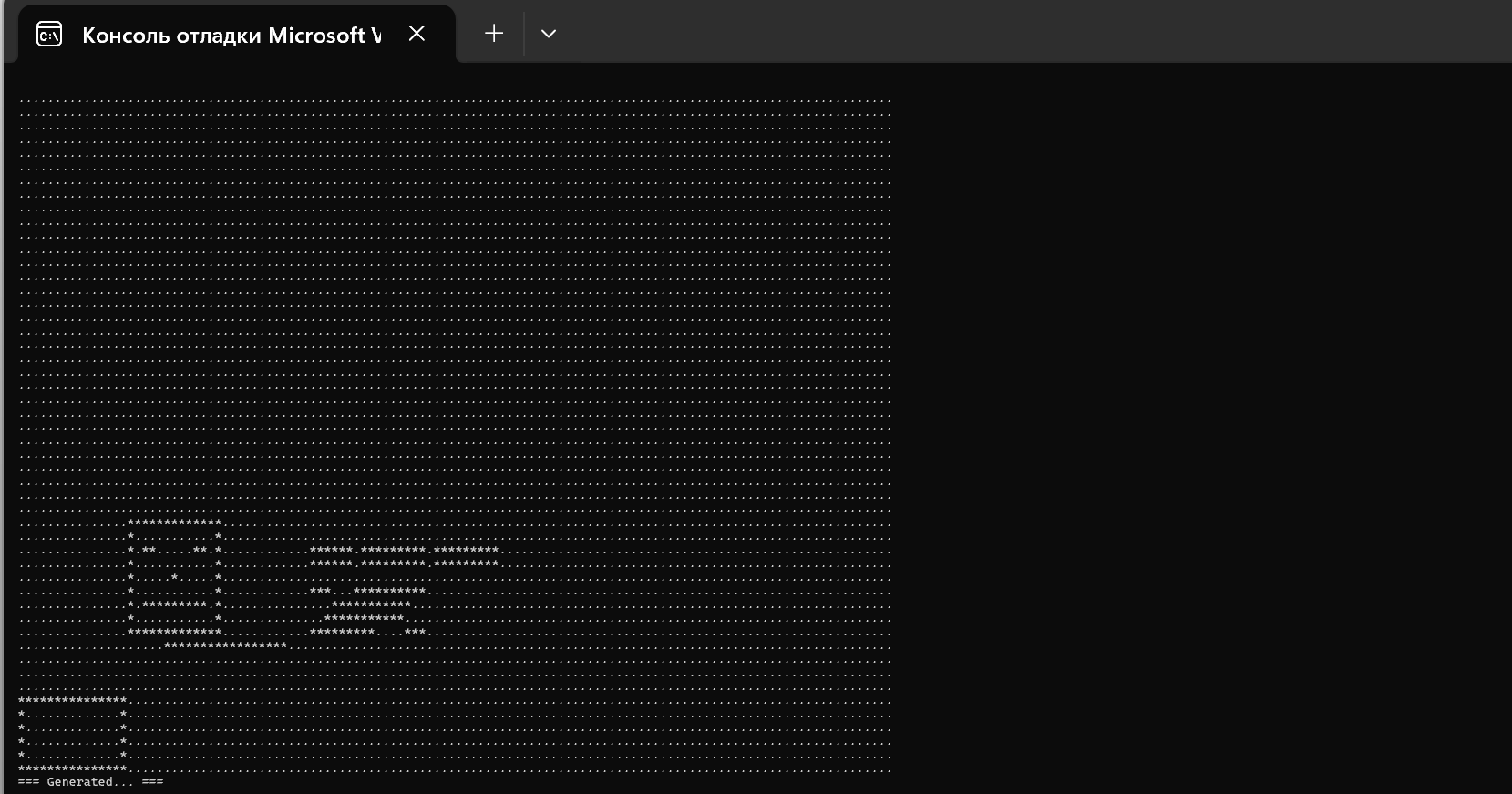
std::cerr << "Drawing error: " << e.what();

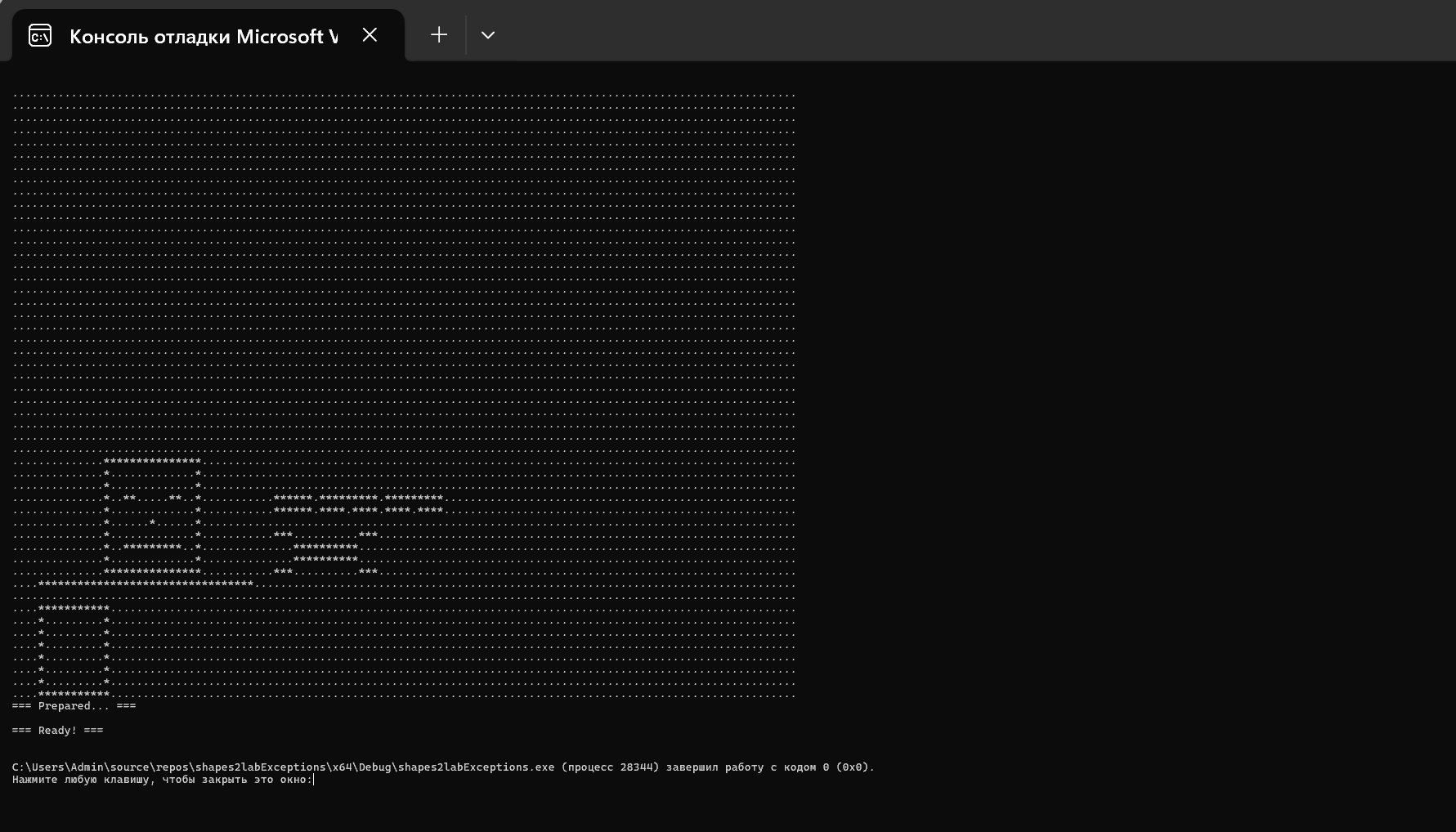
}

}

Контрольные примеры.  
1. Обработка ошибки «Недостаточно места на экране», вызванная вручную специальной фигурой bad\_rect()

****

****

****

Из-за искусственно вызванной ошибки происходит подготовка и размещение фигур, но сборка фигуры не производится.

**Выводы**

1. Реализована полная система обработки исключений для всех критических операций
2. Обеспечена минимальная деградация изображения при ошибках
3. Демонстрация перехвата выполнена для разных типов ошибок:
   * Простая (PointOutOfScreen)
   * Комплексная (AlreadyTransformed с сохранением состояния)
4. Все исключения корректно выводят понятные сообщения об ошибках

Контрольные вопросы

1. Что такое исключительная ситуация при выполнении программы?

Ошибка или исчерпание какого-то ресурса, например, доступной памяти или времени ожидания сигнала, или даже один из предусмотренных вариантов завершения процесса

1. Как можно выявить исключительную ситуацию?

С помощью try-catch блоков ограничить возникновение исключительной ситуации и предпринять способы её обработки

1. Что можно предпринять при выявлении исключительной ситуации?

Создаем новый класс-обработчик ситуации под новую исключительную ситуацию.

1. Можно ли передать в обработчик особых ситуаций какую-либо информацию о произошедшем событии?

Да

1. Можно ли обработать неизвестную особую ситуацию?

Да, можно обработать неизвестные ошибки через catch(...)

1. Можно ли сделать обработчик ситуации пустым?

Можно, но это не повлияет на работу программы дальше

1. Что можно предпринять, если для корректной обработки ситуации в данном месте программы у обработчика недостаточно данных?

Сохранить данные об ошибке и разобраться почему не хватает данных

1. Если требуется несколько обработчиков особых ситуаций, в каком порядке следует их размещать в программе?

От более узконаправленных к общим

1. Можно ли перехватывать одним обработчиком несколько различных особых ситуаций?

Да

1. Как действуют обработчики в случае, когда никакой особой ситуации не произошло?

Не выполняются

1. Как следует размещать блоки контроля, чтобы получить безопасный программный код?

Как можно ближе к части, где может возникнуть исключительная ситуация

1. Можно ли продолжить выполнение программы с точки, в которой выявлена ошибка, после внесения исправлений в данные?

Да

Список литературы:

* П. Г. КОЛИНЬКО «ПОЛЬЗОВАТЕЛЬСКИЕ КОНТЕЙНЕРЫ» Учебно-методическое пособие
* Курс лекций <https://vec.etu.ru/moodle/course/view.php?id=22325>