**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. 3311 | Пасечный Л.В.  Загуменнов И.М. |  |
| Преподаватель | Манирагена В. |  |

Санкт-Петербург

2024

**Введение**

**Цель работы:**

Переработать программу работы с библиотекой фигур, дополнив её меха-низмом 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.

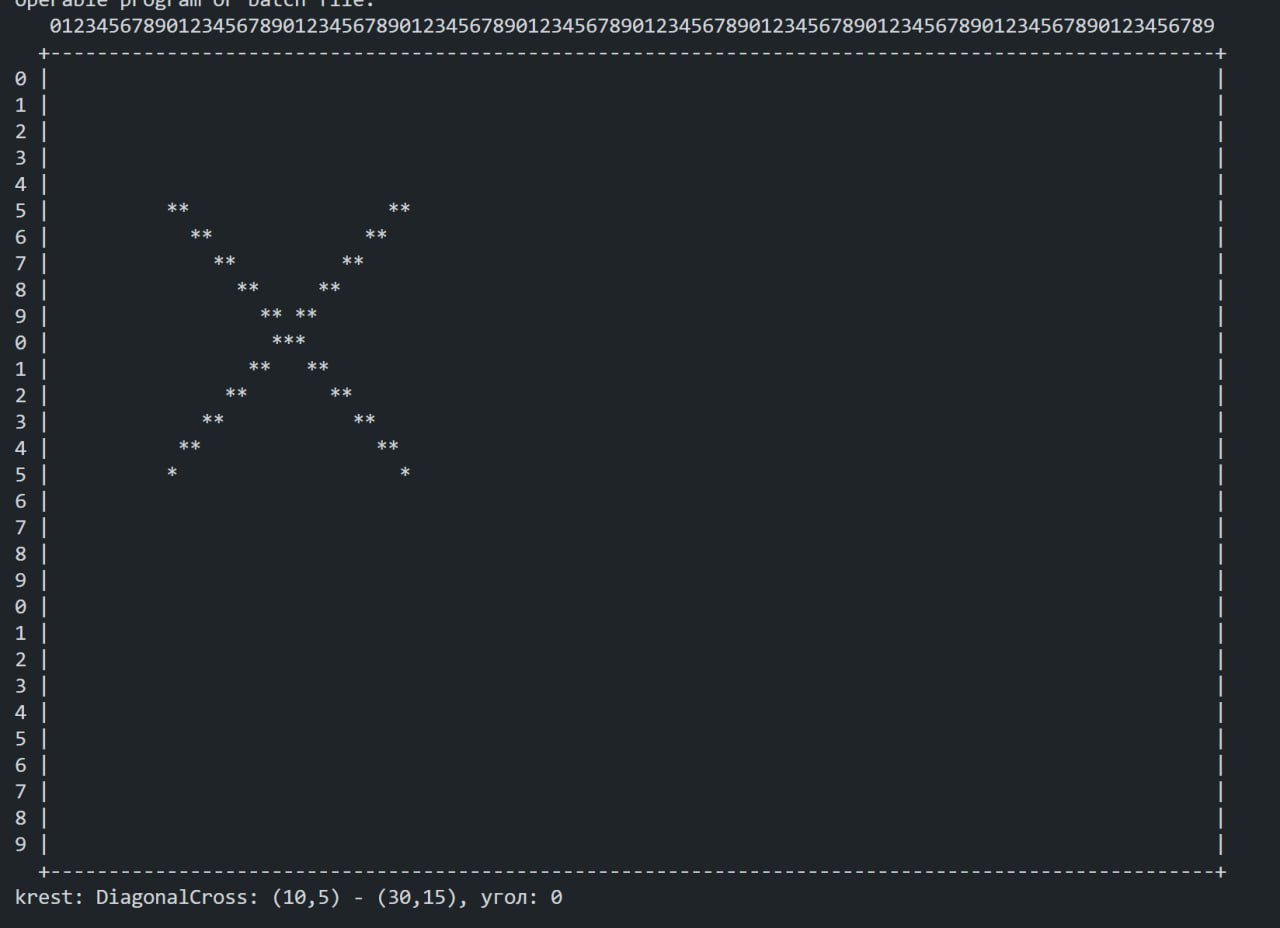
Нужно реализовать генерацию выявление всех возможных ошибок в до-бавляемой на рисунок фигуре и продемонстрировать перехват не менее двух типов ошибок разного уровня сложности.

Если исключение генерируется в конструкторе фигуры, следует обеспе-чить исключение фигуры из списка для рисования или подмену её запасной фигурой — знаком ошибки.

Перехват исключения в той же функции, в которой оно возбуждено, не применяется. В этом случае, когда ошибку можно обработать в точке обнару-жения, механизм исключений избыточен.

Организовать перехват исключений следует таким образом, чтобы иска-жения итоговой картинки были минимальны. Протестировать исключитель-ные ситуации, результаты эксперимента поместить в отчёт.

**Контрольные примеры:**

****

**Контрольные вопросы**

1. **Что такое исключительная ситуация при выполнении программы?**  
   Исключительная ситуация — это ошибка или нестандартное событие, возникающее во время выполнения программы, которое нарушает её обычный поток. В C++ такие ситуации обрабатываются с помощью механизма исключений (try, catch, throw).
2. **Как можно выявить исключительную ситуацию?**  
   Исключение выявляется автоматически при возникновении ошибки (например, деление на ноль, нехватка памяти) или вручную, когда программист вызывает оператор throw. Обнаружение может происходить в специально контролируемом блоке try.
3. **Что можно предпринять при выявлении исключительной ситуации?**  
   При выявлении исключения управление передаётся в блок catch, где можно предпринять действия по обработке ошибки, например, освободить ресурсы или вывести сообщение пользователю. Это помогает избежать аварийного завершения программы.
4. **Можно ли передать в обработчик особых ситуаций какую-либо информацию о произошедшем событии?**  
   Да, при генерации исключения можно передать объект или значение, содержащие информацию об ошибке. Этот объект принимается обработчиком catch и может быть использован для анализа ситуации.
5. **Можно ли обработать неизвестную особую ситуацию?**  
   Да, для этого существует универсальный обработчик catch(...), который перехватывает все исключения, независимо от их типа. Однако он не предоставляет подробной информации о самой ошибке.
6. **Можно ли сделать обработчик ситуации пустым?**  
   Да, можно создать пустой блок catch, если требуется просто подавить исключение. Однако это не рекомендуется, так как может скрыть важную ошибку.
7. **Что можно предпринять, если для корректной обработки ситуации в данном месте программы у обработчика недостаточно данных?**  
   В таком случае можно пробросить исключение дальше с помощью throw; или вызвать другую функцию, обладающую необходимыми данными. Это позволяет централизовать обработку ошибок.
8. **Если требуется несколько обработчиков особых ситуаций, в каком порядке следует их размещать в программе?**  
   Обработчики следует располагать от более специфичных типов исключений к более общим. Это гарантирует, что конкретные исключения будут перехвачены раньше универсальных.
9. **Можно ли перехватывать одним обработчиком несколько различных особых ситуаций?**  
   Да, один блок catch может перехватывать базовый тип, общий для нескольких исключений, или использовать альтернативы, если применимы. Это позволяет сократить дублирование кода.
10. **Как действуют обработчики в случае, когда никакой особой ситуации не произошло?**  
    Если исключение не произошло, блок catch просто игнорируется, и выполнение продолжается после блока try. Таким образом, программа работает как обычно.
11. **Как следует размещать блоки контроля, чтобы получить безопасный программный код?**  
    Блоки try следует использовать вокруг потенциально опасных операций, таких как работа с файлами, памятью или внешними устройствами. Это позволяет локализовать обработку ошибок и сохранить стабильность программы.
12. **Можно ли продолжить выполнение программы с точки, в которой выявлена ошибка, после внесения исправлений в данные?**  
    В большинстве случаев после обработки исключения выполнение продолжается с первой инструкции после блока try, а не с точки ошибки. Для продолжения с того же места нужна специальная логика, так как стандартный механизм исключений этого не поддерживает.

**Заключение**

Во второй лабораторной работе была реализована и протестирована система обработки исключительных ситуаций при работе с графической библиотекой фигур. Основное внимание уделено выявлению и корректной обработке возможных ошибок, возникающих в процессе создания, трансформации и отображения фигур.

**Реализованные проверки:**

1. Контроль за выходом координат фигур за пределы холста. Исключения выбрасываются при попытке создать или переместить фигуру вне допустимых границ.
2. Проверка корректности параметров при создании фигур (например, правильность порядка координат у прямоугольников).
3. Контроль невозможных трансформаций, таких как поворот, ведущий к выходу за экран.
4. Фиксация повторных трансформаций, которые могут привести к некорректному состоянию фигуры.
5. Обработка попыток использования несимметричных фигур в неподходящих позициях (опционально).

**Перехваченные исключения:**

* Простая ошибка: добавление фигуры с уже существующим именем.
* Ошибка более высокого уровня: поворот фигуры с выходом за пределы экрана.

**Особенности реализации:**

* Исключения не перехватываются в точке возникновения, что соответствует требованиям задания. Обработка производится на более высоком уровне.
* При возникновении исключения в конструкторе фигура не попадает в список для рисования или заменяется на специальную резервную фигуру.
* Программа остается работоспособной и визуально корректной даже при наличии ошибок — исключения не мешают формированию итогового изображения.

**Результат:**

В результате переработки код стал устойчивым к ошибкам и позволяет выявлять и локализовывать некорректные ситуации без полного завершения программы. Обработка исключений внедрена грамотно и системно, с минимальным искажением общей композиции. Работа продемонстрировала применение механизма исключений для повышения надежности и удобства в графических приложениях.

**Источники:**

- Учебно-методическое пособие Колинько П. Г. – Последнее посещение 9.04 18:20  
- <https://habr.com/ru/articles/445948/> - Последнее посещение 9.04 18:20  
- <https://habr.com/ru/articles/690038/> - Последнее посещение 7.04 10:05

- <https://ru.stackoverflow.com/questions/328011/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0-%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9> - Последнее посещение 8.04 13:55

**Код программы**

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <vector>

#include <cstdlib>

#include <exception>

#include <string>

#include <algorithm>

#include <cmath>

#include <sstream>

#include <map>

constexpr int WIDTH = 100;

constexpr int HEIGHT = 30;

// Интерфейс для фигур, поддерживающих поворот (метод изменяет состояние фигуры)

class IRotatable {

public:

// Поворот фигуры на угол angle вокруг центра фигуры (накопительный поворот)

virtual void rotate(double angle) = 0;

virtual ~IRotatable() {}

};

struct BoundingRect {

int x1, y1, x2, y2;

};

// Класс исключения для фигур

class ShapeException : public std::exception {

private:

std::string message;

public:

explicit ShapeException(const std::string &msg) : message(msg) {}

const char\* what() const noexcept override {

return message.c\_str();

}

};

// Функция поворота точки вокруг опорной точки (pivotX, pivotY)

std::pair<int, int> rotatePoint(int x, int y, int pivotX, int pivotY, double angle) {

double rad = angle \* M\_PI / 180.0;

double cosA = std::cos(rad);

double sinA = std::sin(rad);

int newX = pivotX + std::round((x - pivotX) \* cosA - (y - pivotY) \* sinA);

int newY = pivotY + std::round((x - pivotX) \* sinA + (y - pivotY) \* cosA);

return {newX, newY};

}

// Предварительное объявление класса Canvas

class Canvas;

// Базовый класс для фигур с уникальным именем и операциями поворота и трансляции

class Shape {

protected:

std::string name;

public:

Shape(const std::string& name) : name(name) {}

virtual void draw(Canvas &canvas) const = 0;

virtual void translate(int dx, int dy) = 0;

std::string getName() const { return name; }

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

virtual BoundingRect getBoundingRect() const = 0;

virtual ~Shape() = default;

};

// Класс холста для текстовой графики и хранения фигур

class Canvas {

private:

std::vector<std::vector<char>> buffer;

std::map<std::string, Shape\*> shapes;

public:

Canvas() : buffer(HEIGHT, std::vector<char>(WIDTH, ' ')) {}

~Canvas() {

clearShapes();

}

void clearBuffer() {

for (auto &row : buffer)

std::fill(row.begin(), row.end(), ' ');

}

void setPixel(int x, int y, char ch) {

if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT)

buffer[y][x] = ch;

}

void display() const {

std::cout << " ";

for (int x = 0; x < WIDTH; ++x)

std::cout << (x % 10);

std::cout << "\n +" << std::string(WIDTH, '-') << "+\n";

for (int y = 0; y < HEIGHT; ++y) {

std::cout << (y % 10) << " |";

for (char ch : buffer[y])

std::cout << ch;

std::cout << "|\n";

}

std::cout << " +" << std::string(WIDTH, '-') << "+\n";

}

void listShapes() const {

for (const auto &pair : shapes) {

std::cout << pair.first << ": " << pair.second->getInfo() << "\n";

}

}

// Отрисовка всех фигур: сначала очищается буфер, затем каждая фигура отрисовывается по своим координатам

void render() {

clearBuffer();

for (auto &pair : shapes) {

pair.second->draw(\*this);

}

display();

}

// Добавление фигуры (если фигура с таким именем уже существует, генерируется исключение)

void addShape(Shape\* shape) {

if (shapes.find(shape->getName()) != shapes.end())

throw ShapeException("Фигура с таким именем уже существует.");

shapes[shape->getName()] = shape;

}

// Удаление всех фигур с холста

void clearShapes() {

for (auto &pair : shapes)

delete pair.second;

shapes.clear();

}

// Удаление фигуры по имени

void removeShape(const std::string &name) {

auto it = shapes.find(name);

if (it == shapes.end())

throw ShapeException("Фигура с таким именем не найдена.");

delete it->second;

shapes.erase(it);

}

// Получение фигуры по имени

Shape\* getShape(const std::string &name) const {

auto it = shapes.find(name);

if (it == shapes.end())

throw ShapeException("Фигура с таким именем не найдена.");

return it->second;

}

};

// ---------------------------

// Реализация фигуры "Линия"

// ---------------------------

class Line : public Shape, public IRotatable {

public:

int x1, y1, x2, y2;

double angle; // накопленный угол поворота

// Конструктор принимает имя, координаты и опциональный угол поворота

Line(const std::string& name, int x1, int y1, int x2, int y2, double angle = 0)

: Shape(name), x1(x1), y1(y1), x2(x2), y2(y2), angle(0) // инициализируем angle нулём

{

if (x1 < 0 || x1 >= WIDTH || y1 < 0 || y1 >= HEIGHT ||

x2 < 0 || x2 >= WIDTH || y2 < 0 || y2 >= HEIGHT)

throw ShapeException("Line: точки вне границ холста.");

if (angle != 0) {

// Поворот по центру линии

int pivotX = (x1 + x2) / 2, pivotY = (y1 + y2) / 2;

auto np1 = rotatePoint(x1, y1, pivotX, pivotY, angle);

auto np2 = rotatePoint(x2, y2, pivotX, pivotY, angle);

x1 = np1.first; y1 = np1.second;

x2 = np2.first; y2 = np2.second;

this->angle = angle; // сохраняем поворот

}

}

std::string getInfo() const override {

std::ostringstream oss;

oss << "Line: (" << x1 << "," << y1 << ") - ("

<< x2 << "," << y2 << "), угол: " << angle;

return oss.str();

}

void draw(Canvas &canvas) const override {

int dx = std::abs(x2 - x1), dy = std::abs(y2 - y1);

int sx = (x1 < x2) ? 1 : -1, sy = (y1 < y2) ? 1 : -1;

int err = dx - dy, e2, x = x1, y = y1;

while (true) {

canvas.setPixel(x, y, '\*');

if (x == x2 && y == y2)

break;

e2 = 2 \* err;

if (e2 > -dy) { err -= dy; x += sx; }

if (e2 < dx) { err += dx; y += sy; }

}

}

BoundingRect getBoundingRect() const override {

int minX = std::min(x1, x2);

int minY = std::min(y1, y2);

int maxX = std::max(x1, x2);

int maxY = std::max(y1, y2);

return { minX, minY, maxX, maxY };

}

// Вращение линии вокруг её центра (in-place)

void rotate(double deltaAngle) override {

int pivotX = (x1 + x2) / 2;

int pivotY = (y1 + y2) / 2;

auto np1 = rotatePoint(x1, y1, pivotX, pivotY, deltaAngle);

auto np2 = rotatePoint(x2, y2, pivotX, pivotY, deltaAngle);

if (np1.first < 0 || np1.first >= WIDTH || np1.second < 0 || np1.second >= HEIGHT ||

np2.first < 0 || np2.first >= WIDTH || np2.second < 0 || np2.second >= HEIGHT)

throw ShapeException("Line: поворот приводит к выходу за границы холста.");

x1 = np1.first;

y1 = np1.second;

x2 = np2.first;

y2 = np2.second;

angle += deltaAngle;

}

// Перемещение линии на dx, dy (in-place)

void translate(int dx, int dy) override {

int newX1 = x1 + dx, newY1 = y1 + dy, newX2 = x2 + dx, newY2 = y2 + dy;

if (newX1 < 0 || newX1 >= WIDTH || newY1 < 0 || newY1 >= HEIGHT ||

newX2 < 0 || newX2 >= WIDTH || newY2 < 0 || newY2 >= HEIGHT)

throw ShapeException("Line: перевод приводит к выходу за границы холста.");

x1 = newX1; y1 = newY1; x2 = newX2; y2 = newY2;

}

};

// ------------------------------

// Реализация фигуры "Прямоугольник"

// ------------------------------

class Rectangle : public Shape, public IRotatable {

private:

int orig\_x1, orig\_y1, orig\_x2, orig\_y2;

double angle; // накопленный угол поворота

public:

Rectangle(const std::string& name, int x1, int y1, int x2, int y2, double angle = 0)

: Shape(name), orig\_x1(x1), orig\_y1(y1), orig\_x2(x2), orig\_y2(y2), angle(angle)

{

if (x1 < 0 || x1 >= WIDTH || y1 < 0 || y1 >= HEIGHT ||

x2 < 0 || x2 >= WIDTH || y2 < 0 || y2 >= HEIGHT)

throw ShapeException("Rectangle: вершины вне холста.");

if (x1 >= x2 || y1 >= y2)

throw ShapeException("Rectangle: некорректные параметры.");

}

std::string getInfo() const override {

std::ostringstream oss;

oss << "Rectangle: (" << orig\_x1 << "," << orig\_y1 << ") - ("

<< orig\_x2 << "," << orig\_y2 << "), угол: " << angle;

return oss.str();

}

void draw(Canvas &canvas) const override {

std::vector<std::pair<int, int>> verts = { {orig\_x1, orig\_y1}, {orig\_x2, orig\_y1}, {orig\_x2, orig\_y2}, {orig\_x1, orig\_y2} };

int centerX = (orig\_x1 + orig\_x2) / 2;

int centerY = (orig\_y1 + orig\_y2) / 2;

if (angle != 0) {

for (auto &p : verts)

p = rotatePoint(p.first, p.second, centerX, centerY, angle);

}

for (size\_t i = 0; i < verts.size(); ++i) {

auto [x1, y1] = verts[i];

auto [x2, y2] = verts[(i + 1) % verts.size()];

Line("temp", x1, y1, x2, y2).draw(canvas);

}

}

// Поворот прямоугольника (in-place)

void rotate(double deltaAngle) override {

double newAngle = angle + deltaAngle;

std::vector<std::pair<int, int>> verts = { {orig\_x1, orig\_y1}, {orig\_x2, orig\_y1}, {orig\_x2, orig\_y2}, {orig\_x1, orig\_y2} };

int centerX = (orig\_x1 + orig\_x2) / 2;

int centerY = (orig\_y1 + orig\_y2) / 2;

if (newAngle != 0) {

for (auto &p : verts)

p = rotatePoint(p.first, p.second, centerX, centerY, newAngle);

}

int minX = verts[0].first, maxX = verts[0].first, minY = verts[0].second, maxY = verts[0].second;

for (const auto &p : verts) {

minX = std::min(minX, p.first);

maxX = std::max(maxX, p.first);

minY = std::min(minY, p.second);

maxY = std::max(maxY, p.second);

}

if(minX < 0 || minY < 0 || maxX >= WIDTH || maxY >= HEIGHT)

throw ShapeException("Rectangle: поворот приводит к выходу за границы холста.");

angle = newAngle;

}

// Перемещение прямоугольника (in-place)

void translate(int dx, int dy) override {

int new\_x1 = orig\_x1 + dx, new\_y1 = orig\_y1 + dy, new\_x2 = orig\_x2 + dx, new\_y2 = orig\_y2 + dy;

std::vector<std::pair<int, int>> verts = { {new\_x1, new\_y1}, {new\_x2, new\_y1}, {new\_x2, new\_y2}, {new\_x1, new\_y2} };

int centerX = (new\_x1 + new\_x2) / 2;

int centerY = (new\_y1 + new\_y2) / 2;

if (angle != 0) {

for (auto &p : verts)

p = rotatePoint(p.first, p.second, centerX, centerY, angle);

}

int minX = verts[0].first, maxX = verts[0].first, minY = verts[0].second, maxY = verts[0].second;

for (const auto &p : verts) {

minX = std::min(minX, p.first);

maxX = std::max(maxX, p.first);

minY = std::min(minY, p.second);

maxY = std::max(maxY, p.second);

}

if(minX < 0 || minY < 0 || maxX >= WIDTH || maxY >= HEIGHT)

throw ShapeException("Rectangle: перевод приводит к выходу за границы холста.");

orig\_x1 = new\_x1; orig\_y1 = new\_y1; orig\_x2 = new\_x2; orig\_y2 = new\_y2;

}

BoundingRect getBoundingRect() const override {

std::vector<std::pair<int, int>> verts = { {orig\_x1, orig\_y1}, {orig\_x2, orig\_y1},

{orig\_x2, orig\_y2}, {orig\_x1, orig\_y2} };

int centerX = (orig\_x1 + orig\_x2) / 2;

int centerY = (orig\_y1 + orig\_y2) / 2;

if (angle != 0) {

for (auto &p : verts)

p = rotatePoint(p.first, p.second, centerX, centerY, angle);

}

int minX = verts[0].first, minY = verts[0].second;

int maxX = verts[0].first, maxY = verts[0].second;

for (const auto &p : verts) {

minX = std::min(minX, p.first);

minY = std::min(minY, p.second);

maxX = std::max(maxX, p.first);

maxY = std::max(maxY, p.second);

}

return { minX, minY, maxX, maxY };

}

};

// -----------------------------

// Реализация фигуры "Диагональный крест"

// -----------------------------

class DiagonalCross : public Shape, public IRotatable {

private:

int orig\_x1, orig\_y1, orig\_x2, orig\_y2;

double angle;

public:

DiagonalCross(const std::string& name, int x1, int y1, int x2, int y2, double angle = 0)

: Shape(name), orig\_x1(x1), orig\_y1(y1), orig\_x2(x2), orig\_y2(y2), angle(angle)

{

if (x1 < 0 || x1 >= WIDTH || y1 < 0 || y1 >= HEIGHT ||

x2 < 0 || x2 >= WIDTH || y2 < 0 || y2 >= HEIGHT)

throw ShapeException("DiagonalCross: точки вне холста.");

}

std::string getInfo() const override {

std::ostringstream oss;

oss << "DiagonalCross: (" << orig\_x1 << "," << orig\_y1 << ") - ("

<< orig\_x2 << "," << orig\_y2 << "), угол: " << angle;

return oss.str();

}

void draw(Canvas &canvas) const override {

std::vector<std::pair<int, int>> verts = { {orig\_x1, orig\_y1}, {orig\_x2, orig\_y1}, {orig\_x2, orig\_y2}, {orig\_x1, orig\_y2} };

int centerX = (orig\_x1 + orig\_x2) / 2;

int centerY = (orig\_y1 + orig\_y2) / 2;

if (angle != 0) {

for (auto &p : verts)

p = rotatePoint(p.first, p.second, centerX, centerY, angle);

}

Line("temp", verts[0].first, verts[0].second, verts[2].first, verts[2].second).draw(canvas);

Line("temp", verts[1].first, verts[1].second, verts[3].first, verts[3].second).draw(canvas);

}

// Вращение креста (in-place)

void rotate(double deltaAngle) override {

double newAngle = angle + deltaAngle;

std::vector<std::pair<int, int>> verts = { {orig\_x1, orig\_y1}, {orig\_x2, orig\_y1}, {orig\_x2, orig\_y2}, {orig\_x1, orig\_y2} };

int centerX = (orig\_x1 + orig\_x2) / 2;

int centerY = (orig\_y1 + orig\_y2) / 2;

if (newAngle != 0) {

for (auto &p : verts)

p = rotatePoint(p.first, p.second, centerX, centerY, newAngle);

}

int minX = verts[0].first, maxX = verts[0].first, minY = verts[0].second, maxY = verts[0].second;

for (const auto &p : verts) {

minX = std::min(minX, p.first);

maxX = std::max(maxX, p.first);

minY = std::min(minY, p.second);

maxY = std::max(maxY, p.second);

}

if(minX < 0 || minY < 0 || maxX >= WIDTH || maxY >= HEIGHT)

throw ShapeException("DiagonalCross: поворот приводит к выходу за границы холста.");

angle = newAngle;

}

// Перемещение креста (in-place)

void translate(int dx, int dy) override {

int new\_x1 = orig\_x1 + dx, new\_y1 = orig\_y1 + dy, new\_x2 = orig\_x2 + dx, new\_y2 = orig\_y2 + dy;

std::vector<std::pair<int, int>> verts = { {new\_x1, new\_y1}, {new\_x2, new\_y1}, {new\_x2, new\_y2}, {new\_x1, new\_y2} };

int centerX = (new\_x1 + new\_x2) / 2;

int centerY = (new\_y1 + new\_y2) / 2;

if (angle != 0) {

for (auto &p : verts)

p = rotatePoint(p.first, p.second, centerX, centerY, angle);

}

int minX = verts[0].first, maxX = verts[0].first, minY = verts[0].second, maxY = verts[0].second;

for (const auto &p : verts) {

minX = std::min(minX, p.first);

maxX = std::max(maxX, p.first);

minY = std::min(minY, p.second);

maxY = std::max(maxY, p.second);

}

if(minX < 0 || minY < 0 || maxX >= WIDTH || maxY >= HEIGHT)

throw ShapeException("DiagonalCross: перевод приводит к выходу за границы холста.");

orig\_x1 = new\_x1; orig\_y1 = new\_y1; orig\_x2 = new\_x2; orig\_y2 = new\_y2;

}

BoundingRect getBoundingRect() const override {

std::vector<std::pair<int, int>> verts = { {orig\_x1, orig\_y1}, {orig\_x2, orig\_y1},

{orig\_x2, orig\_y2}, {orig\_x1, orig\_y2} };

int centerX = (orig\_x1 + orig\_x2) / 2;

int centerY = (orig\_y1 + orig\_y2) / 2;

if (angle != 0) {

for (auto &p : verts)

p = rotatePoint(p.first, p.second, centerX, centerY, angle);

}

int minX = verts[0].first, minY = verts[0].second;

int maxX = verts[0].first, maxY = verts[0].second;

for (const auto &p : verts) {

minX = std::min(minX, p.first);

minY = std::min(minY, p.second);

maxX = std::max(maxX, p.first);

maxY = std::max(maxY, p.second);

}

return { minX, minY, maxX, maxY };

}

};

// -----------------------------

// Реализация фигуры "Голова"

// -----------------------------

class Head : public Shape {

private:

int orig\_x1, orig\_y1, orig\_x2, orig\_y2;

public:

Head(const std::string& name, int x1, int y1, int x2, int y2)

: Shape(name), orig\_x1(x1), orig\_y1(y1), orig\_x2(x2), orig\_y2(y2)

{

if (x1 < 0 || x1 >= WIDTH || y1 < 0 || y1 >= HEIGHT ||

x2 < 0 || x2 >= WIDTH || y2 < 0 || y2 >= HEIGHT)

throw ShapeException("Head: координаты головы вне холста.");

if (x1 >= x2 || y1 >= y2)

throw ShapeException("Head: некорректные параметры.");

}

std::string getInfo() const override {

std::ostringstream oss;

oss << "Head: (" << orig\_x1 << "," << orig\_y1 << ") - ("

<< orig\_x2 << "," << orig\_y2 << "), угол: 0";

return oss.str();

}

void draw(Canvas &canvas) const override {

int x1 = orig\_x1, y1 = orig\_y1, x2 = orig\_x2, y2 = orig\_y2;

int total\_height = y2 - y1;

if (total\_height < 6)

throw ShapeException("Head: недостаточная высота.");

int head\_width = x2 - x1;

int hat\_height = total\_height / 3;

int head\_height = total\_height - hat\_height;

if (hat\_height < 1 || head\_height < 1)

throw ShapeException("Head: слишком мала высота.");

int hat\_width = head\_width / 2;

int hat\_x1 = x1 + hat\_width / 2;

int hat\_y1 = y1;

int hat\_x2 = x2 - hat\_width / 2;

int hat\_y2 = y1 + hat\_height - 1;

int head\_x1 = x1;

int head\_y1 = hat\_y2 + 1;

int head\_x2 = x2;

int head\_y2 = y2;

int eye\_y = head\_y1 + (head\_height / 3);

int eye\_width = static\_cast<int>(head\_width \* 0.2);

if (eye\_width < 1) eye\_width = 1;

int eye\_x1 = x1 + eye\_width + 1;

int eye\_x2 = eye\_x1 + 2 \* eye\_width;

if (eye\_x2 > x2)

throw ShapeException("Head: глаза некорректно расположены.");

int nose\_x = (x1 + x2) / 2;

int nose\_y1 = eye\_y + head\_height / 6;

int nose\_y2 = nose\_y1 + head\_height / 6;

int mouth\_y = nose\_y2 + head\_height / 6;

int mouth\_width = head\_width \* 2 / 3;

int mouth\_x1 = x1 + (head\_width - mouth\_width) / 2;

int mouth\_x2 = mouth\_x1 + mouth\_width;

if (mouth\_x2 > x2 || mouth\_y > y2)

throw ShapeException("Head: рот некорректно расположен.");

Rectangle("temp", hat\_x1, hat\_y1, hat\_x2, hat\_y2).draw(canvas);

Line("temp", hat\_x1 - 1, hat\_y2, hat\_x2 + 1, hat\_y2).draw(canvas);

Rectangle("temp", head\_x1, head\_y1, head\_x2, head\_y2).draw(canvas);

for (int i = 0; i < (eye\_x2 - eye\_x1) / 2; i++) {

canvas.setPixel(eye\_x1 + i, eye\_y, '-');

canvas.setPixel(eye\_x2 + i, eye\_y, '-');

}

Line("temp", nose\_x, nose\_y1, nose\_x, nose\_y2).draw(canvas);

Line("temp", mouth\_x1, mouth\_y, mouth\_x2, mouth\_y).draw(canvas);

}

// Перемещение головы (in-place)

void translate(int dx, int dy) override {

int new\_x1 = orig\_x1 + dx, new\_y1 = orig\_y1 + dy, new\_x2 = orig\_x2 + dx, new\_y2 = orig\_y2 + dy;

if(new\_x1 < 0 || new\_x1 >= WIDTH || new\_y1 < 0 || new\_y1 >= HEIGHT ||

new\_x2 < 0 || new\_x2 >= WIDTH || new\_y2 < 0 || new\_y2 >= HEIGHT)

throw ShapeException("Head: перевод приводит к выходу за границы холста.");

orig\_x1 = new\_x1; orig\_y1 = new\_y1; orig\_x2 = new\_x2; orig\_y2 = new\_y2;

}

BoundingRect getBoundingRect() const override {

return { orig\_x1, orig\_y1, orig\_x2, orig\_y2 };

}

};

// -----------------------------

// Главная функция с меню

// -----------------------------

int main() {

Canvas canvas;

int choice;

while (true) {

system("clear");

canvas.render();

canvas.listShapes();

std::cout << "\n\n=== Меню ===\n";

std::cout << "0. Выход\n";

std::cout << "1. Очистить холст (удалить все фигуры)\n";

std::cout << "2. Добавить линию (имя x1 y1 x2 y2 [angle])\n";

std::cout << "3. Добавить прямоугольник (имя x1 y1 x2 y2 [angle])\n";

std::cout << "4. Добавить крест (имя x1 y1 x2 y2 [angle])\n";

std::cout << "5. Добавить голову (имя x1 y1 x2 y2)\n";

std::cout << "6. Удалить фигуру (имя)\n";

std::cout << "7. Переместить фигуру (имя dx dy)\n";

std::cout << "8. Повернуть фигуру (имя angle)\n";

std::cout << "9. Вывести все фигуры и их координаты\n";

std::cout << "Выберите опцию: ";

std::cin >> choice;

std::cin.ignore();

try {

if (choice == 0) {

std::cout << "Выход...\n";

break;

}

else if (choice == 1) {

canvas.clearShapes();

}

else if (choice == 2) {

std::cout << "Введите параметры для линии (имя x1 y1 x2 y2 [angle]): ";

std::string input;

std::getline(std::cin, input);

std::istringstream iss(input);

std::string name;

int x1, y1, x2, y2;

double angle = 0;

iss >> name >> x1 >> y1 >> x2 >> y2;

if (!(iss >> angle))

angle = 0;

Line\* line = new Line(name, x1, y1, x2, y2, angle);

canvas.addShape(line);

}

else if (choice == 3) {

std::cout << "Введите параметры для прямоугольника (имя x1 y1 x2 y2 [angle]): ";

std::string input;

std::getline(std::cin, input);

std::istringstream iss(input);

std::string name;

int x1, y1, x2, y2;

double angle = 0;

iss >> name >> x1 >> y1 >> x2 >> y2;

if (!(iss >> angle))

angle = 0;

Rectangle\* rect = new Rectangle(name, x1, y1, x2, y2, angle);

canvas.addShape(rect);

}

else if (choice == 4) {

std::cout << "Введите параметры для креста (имя x1 y1 x2 y2 [angle]): ";

std::string input;

std::getline(std::cin, input);

std::istringstream iss(input);

std::string name;

int x1, y1, x2, y2;

double angle = 0;

iss >> name >> x1 >> y1 >> x2 >> y2;

if (!(iss >> angle))

angle = 0;

DiagonalCross\* cross = new DiagonalCross(name, x1, y1, x2, y2, angle);

canvas.addShape(cross);

}

else if (choice == 5) {

std::cout << "Введите параметры для головы (имя x1 y1 x2 y2): ";

std::string input;

std::getline(std::cin, input);

std::istringstream iss(input);

std::string name;

int x1, y1, x2, y2;

iss >> name >> x1 >> y1 >> x2 >> y2;

Head\* head = new Head(name, x1, y1, x2, y2);

canvas.addShape(head);

}

else if (choice == 6) {

std::cout << "Введите имя фигуры для удаления: ";

std::string name;

std::cin >> name;

canvas.removeShape(name);

}

else if (choice == 7) {

std::cout << "Введите параметры для перемещения (имя dx dy): ";

std::string name;

int dx, dy;

std::cin >> name >> dx >> dy;

Shape\* shape = canvas.getShape(name);

shape->translate(dx, dy);

}

else if (choice == 8) {

std::cout << "Введите параметры для поворота (имя angle): ";

std::string name;

double angle;

std::cin >> name >> angle;

Shape\* shape = canvas.getShape(name);

IRotatable\* rotatable = dynamic\_cast<IRotatable\*>(shape);

if (rotatable) {

rotatable->rotate(angle);

} else {

throw ShapeException("Данная фигура не поддерживает вращение");

}

}

else if (choice == 9) {

canvas.listShapes();

std::cout << "Нажмите Enter для продолжения...";

std::cin.ignore();

std::getchar();

}

else {

std::cout << "Неправильный выбор!\n";

}

}

catch (const ShapeException &e) {

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

std::cout << "Нажмите Enter для продолжения...";

std::cin.ignore();

std::getchar();

}

}

std::cout << "Программа завершена.\n";

return 0;

}