МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра МОЭВМ

отчет

по лабораторной работе №2

по дисциплине «Конструирование ПО»

Тема: Разработка приложений с графическим интерфейсом пользователя

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Иванов В.С. |
| Преподаватель |  | Спицын А.В. |

Санкт-Петербург

2019

**Содержание**

[Цель работы 3](#_Toc25224787)

[Постановка задачи 3](#_Toc25224788)

[Индивидуальное задание 3](#_Toc25224789)

[Ход работы 4](#_Toc25224790)

[Выделение элементов архитектуры MVC 4](#_Toc25224791)

[Отображение графических фигур 5](#_Toc25224792)

[Работа с файлами 8](#_Toc25224793)

[Выводы 9](#_Toc25224794)

[Список использованных источников 10](#_Toc25224795)

[ПРИЛОЖЕНИЕ А 11](#_Toc25224796)

# **Цель работы**

Изучение модели построения программы – MVC (Model-View-Controller). Применение полученных знаний на практике путём создания программы с графическим интерфейсом пользователя с использованием набора полиморфных классов геометрических фигур из первой лабораторной работы.

# **Постановка задачи**

1. Просмотреть структуру классов проекта и выделить элементы архитектуры Model-View-Controller.
2. Модифицировать классы графических объектов из л/р 1 так чтобы они обладали возможностью отображать объекты в окне.

* Использовать для хранения элементов контейнер из л/р 1.
* К существующему режиму рисования элементов добавить режим изменения позиции элементов.
* Добавить изменение масштаба отображения фигур.

1. Добавить сериализацию и десериализацию из файла.

# **Индивидуальное задание**

1. Фигуры – правильный треугольник, правильный треугольник со сглаженными углами, текст, текст в треугольнике.
2. Контейнер – очередь с приоритетами на базе массива.

# **Ход работы**

Полный исходный код программы представлен в приложении А.

**Выделение элементов архитектуры MVC**

В качестве элемента представления представлен класс ShapeView. Класс отображает графическую сцену с фигурами с переопределенным событием поворота колеса мыши для изменения масштаба. Класс представлен на рисунке 1.

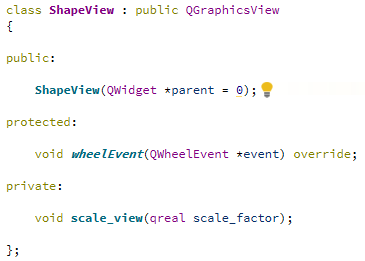


Рисунок 1 – Класс ShapeView

В качестве элемента модели представлен класс FiguresScene. Класс хранит фигуры и моделирует взаимодействие графических объектов для передачи информации компоненту отображения. Класс представлен на рисунке 2.



Рисунок 2 – Класс FiguresScene

В качестве элемента контроллера представлен класс MainWindow. Класс передает соответствующие изменения в элементы модели и представления в результате реакции на события взаимодействия с пользователем. Класс представлен на рисунке 3.



Рисунок 3 – Класс MainWindow

## **Отображение графических фигур**

Классы фигур из первой лабораторной работы унаследованы от QGraphicsItem и переопределяют метод paint() и boundingRect(). Классы представлены на рисунках 4-8. Демонстрация фигур представлена на рисунке 9.



Рисунок 4 – Класс Shape

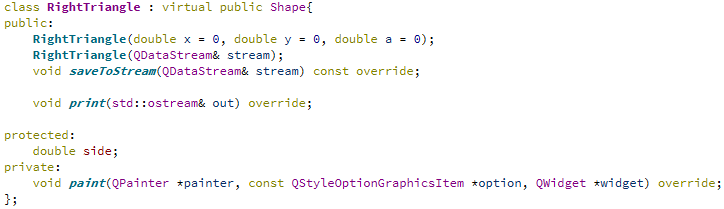


Рисунок 5 – Класс RightTriangle

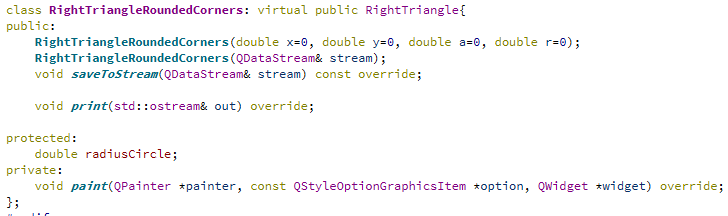


Рисунок 6 – Класс RightTriangleRoundedCorners

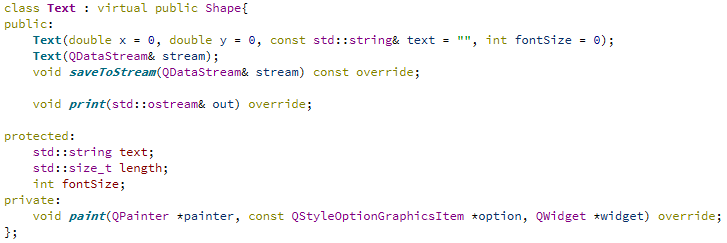


Рисунок 7 – Класс Text

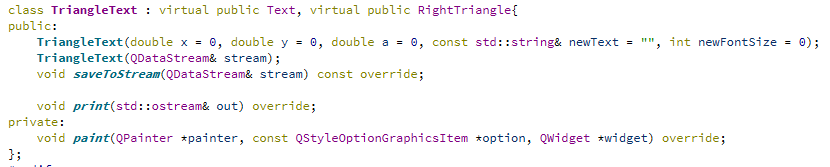


Рисунок 8 – Класс TriangleText

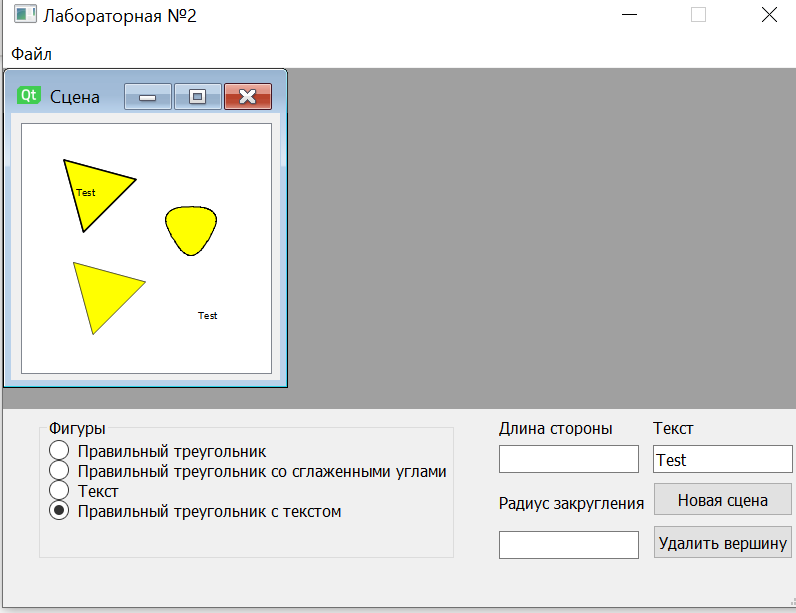


Рисунок 9 – Демонстрация отображения фигур

Для хранения фигур используется контейнер PriorityQueue из первой лабораторной работы.

Для изменения позиции элементов у фигур переопределён метод mouseMoveEvent(), а также переопределены методы mousePressEvent() и mouseReleaseEvent() для наглядности. Переопределение методов представлено на рисунке 10.

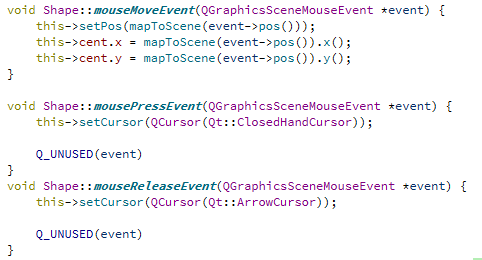


Рисунок 10 – Переопределенные методы для изменения позиции

Для изменения масштаба представления добавлена обработка событий колеса мыши. Обработка событий колеса мыши представлена на рисунке 11.

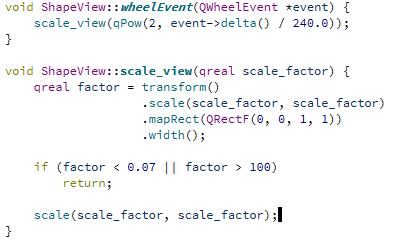


Рисунок 11 – Обработка событий колеса мыши

## **Работа с файлами**

Для сохранения фигур в файл и чтения фигур из файла используется оператор вывода фигур в поток и реализованы методы serialize() и deserialize(). Методы представлены на рисунке 12.

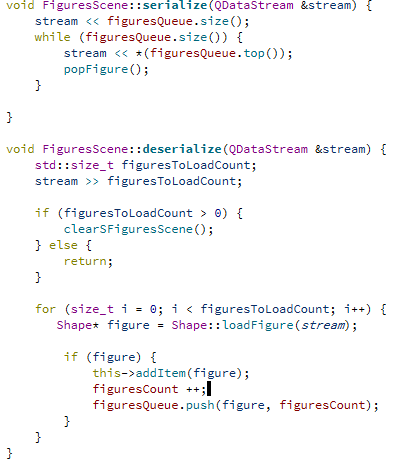


Рисунок 12 – Методы serialize() и deserialize()

**Выводы**

В результате выполнения лабораторной работы была изучена модель построения программы – MVC, была создана программа с графическим интерфейсом пользователя с возможностью отображения геометрических фигур из первой лабораторной работы. Также были изучены методы сохранения графических фигур в файлы и чтения из файлов.

# **Список использованных источников**

1. Шлее М. Qt 5.10. Профессиональное программирование на C++. – СПб.: БХВ-Петербург. – 2018. – 1072 с.
2. Cppreference. [Справка по C++](Справка%20по%20C++) [Электронный ресурс]. URL: <https://ru.cppreference.com/w/> (дата обращения: 17.11.2019).
3. Evileg. Qt/C++ [Электронный ресурс]. URL: <https://evileg.com/ru/knowledge/qt/> (дата обращения: 19.11.2019).
4. Qt. Qt Documentation [Электронный ресурс]. URL: <https://doc.qt.io/> (дата обращения: 20.11.2019).

# **ПРИЛОЖЕНИЕ А**

**Исходный код**

#ifndef FIGURESSCENE\_H

#define FIGURESSCENE\_H

#include <QWidget>

#include <QGraphicsScene>

#include "shape.h"

#include "priorityQueueContainer.h"

namespace **Ui** {

class **FiguresScene**;

}

class **FiguresScene** : public QGraphicsScene

{

Q\_OBJECT

public:

explicit **FiguresScene**(QObject \*parent = nullptr);

void **setFigureType**(QString newFigureType);

void **setFigureSideLength**(int newLength);

void **setFigureRoundedRadius**(int newRadius);

void **setFigureText**(QString newText);

void **popFigure**();

void **clearSFiguresScene**();

void **serialize**(QDataStream& stream);

void **deserialize**(QDataStream& stream);

QString **getFigureType**() const;

~***FiguresScene***();

private:

Ui::FiguresScene \*ui;

QGraphicsScene \*scene; // Объявляем графическую сцену

QString typeFigure = "triangle";

int figureSideLength = 150;

int figureRoundedRadius = 40;

QString figureText = "\"\"";

Shape\* shape;

PriorityQueue<Shape\*> figuresQueue;

int figuresCount = 0;

protected:

void ***mouseDoubleClickEvent***(QGraphicsSceneMouseEvent \*event) override;

};

#endif // FIGURESSCENE\_H

#pragma once

class **Colour**{

public:

short r;

short g;

short b;

**Colour** (short r, short g, short b):r(r),g(g),b(b){}

};

#ifndef FIGURESSCENEMDI\_H

#define FIGURESSCENEMDI\_H

#include "figuresscene.h"

#include "mainwindow.h"

#include <QWidget>

namespace Ui {

class FiguresSceneMdi;

}

class FiguresSceneMdi : public QWidget

{

Q\_OBJECT

friend class MainWindow;

public:

explicit FiguresSceneMdi(QWidget \*parent = nullptr);

~FiguresSceneMdi();

private:

Ui::FiguresSceneMdi \*ui;

FiguresScene\* figureScene;

};

#endif // FIGURESSCENEMDI\_H

#ifndef HEADERS\_H

#define HEADERS\_H

#include <iostream>

#include <cmath>

#include <vector>

#include <algorithm>

#include <string>

#include <memory>

#include <QPair>

#include <QSharedPointer>

#include <QPainter>

#include <QPen>

#include <QTextOption>

#include <QtMath>

#include "Point.h"

#include "Colour.h"

#endif //HEADERS\_H

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QList>

#include "figuresscene.h"

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private slots:

void **on\_radioButtonTriangle\_clicked**();

void **on\_radioButtonTriangleWithCorners\_clicked**();

void **on\_radioButtonText\_clicked**();

void **on\_radioButtonTriangleText\_clicked**();

void **on\_lineEditLength\_textChanged**(const QString &arg1);

void **on\_lineEditText\_textChanged**(const QString &arg1);

void **on\_lineEditRadius\_textChanged**(const QString &arg1);

void **on\_pushButton\_clicked**();

void **on\_newSceneButton\_clicked**();

void **on\_openAction\_triggered**();

void **on\_saveAction\_triggered**();

private:

Ui::MainWindow \*ui;

FiguresScene\* **getCurrentScene**();

QList<FiguresScene\*> **getAllScenes**();

};

#endif // MAINWINDOW\_H

#pragma once

class **Point** {

public:

**Point**(double x, double y): x(x),y(y){}

double x;

double y;

};

#ifndef PRIORITYQUEUECONTAINER\_H

#define PRIORITYQUEUECONTAINER\_H

#include "headers.h"

#include "priorityQueueExceptions.h"

template<typename T, class Container = std::vector<QPair<int, T>>>

class PriorityQueue;

template <typename T>

class PriorityQueueIterator {

public:

PriorityQueueIterator (typename std::vector<QPair<int, T>>::reverse\_iterator ptr\_, bool empty) : ptr(ptr\_), isEmpty(empty) {}

QPair<int, T>& operator\*()

{

if (isEmpty) {

throw QueueInvalidIterator<T>(isEmpty, ptr);

}

return \*ptr;

}

const typename std::vector<QPair<int, T>>::reverse\_iterator operator->() {

if (isEmpty) {

throw QueueInvalidIterator<T>(isEmpty, ptr);

}

return ptr;

}

typename std::vector<QPair<int, T>>::reverse\_iterator operator++() {

return ++ptr;

}

PriorityQueueIterator operator++(int) {

PriorityQueueIterator<T> orig = \*this;

++(\*this);

return orig;

}

typename std::vector<QPair<int, T>>::reverse\_iterator operator--() {

return --ptr;

}

PriorityQueueIterator operator--(int) {

PriorityQueueIterator<T> orig = \*this;

--(\*this);

return orig;

}

bool operator==(const PriorityQueueIterator& other) const {

return ptr == other.ptr;

}

bool operator!=(const PriorityQueueIterator& other) const {

return !(\*this == other);

}

private:

typename std::vector<QPair<int, T>>::reverse\_iterator ptr;

bool isEmpty;

};

template<typename T, class Container>

class PriorityQueue {

public:

friend class PriorityQueueIterator<T>;

void push(T const& lhs, int priority) {

if (!elements.size() || elements.back().first < priority) {

elements.push\_back(qMakePair(priority,lhs));

}

else {

// std::sort(elements.begin(), elements.end(), comparePairs<int, T>);

for (int i = 0; i < elements.size(); i++) {

if (elements[i].first > priority) {

elements.insert(elements.begin() + i, qMakePair(priority,lhs));

break;

}

}

}

}

void push(T const& lhs) {

if (elements.size()){

elements.push\_back(qMakePair(elements.back().first + 1,lhs));

} else {

elements.push\_back(qMakePair(1,lhs));

}

}

void pop() {

if (empty()) {

throw QueueBlank<T>(elements);

}

elements.pop\_back();

}

T const& top() const {

if (empty()) {

throw QueueBlank<T>(elements);

}

return elements.back().second;

}

typedef typename Container::size\_type size\_type;

size\_type size() const {

return elements.size();

}

bool empty() const {

return elements.empty();

}

PriorityQueueIterator<T> begin() {

return PriorityQueueIterator<T>(elements.rbegin(), empty());

}

PriorityQueueIterator<T> end() {

return PriorityQueueIterator<T>(elements.rend(), empty());

}

private:

Container elements;

};

#endif // PRIORITYQUEUECONTAINER\_H

#ifndef PRIORITYQUEUEEXCEPTIONS\_H

#define PRIORITYQUEUEEXCEPTIONS\_H

#include "headers.h"

class **QueueException** {

public:

**QueueException**() = default;

friend inline std::ostream &operator<<(std::ostream &out, QueueException &e){

e.*print*(*out*);

return out;

};

virtual void ***print***(std::ostream& out) = 0;

virtual ~***QueueException***() = default;

};

template <typename T>

class **QueueInvalidIterator** : public QueueException {

public:

inline **QueueInvalidIterator**(const bool isEmpty, typename std::vector<QPair<int, T>>::reverse\_iterator ptr): isEmpty(isEmpty), ptr(ptr) {

}

void ***print***(std::ostream &out) {

out << "QueueInvalidIterator: " << &ptr;

}

private:

typename std::vector<QPair<int, T>>::reverse\_iterator ptr;

bool isEmpty;

};

template <typename T>

class **QueueBlank**: public QueueException {

public:

inline **QueueBlank**(const std::vector<QPair<int, T>>& queue): queue(queue) {

queueLength = queue.size();

}

void ***print***(std::ostream &out) {

out << "QueueBlank: " << queueLength;

}

private:

std::vector<QPair<int, T>> queue;

std::size\_t queueLength;

};

#endif // PRIORITYQUEUEEXCEPTIONS\_H

#ifndef RIGHTTRIANGLE\_H

#define RIGHTTRIANGLE\_H

#include "shape.h"

class **RightTriangle** : virtual public Shape{

public:

**RightTriangle**(double x = 0, double y = 0, double a = 0);

**RightTriangle**(QDataStream& stream);

void ***saveToStream***(QDataStream& stream) const override;

void ***print***(std::ostream& out) override;

protected:

double side;

private:

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

};

#endif

#ifndef RIGHTTRIANGLEROUNDEDCORNERS\_H

#define RIGHTTRIANGLEROUNDEDCORNERS\_H

#include "headers.h"

#include "righttriangle.h"

class **RightTriangleRoundedCorners**: virtual public RightTriangle{

public:

**RightTriangleRoundedCorners**(double x=0, double y=0, double a=0, double r=0);

**RightTriangleRoundedCorners**(QDataStream& stream);

void ***saveToStream***(QDataStream& stream) const override;

void ***print***(std::ostream& out) override;

protected:

double radiusCircle;

private:

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

};

#endif

#ifndef SHAPE\_H

#define SHAPE\_H

#include <QDataStream>

#include <QGraphicsObject>

#include <QGraphicsSceneMouseEvent>

class **FiguresScene**;

#include "headers.h"

class **Shape**: public QGraphicsObject{

friend class FiguresScene;

public:

**Shape**(double x = 0, double y = 0);

**Shape**(QDataStream& stream);

static Shape\* **loadFigure**(QDataStream& stream);

virtual void ***changePos***(double x, double y);

void virtual ***saveToStream***(QDataStream& stream) const = 0;

virtual void ***forPrint***(std::ostream& out);

virtual void ***changeColour***(short r, short g, short b);

Point **getCentCoords**() const;

virtual void ***print***(std::ostream& out) = 0;

friend std::ostream& operator<<(std::ostream& out, Shape& sh){

sh.*forPrint*(*out*);

sh.*print*(*out*);

return out;

}

virtual ~***Shape***(){}

protected:

Point cent = Point(0,0);

int ang;

Colour col = Colour(255,255,0);

std:: vector<Point> pts;

QRectF figureRect;

QRectF ***boundingRect***() const override;

void ***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event) override;

void ***mousePressEvent***(QGraphicsSceneMouseEvent \*event) override;

void ***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event) override;

};

QDataStream& operator<<(QDataStream& stream, const Shape& shape);

QDataStream& operator<<(QDataStream& stream, const Shape\* shape);

#endif

#ifndef SHAPEVIEW\_H

#define SHAPEVIEW\_H

#include <QGraphicsView>

class **ShapeView** : public QGraphicsView

{

public:

**ShapeView**(QWidget \*parent = 0);

protected:

void ***wheelEvent***(QWheelEvent \*event) override;

private:

void **scale\_view**(qreal scale\_factor);

};

#endif // SHAPEVIEW\_H

#ifndef TEXT\_H

#define TEXT\_H

#include <string>

#include "headers.h"

#include "shape.h"

class **Text** : virtual public Shape{

public:

**Text**(double x = 0, double y = 0, const std::string& text = "", int fontSize = 0);

**Text**(QDataStream& stream);

void ***saveToStream***(QDataStream& stream) const override;

void ***print***(std::ostream& out) override;

protected:

std::string text;

std::size\_t length;

int fontSize;

private:

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

};

#endif

#ifndef TRIANGLETEXT\_H

#define TRIANGLETEXT\_H

#include "text.h"

#include "righttriangle.h"

#include "headers.h"

class **TriangleText** : virtual public Text, virtual public RightTriangle{

public:

**TriangleText**(double x = 0, double y = 0, double a = 0, const std::string& newText = "", int newFontSize = 0);

**TriangleText**(QDataStream& stream);

void ***saveToStream***(QDataStream& stream) const override;

void ***print***(std::ostream& out) override;

private:

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) override;

};

#endif

#include "figuresscene.h"

#include "righttriangle.h"

#include "righttriangleroundedcorners.h"

#include "text.h"

#include "shape.h"

#include "triangletext.h"

#include <QMouseEvent>

#include <QGraphicsSceneEvent>

FiguresScene::**FiguresScene**(QObject \*parent)

: QGraphicsScene(parent) {

setItemIndexMethod(QGraphicsScene::NoIndex);

}

FiguresScene::~***FiguresScene***(){

}

void FiguresScene::**setFigureType**(QString newFigureType) {

typeFigure = newFigureType;

}

void FiguresScene::**setFigureRoundedRadius**(int newRadius) {

if (newRadius < 0) {

newRadius = 0;

}

figureRoundedRadius = newRadius;

}

void FiguresScene::**setFigureSideLength**(int newLength) {

if (newLength <= 0) {

newLength = 1;

}

figureSideLength = newLength;

}

void FiguresScene::**setFigureText**(QString newText) {

if (!newText.size()) {

newText = "\"\"";

}

figureText = newText;

}

void FiguresScene::**popFigure**() {

try {

auto item = this->itemAt(figuresQueue.top()->getCentCoords().x, figuresQueue.top()->getCentCoords().y, QTransform::fromScale(1, 1));

this->removeItem(item);

figuresQueue.pop();

figuresCount--;

} catch (QueueException& e) {

std::cout<<"Exception"<<std::endl;

}

}

void FiguresScene::**serialize**(QDataStream &stream) {

stream << figuresQueue.size();

while (figuresQueue.size()) {

stream << \*(figuresQueue.top());

popFigure();

}

}

void FiguresScene::**clearSFiguresScene**() {

this->clear();

while (figuresQueue.size()) {

figuresQueue.pop();

};

}

void FiguresScene::**deserialize**(QDataStream &stream) {

std::size\_t figuresToLoadCount;

stream >> figuresToLoadCount;

if (figuresToLoadCount > 0) {

clearSFiguresScene();

} else {

return;

}

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

Shape\* figure = Shape::loadFigure(*stream*);

if (figure) {

this->addItem(figure);

figuresCount ++;

figuresQueue.push(figure, figuresCount);

}

}

}

QString FiguresScene::**getFigureType**() const {

return typeFigure;

}

void FiguresScene::***mouseDoubleClickEvent***(QGraphicsSceneMouseEvent \*event) {

auto item = this->itemAt(event->scenePos(), QTransform::fromScale(1, 1));

if(!item) {

auto type = this->getFigureType().toStdString();

if (type == "triangle") {

shape = new RightTriangle(event->scenePos().rx(),event->scenePos().ry(),this->figureSideLength);

} else if (type == "triangleWithCorners") {

shape = new RightTriangleRoundedCorners(event->scenePos().rx(),event->scenePos().ry(),this->figureSideLength,this->figureRoundedRadius);

} else if (type == "text") {

shape = new Text(event->scenePos().rx(),event->scenePos().ry(),this->figureText.toStdString(),15);

} else {

shape = new TriangleText(event->scenePos().rx(),event->scenePos().ry(),this->figureSideLength,this->figureText.toStdString(),12);

}

shape->setPos(event->scenePos());

this->addItem(shape);

figuresCount ++;

figuresQueue.push(shape, figuresCount);

}

}

#include "figuresscenemdi.h"

#include "ui\_figuresscene.h"

#include "shapeview.h"

#include "righttriangle.h"

#include "righttriangleroundedcorners.h"

#include "text.h"

#include "triangletext.h"

FiguresSceneMdi::**FiguresSceneMdi**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::FiguresSceneMdi)

{

ui->setupUi(this);

figureScene = new FiguresScene(this);

ui->graphicsView->setScene(figureScene);

}

FiguresSceneMdi::~***FiguresSceneMdi***()

{

delete ui;

}

#include "mainwindow.h"

#include "headers.h"

#include "Point.h"

#include "Colour.h"

#include "shape.h"

#include "righttriangle.h"

#include "righttriangleroundedcorners.h"

#include "text.h"

#include "triangletext.h"

#include "priorityQueueExceptions.h"

#include "priorityQueueContainer.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "figuresscene.h"

#include "figuresscenemdi.h"

#include <QMdiSubWindow>

#include <QMdiArea>

#include <QGraphicsView>

#include <QFileDialog>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

this->setWindowTitle("Лабораторная №2");

this->setFixedSize(QSize(this->width(), this->height()));

auto newWindow = new FiguresSceneMdi(ui->mdiArea);

auto newFiguresScene = ui->mdiArea->addSubWindow(newWindow);

newFiguresScene->setFixedSize(QSize(285, 320));

newFiguresScene->setWindowTitle("Сцена");

}

FiguresScene \*MainWindow::**getCurrentScene**() {

auto currentSubwindow = ui->mdiArea->currentSubWindow();

if (currentSubwindow) {

auto currentScene = currentSubwindow->widget();

auto currentWindow = dynamic\_cast<FiguresSceneMdi\*>(currentScene);

return currentWindow->figureScene;

} else {

return nullptr;

}

}

QList<FiguresScene \*> MainWindow::**getAllScenes**() {

auto scenes = ui->mdiArea->subWindowList();

QList<FiguresScene\*> result;

for (auto scene : scenes) {

auto scene\_as\_widget = scene->widget();

auto shape\_scene\_mdi\_window = dynamic\_cast<FiguresSceneMdi\*>(scene\_as\_widget);

result.push\_back(shape\_scene\_mdi\_window->figureScene);

}

return result;

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_radioButtonTriangle\_clicked**()

{

auto scenes = getAllScenes();

for (auto scene : scenes) {

scene->setFigureType("triangle");

}

}

void MainWindow::**on\_radioButtonTriangleWithCorners\_clicked**()

{

auto scenes = getAllScenes();

for (auto scene : scenes) {

scene->setFigureType("triangleWithCorners");

}

}

void MainWindow::**on\_radioButtonText\_clicked**()

{

auto scenes = getAllScenes();

for (auto scene : scenes) {

scene->setFigureType("text");

}

}

void MainWindow::**on\_radioButtonTriangleText\_clicked**()

{

auto scenes = getAllScenes();

for (auto scene : scenes) {

scene->setFigureType("triangleText");

}

}

void MainWindow::**on\_lineEditLength\_textChanged**(const QString &arg1)

{

auto scenes = getAllScenes();

for (auto scene : scenes) {

scene->setFigureSideLength(arg1.toInt());

}

}

void MainWindow::**on\_lineEditText\_textChanged**(const QString &arg1)

{

auto scenes = getAllScenes();

for (auto scene : scenes) {

scene->setFigureText(arg1);

}

}

void MainWindow::**on\_lineEditRadius\_textChanged**(const QString &arg1)

{

auto scenes = getAllScenes();

for (auto scene : scenes) {

scene->setFigureRoundedRadius(arg1.toInt());

}

}

void MainWindow::**on\_pushButton\_clicked**()

{

auto scene = getCurrentScene();

scene->popFigure();

}

void MainWindow::**on\_newSceneButton\_clicked**()

{

auto newWindow = new FiguresSceneMdi(ui->mdiArea);

auto newFiguresScene = ui->mdiArea->addSubWindow(newWindow);

newFiguresScene->setFixedSize(QSize(285, 320));

newFiguresScene->setWindowTitle("Сцена");

newWindow->show();

}

void MainWindow::**on\_openAction\_triggered**()

{

auto currentScene = getCurrentScene();

if (!currentScene) {

return;

}

auto file\_name = QFileDialog::getOpenFileName(this, "Открыть из файла", QString(), "Text File(\*.txt)");

if (file\_name.isEmpty()) {

return;

}

QFile file(file\_name);

if (file.*open*(QIODevice::ReadOnly)) {

QDataStream input(&file);

currentScene->deserialize(*input*);

}

file.*close*();

}

void MainWindow::**on\_saveAction\_triggered**()

{

auto currentScene = getCurrentScene();

if (!currentScene) {

return;

}

auto file\_name = QFileDialog::getSaveFileName(this, "Сохранить в файл", QString(), "Text File(\*.txt)");

if (file\_name.isEmpty()) {

return;

}

QFile file(file\_name);

if (file.*open*(QIODevice::WriteOnly)) {

QDataStream output(&file);

currentScene->serialize(*output*);

}

file.*close*();

}

#include "righttriangle.h"

#include <QTextOption>

RightTriangle::**RightTriangle**(double x, double y, double a) : Shape(x,y), side(a){

pts.push\_back({cent.x,cent.y+a\*sqrt(3)/3});

pts.push\_back({cent.x-a/2,cent.y-a\*sqrt(3)/6});

pts.push\_back({cent.x+a/2,cent.y-a\*sqrt(3)/6});

figureRect = QRectF(-(a\*cos(M\_PI/12))/2, -(a\*cos(M\_PI/12)) / 2, a\*cos(M\_PI/12),a\*cos(M\_PI/12));

}

RightTriangle::**RightTriangle**(QDataStream &stream)

: Shape(*stream*) {

stream >> side;

}

void RightTriangle::***saveToStream***(QDataStream &stream) const {

stream << QString::fromStdString("triangle");

stream << figureRect;

stream << QPoint(cent.x, cent.y);

stream << scenePos();

stream << side;

}

void RightTriangle::***print***(std::ostream& out){

out<<"Сторона "<< side<<"\n";

}

void RightTriangle::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) {

QPolygon polygon; // Используем класс полигона, чтобы отрисовать треугольник

// Помещаем координаты точек в полигональную модель

// for (Point point: pts) {

// polygon << QPoint(point.x,point.y);

// }

auto firstPoint = figureRect.topLeft().toPoint();

auto secondPoint = QPoint(figureRect.bottomLeft().toPoint().rx() + side\*sin(M\_PI/12),figureRect.bottomRight().toPoint().ry());

auto thirdPoint = QPoint(figureRect.topRight().toPoint().rx(),figureRect.topRight().toPoint().ry() + side\*sin(M\_PI/12));

polygon << firstPoint << secondPoint << thirdPoint;

painter->drawPolygon(polygon);

painter->setBrush(QColor(col.r, col.g, col.b)); // Устанавливаем кисть, которой будем отрисовывать объект

painter->drawPolygon(polygon); // Рисуем треугольник по полигональной модели

Q\_UNUSED(option)

Q\_UNUSED(widget)

}

#include "righttriangleroundedcorners.h"

double **getPt**( int n1 , int n2 , float perc )

{

double diff = n2 - n1;

return n1 + ( diff \* perc );

}

RightTriangleRoundedCorners::**RightTriangleRoundedCorners**(double x, double y, double a, double r) : RightTriangle(x,y,a), radiusCircle(r){

pts.clear();

if (r < a\*sqrt(3)/3) {

float x1 = cent.x + r\*sqrt(3)/2;

float x2 = cent.x;

float x3 = cent.x - r\*sqrt(3)/2;

float y1 = cent.y + a\*sqrt(3)/3 - 3\*r/2;

float y2 = cent.y+a\*sqrt(3)/3;

float y3 = cent.y + a\*sqrt(3)/3 - 3\*r/2;

for( float i = 0 ; i < 1 ; i += 0.01 )

{

pts.push\_back({getPt( getPt( x1 , x2 , i ) , getPt( x2 , x3 , i ) , i ), getPt(getPt( y1 , y2 , i ) , getPt( y2 , y3 , i ) , i)});

}

//лево низ

x1 = cent.x - a/2 + r\*sqrt(3)/2;

x2 = cent.x-a/2;

x3 = cent.x - a/2 + r\*sqrt(3);

y1 = cent.y - a\*sqrt(3)/6 + 3\*r/2;

y2 = cent.y-a\*sqrt(3)/6;

y3 = cent.y - a\*sqrt(3)/6;

for( float i = 0 ; i < 1 ; i += 0.01 )

{

pts.push\_back({getPt( getPt( x1 , x2 , i ) , getPt( x2 , x3 , i ) , i ), getPt(getPt( y1 , y2 , i ) , getPt( y2 , y3 , i ) , i)});

}

//право низ

x1 = cent.x + a/2 - r\*sqrt(3);

x2 = cent.x+a/2;

x3 = cent.x + a/2 - r\*sqrt(3)/2;

y1 = cent.y - a\*sqrt(3)/6;

y2 = cent.y-a\*sqrt(3)/6;

y3 = cent.y - a\*sqrt(3)/6 + 3\*r/2;

for( float i = 0 ; i < 1 ; i += 0.01 )

{

pts.push\_back({getPt( getPt( x1 , x2 , i ) , getPt( x2 , x3 , i ) , i ), getPt(getPt( y1 , y2 , i ) , getPt( y2 , y3 , i ) , i)});

}

}

else {

pts.push\_back({cent.x - r\*sqrt(3)/2, cent.y + r/2});

pts.push\_back({cent.x + r\*sqrt(3)/2, cent.y + r/2});

pts.push\_back({cent.x, cent.y - r});

}

figureRect = QRectF(-(a\*cos(M\_PI/12))/2, -(a\*cos(M\_PI/12)) / 2, a\*cos(M\_PI/12),a\*cos(M\_PI/12));

}

void RightTriangleRoundedCorners::***saveToStream***(QDataStream &stream) const {

stream << QString::fromStdString("triangleWithCorners");

stream << figureRect;

stream << QPoint(cent.x, cent.y);

stream << scenePos();

stream << side;

stream << radiusCircle;

}

RightTriangleRoundedCorners::**RightTriangleRoundedCorners**(QDataStream &stream)

: RightTriangle(*stream*) {

stream >> radiusCircle;

if (radiusCircle < side\*sqrt(3)/3) {

float x1 = cent.x + radiusCircle\*sqrt(3)/2;

float x2 = cent.x;

float x3 = cent.x - radiusCircle\*sqrt(3)/2;

float y1 = cent.y + side\*sqrt(3)/3 ;

float y2 = cent.y + side\*sqrt(3)/3 - 3\*radiusCircle/2;

float y3 = cent.y + side\*sqrt(3)/3 ;

for( float i = 0 ; i < 1 ; i += 0.01 )

{

pts.push\_back({getPt( getPt( x1 , x2 , i ) , getPt( x2 , x3 , i ) , i ), getPt(getPt( y1 , y2 , i ) , getPt( y2 , y3 , i ) , i)});

}

//верх

//pts.push\_back({cent.x + r\*sqrt(3)/2, cent.y + a\*sqrt(3)/3 - 3\*r/2});

//pts.push\_back({cent.x, cent.y + a\*sqrt(3)/3 - r});

//pts.push\_back({cent.x - r\*sqrt(3)/2, cent.y + a\*sqrt(3)/3 - 3\*r/2});

//лево низ

x1 = cent.x + side/2 - radiusCircle\*sqrt(3)/2;

x2 = cent.x+side/2;

x3 = cent.x + side/2 - radiusCircle\*sqrt(3);

y1 = cent.y - side\*sqrt(3)/6 ;

y2 = cent.y-side\*sqrt(3)/6+ 3\*radiusCircle/2;

y3 = cent.y - side\*sqrt(3)/6+ 3\*radiusCircle/2;

for( float i = 0 ; i < 1 ; i += 0.01 )

{

pts.push\_back({getPt( getPt( x1 , x2 , i ) , getPt( x2 , x3 , i ) , i ), getPt(getPt( y1 , y2 , i ) , getPt( y2 , y3 , i ) , i)});

}

//pts.push\_back({cent.x - a/2 + r\*sqrt(3)/2, cent.y - a\*sqrt(3)/6 + 3\*r/2});

// pts.push\_back({cent.x - a/2 + r\*sqrt(3)/2, cent.y - a\*sqrt(3)/6 + r/2});

//pts.push\_back({cent.x - a/2 + r\*sqrt(3), cent.y - a\*sqrt(3)/6});

//право низ

x1 = cent.x - side/2+ radiusCircle\*sqrt(3);

x2 = cent.x-side/2;

x3 = cent.x - side/2 + radiusCircle\*sqrt(3)/2;

y1 = cent.y - side\*sqrt(3)/6+ 3\*radiusCircle/2;

y2 = cent.y-side\*sqrt(3)/6+ 3\*radiusCircle/2;

y3 = cent.y - side\*sqrt(3)/6 ;

for( float i = 0 ; i < 1 ; i += 0.01 )

{

pts.push\_back({getPt( getPt( x1 , x2 , i ) , getPt( x2 , x3 , i ) , i ), getPt(getPt( y1 , y2 , i ) , getPt( y2 , y3 , i ) , i)});

}

//pts.push\_back({cent.x + a/2 - r\*sqrt(3), cent.y - a\*sqrt(3)/6});

//pts.push\_back({cent.x + a/2 - r\*sqrt(3)/2, cent.y - a\*sqrt(3)/6 + r/2});

//pts.push\_back({cent.x + a/2 - r\*sqrt(3)/2, cent.y - a\*sqrt(3)/6 + 3\*r/2});

}

else {

pts.push\_back({cent.x - radiusCircle\*sqrt(3)/2, cent.y + radiusCircle/2});

pts.push\_back({cent.x + radiusCircle\*sqrt(3)/2, cent.y + radiusCircle/2});

pts.push\_back({cent.x, cent.y - radiusCircle});

}

}

void RightTriangleRoundedCorners::***print***(std::ostream& out){

out<<"Радиус окружности сглаживания "<< radiusCircle<<"\n";

}

void RightTriangleRoundedCorners::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) {

QPolygon polygon; // Используем класс полигона, чтобы отрисовать треугольник

// Помещаем координаты точек в полигональную модель

for (Point point: pts) {

polygon << QPoint(point.x,point.y);

}

painter->drawPolygon(polygon);

painter->setPen(QPen(Qt::black, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter->setBrush(QColor(col.r, col.g, col.b)); // Устанавливаем кисть, которой будем отрисовывать объект

painter->drawPolygon(polygon); // Рисуем треугольник по полигональной модели

Q\_UNUSED(option)

Q\_UNUSED(widget)

}

#include "shape.h"

#include "righttriangle.h"

#include "righttriangleroundedcorners.h"

#include "text.h"

#include "triangletext.h"

#include <QGraphicsSceneEvent>

#include <QCursor>

Shape::**Shape**(double x, double y): ang(0), cent(x,y), col(255,255,0){

this->setPos(x,y);

}

Shape::**Shape**(QDataStream& stream) {

stream >> figureRect;

QPoint qcent;

stream >> qcent;

cent = Point(qcent.x(),qcent.y());

QPointF pos;

stream >> pos;

setPos(pos);

}

void Shape::***changePos***(double x, double y){

for(auto& it: pts){

it.x+=x - cent.x;

it.y+=y - cent.y;

}

this->setPos(x,y);

cent.x=x;

cent.y=y;

}

Shape\* Shape::**loadFigure**(QDataStream &stream) {

QString type;

stream >> type;

if (type == "triangle") {

return new RightTriangle(*stream*);

} else if (type == "triangleWithCorners") {

return new RightTriangleRoundedCorners(*stream*);

} else if (type == "text") {

return new Text(*stream*);

} else {

return new TriangleText(*stream*);

}

}

Point Shape::**getCentCoords**() const {

return cent;

}

QRectF Shape::***boundingRect***() const {

return figureRect;

}

void Shape::***changeColour***(short r, short g, short b){

col={r,g,b};

}

void Shape::***forPrint***(std::ostream& out){

out<<"Центр. коорд. "<<cent.x<<" "<<cent.y<<std::endl;

out<<"Угол поворота "<<ang<<std::endl;

out<<"Точки "<<std::endl;

int count=0;

for(const auto& it: pts){

count++;

out<<count<<") ("<<it.x<<", "<<it.y<<")\n";

}

out<<"Цвет "<<col.r<<" "<<col.g<<" "<<col.b<<std::endl;

}

void Shape::***mouseMoveEvent***(QGraphicsSceneMouseEvent \*event) {

this->setPos(mapToScene(event->pos()));

this->cent.x = mapToScene(event->pos()).x();

this->cent.y = mapToScene(event->pos()).y();

}

void Shape::***mousePressEvent***(QGraphicsSceneMouseEvent \*event) {

this->setCursor(QCursor(Qt::ClosedHandCursor));

Q\_UNUSED(event)

}

void Shape::***mouseReleaseEvent***(QGraphicsSceneMouseEvent \*event) {

this->setCursor(QCursor(Qt::ArrowCursor));

Q\_UNUSED(event)

}

QDataStream& operator<<(QDataStream& stream, const Shape& shape) {

shape.*saveToStream*(*stream*);

return stream;

}

QDataStream& operator<<(QDataStream& stream, const Shape\* shape) {

shape->*saveToStream*(*stream*);

return stream;

}

#include "shapeview.h"

#include <QWheelEvent>

#include <QtMath>

ShapeView::**ShapeView**(QWidget \*parent)

: QGraphicsView(parent) {

setViewportUpdateMode(QGraphicsView::BoundingRectViewportUpdate);

setRenderHint(QPainter::Antialiasing, true);

scale(1, 1);

setDragMode(QGraphicsView::RubberBandDrag);

setTransformationAnchor(AnchorUnderMouse);

}

void ShapeView::***wheelEvent***(QWheelEvent \*event) {

scale\_view(qPow(2, event->delta() / 240.0));

}

void ShapeView::**scale\_view**(qreal scale\_factor) {

qreal factor = transform()

.scale(scale\_factor, scale\_factor)

.mapRect(QRectF(0, 0, 1, 1))

.width();

if (factor < 0.07 || factor > 100)

return;

scale(scale\_factor, scale\_factor);

}

#include "text.h"

Text::**Text**(double x, double y, const std::string& text, int fontSize) : Shape(x,y), text(text), fontSize(fontSize){

length = text.length();

figureRect = QRectF(-10/1.5,-10/2, (10\*1.2)\*length, 30);

}

Text::**Text**(QDataStream &stream)

: Shape(*stream*) {

QString qtext;

stream >> qtext;

text = qtext.toStdString();

stream >> fontSize;

length = text.length();

}

void Text::***saveToStream***(QDataStream &stream) const {

stream << QString::fromStdString("text");

stream << figureRect;

stream << QPoint(cent.x, cent.y);

stream << scenePos();

stream << QString::fromStdString(text);

stream << fontSize;

}

void Text::***print***(std::ostream& out){

out<<"Длина текста "<< length<<"\n";

out<<"Размер шрифта "<< fontSize<<"\n";

out<<"Текст "<< text<<"\n";

}

void Text::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) {

painter->setPen(Qt::white);

painter->setBrush(Qt::white);

painter->drawRect(figureRect);

painter->setPen(QPen(Qt::black, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter->setBrush(QColor(col.r, col.g, col.b));

auto font = painter->font();

font.setPointSize(10);

painter->setFont(font);

painter->drawText(figureRect, QString::fromStdString(text));

Q\_UNUSED(option)

Q\_UNUSED(widget)

}

#include "triangletext.h"

TriangleText::**TriangleText**(double x, double y, double a, const std::string& newText, int newFontSize) : Shape(x,y){

side = a;

pts.clear();

pts.push\_back({cent.x,cent.y+a\*sqrt(3)/3});

pts.push\_back({cent.x-a/2,cent.y-a\*sqrt(3)/6});

pts.push\_back({cent.x+a/2,cent.y-a\*sqrt(3)/6});

text = newText;

fontSize = newFontSize;

length = text.length();

figureRect = QRectF(-(a\*cos(M\_PI/12))/2, -(a\*cos(M\_PI/12)) / 2, a\*cos(M\_PI/12),a\*cos(M\_PI/12));

}

TriangleText::**TriangleText**(QDataStream &stream)

: Shape(*stream*) {

stream >> side;

QString qtext;

stream >> qtext;

text = qtext.toStdString();

stream >> fontSize;

length = text.length();

}

void TriangleText::***saveToStream***(QDataStream &stream) const {

stream << QString::fromStdString("triangleText");

stream << figureRect;

stream << QPoint(cent.x, cent.y);

stream << scenePos();

stream << side;

stream << QString::fromStdString(text);

stream << fontSize;

}

void TriangleText::***print***(std::ostream& out){

out << "Сторона треугольника " << side<<"\n";

out<<"Длина текста "<< length<<"\n";

out<<"Размер шрифта "<< fontSize<<"\n";

out<<"Текст "<< text<<"\n";

}

void TriangleText::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*option, QWidget \*widget) {

painter->setPen(QPen(Qt::black, 3, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

painter->setBrush(QColor(col.r, col.g, col.b));

QPolygon polygon; // Используем класс полигона, чтобы отрисовать треугольник

// Помещаем координаты точек в полигональную модель

auto firstPoint = figureRect.topLeft().toPoint();

auto secondPoint = QPoint(figureRect.bottomLeft().toPoint().rx() + side\*sin(M\_PI/12),figureRect.bottomRight().toPoint().ry());

auto thirdPoint = QPoint(figureRect.topRight().toPoint().rx(),figureRect.topRight().toPoint().ry() + side\*sin(M\_PI/12));

polygon << firstPoint << secondPoint << thirdPoint;

painter->drawPolygon(polygon);

painter->drawPolygon(polygon);

auto font = painter->font();

font.setPointSize(10);

painter->setFont(font);

painter->drawText(figureRect.topLeft().x() + (side\*cos(M\_PI/12))/6,figureRect.topLeft().y() + (side\*cos(M\_PI/12))/2, QString::fromStdString(text));

Q\_UNUSED(option)

Q\_UNUSED(widget)

}