МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №10**

по дисциплине: Объектно-ориентированное программирование

тема: «**Закрепление навыков программирования в объектно-ориентированном стиле. Визуальные компоненты. Знакомство с QT.»**

Выполнил: ст. группы ПВ-233

Ситников Алексей Павлович

Проверил:

Белгород 2025 г.

Вариант 3 (13)

**Цель работы:** приобретение практических навыков создания приложений на языке С++.

Создать графический редактор типа Painter, отрисовка стандартных графических примитивов, выбор цвета, толщины нажима. Сохранение в файл, загрузка из файла.

smartPointer.h

template <typename T>  
class smart\_ptr {  
 T \* obj;  
 int \*count;  
 bool useNew;  
public:  
  
 smart\_ptr(T &obj) :obj(obj), count(new int(1)){  
 useNew = false;  
 }  
  
 smart\_ptr(const smart\_ptr &s) :obj(s.obj), count(s.count), useNew(s.useNew){  
 (\*count)++;  
 }  
  
 smart\_ptr(T\* obj, bool usenew = true) : obj(obj), count(new int(1)) {  
 useNew = usenew;  
 }  
  
 smart\_ptr(smart\_ptr&& s) noexcept : obj(s.obj), count(s.count), useNew(s.useNew) {  
 s.obj = nullptr; // Обнуляем указатель в перемещаемом объекте  
 s.count = nullptr; // Обнуляем счетчик в перемещаемом объекте  
 }  
  
 ~smart\_ptr() {  
 if(count && --(\*count) == 0){  
 if(useNew){  
 delete obj;  
 }  
 delete count;  
  
 }  
 }  
  
 T& operator**\***(){  
 return \*obj;  
 }  
  
 T\* operator**->**(){  
 return obj;  
 }  
};

mainsindow.h

#ifndef **QPAINTWIDGET\_H**#define **QPAINTWIDGET\_H**#include <QWidget>  
#include <QPainter>  
#include <QPaintEvent>  
#include <QMenuBar>  
#include <QMenu>  
#include <QAction>  
#include <QVBoxLayout>  
#include <QPainterPath>  
#include <QKeyEvent>  
#include <QFile>  
#include <QPicture>  
#include <QLabel>  
#include <QFileDialog>  
#include "smartPointer.h"  
  
enum DrawMode { Line\_, Square, Rectangle, Circle, Triangle, Image};  
struct Shaps\_{  
 QRectF m\_shapes;  
 QPolygon triangle;  
 DrawMode name;  
 QVector<QPoint> points;  
 QColor color;  
 int penSize;  
};  
  
  
  
class QPaintWidget : public QWidget  
{  
 Q\_OBJECT  
public:  
 QPaintWidget(QWidget \*parent = nullptr);  
 ~QPaintWidget();  
 void setPenSize();  
  
protected:  
 void paintEvent(QPaintEvent \*event) override;  
 void mousePressEvent(QMouseEvent \*event) override;  
 void mouseMoveEvent(QMouseEvent \*event) override;  
 void mouseReleaseEvent(QMouseEvent \*event) override;  
  
private:  
 QRect QPolygonToRect(const QPolygon& polygon);  
 QPainterPath m\_path;  
 QVector<QPoint> m\_currentLine; // Текущая рисуемая линия  
 bool m\_isDrawing = false; // Флаг рисования  
 QPen m\_pen;  
 QMenuBar \*menuBar;  
 QColor m\_penColor;  
 int m\_penSize;  
 void setSquareMode();  
 void setRectangleMode();  
 void setCircleMode();  
 void setTriangleMode();  
 QVector<Shaps\_> shapes;  
 QVector<QPolygon> m\_polygons;  
 void setDrawMode(DrawMode mode);  
 bool m\_isDrawingShape;// Флаг, указывающий, что фигура рисуется  
 QPoint m\_tempStartPoint; // Начальная точка для фигуры  
 QPoint m\_tempEndPoint; // Конечная точка для фигуры  
 bool m\_isDrawingTriangle;  
 Shaps\_ newShapeTriangle; //для отрисовки треугольника  
 void EditBack();  
 void EditForward();  
 long long int lenShapes;  
 QVector<Shaps\_> shapesDeleted;  
 void keyPressEvent(QKeyEvent \*event);  
 void saveFile();  
 void openFile();  
 QPixmap pixmap;  
 QString currentFileName;  
 bool needSave;  
 void creatNewFile();  
 void promptSaveFile(smart\_ptr<bool>& flag);  
 void promptSaveFile();  
 void close();  
  
  
private slots:  
 void chooseColor(); // Слот для выбора цвета  
};  
  
#endif // QPAINTWIDGET\_H

mainwindow.cpp

#include "mainwindow.h"  
#include <QMouseEvent>  
#include <QPainter>  
#include <QColorDialog>  
#include <QInputDialog>  
#include <QMessageBox>  
  
  
  
  
DrawMode m\_drawMode;  
QPaintWidget::QPaintWidget(QWidget \*parent) : QWidget(parent)  
{  
 setAttribute(Qt::WA\_StaticContents); // Для оптимизации рисования  
 m\_pen.setColor(Qt::red);  
 m\_pen.setWidth(3);  
 m\_penColor = Qt::red;  
 m\_penSize = 3;  
 m\_drawMode = Line\_;  
 needSave = false;  
 m\_isDrawingTriangle = false;  
 lenShapes = 0;  
 currentFileName.clear();  
 // Создаем панель меню  
 QMenuBar \*menuBar = new QMenuBar(this);  
  
 // Добавляем панель меню в главное окно  
 // QVBoxLayout \*layout = new QVBoxLayout(this);  
  
 smart\_ptr<QVBoxLayout> layout(new QVBoxLayout(this));  
  
 layout->setMenuBar(menuBar);  
 layout->addWidget(this); // Добавляем ваш QPaintWidget в layout  
 smart\_ptr<QLabel> label(new QLabel(this), true);  
 label->setVisible(false);  
 layout->addWidget(label.operator->()); // Добавляем QLabel в layout  
 setLayout(layout.operator->());  
  
  
 pixmap = QPixmap();  
 // Создаем меню "Файл"  
  
  
 QMenu \*fileMenu = new QMenu("Файл", this);  
 QAction \*newAction = new QAction("Новый", this);  
 QAction \*openAction = new QAction("Открыть", this);  
 QAction \*saveAction = new QAction("Сохранить", this);  
 QAction \*exitAction = new QAction("Выход", this);  
  
 // Добавляем действия в меню "Файл"  
 fileMenu->addAction(newAction);  
 fileMenu->addAction(openAction);  
 fileMenu->addAction(saveAction);  
 fileMenu->addSeparator(); // Разделитель  
 fileMenu->addAction(exitAction);  
  
 // Добавляем меню "Файл" в панель меню  
 menuBar->addMenu(fileMenu);  
  
  
  
 // Подключаем действие выхода  
 connect(exitAction, &QAction::triggered, this, &QPaintWidget::close);  
 connect(saveAction, &QAction::triggered, this, &QPaintWidget::saveFile);  
 connect(openAction, &QAction::triggered, this, &QPaintWidget::openFile);  
 connect(newAction, &QAction::triggered, this, &QPaintWidget::creatNewFile);  
  
 QMenu \*EditMenu = new QMenu("Настройки", this);  
  
 QAction \*color = new QAction("Цвет", this);  
 QAction \*size = new QAction("размер", this);  
 QMenu \*figureMenu = new QMenu("Фигуры", this);  
 EditMenu->addAction(color);  
 EditMenu->addAction(size);  
 EditMenu->addMenu(figureMenu);  
 menuBar->addMenu(EditMenu);  
  
  
 connect(color, &QAction::triggered, this, &QPaintWidget::chooseColor);  
 connect(size, &QAction::triggered, this, &QPaintWidget::setPenSize);  
  
 QAction \*lineAction = new QAction("Линия", this);  
 QAction \*squareAction = new QAction("Квадрат", this);  
 QAction \*rectangleAction = new QAction("Прямоугольник", this);  
 QAction \*circleAction = new QAction("Круг", this);  
 QAction \*triangleAction = new QAction("Треугольник", this);  
  
 figureMenu->addAction(lineAction);  
 figureMenu->addAction(squareAction);  
 figureMenu->addAction(rectangleAction);  
 figureMenu->addAction(circleAction);  
 figureMenu->addAction(triangleAction);  
  
 DrawMode l = Line\_;  
 DrawMode s = Line\_;  
 DrawMode r = Line\_;  
 DrawMode c = Line\_;  
 DrawMode t = Line\_;  
  
 connect(lineAction, &QAction::triggered, this, [=]() { setDrawMode(Line\_); });  
 connect(squareAction, &QAction::triggered, this, [=]() { setDrawMode(Square); });  
 connect(rectangleAction, &QAction::triggered, this, [=]() { setDrawMode(Rectangle); });  
 connect(circleAction, &QAction::triggered, this, [=]() { setDrawMode(Circle); });  
 connect(triangleAction, &QAction::triggered, this, [=]() { setDrawMode(Triangle); });  
  
  
 QMenu \*Edit = new QMenu("Правка", this);  
 QAction \*back = new QAction("Вернуть назад", this);  
 QAction \*forward = new QAction("Вернуть вперёд", this);  
 Edit->addAction(back);  
 Edit->addAction(forward);  
 menuBar->addMenu(Edit);  
  
 connect(back, &QAction::triggered, this, QPaintWidget::EditBack);  
 connect(forward, &QAction::triggered, this, QPaintWidget::EditForward);  
  
 resize(800, 600);  
 m\_isDrawingShape = false;  
}  
  
QPaintWidget::~QPaintWidget() {}  
  
void QPaintWidget::paintEvent(QPaintEvent \*event)  
{  
 QPainter painter(this);  
 painter.setRenderHint(QPainter::Antialiasing);  
  
 // Рисуем все сохраненные линии  
 for (const auto& line : shapes) {  
 painter.setPen(QPen(line.color, line.penSize)); // Используем цвет и размер линии  
 painter.drawPolyline(line.points.constData(), line.points.size());  
 }  
  
 // Рисуем текущую линию  
 if (m\_isDrawing && m\_currentLine.size() > 1 && m\_drawMode == Line\_) {  
 painter.setPen(QPen(m\_penColor, m\_penSize)); // Используем текущий цвет и размер  
 painter.drawPolyline(m\_currentLine.constData(), m\_currentLine.size());  
 }  
  
  
 // Рисуем текущую фигуру, если она рисуется  
 if (m\_isDrawingShape) {  
 painter.setPen(QPen(m\_penColor, m\_penSize)); // Используем текущий цвет и размер  
 if (m\_drawMode == Square) {  
 int width = m\_tempEndPoint.x() - m\_tempStartPoint.x();  
 int height = m\_tempEndPoint.y() - m\_tempStartPoint.y();  
 int size = std::min(width, height);  
 QRectF square(m\_tempStartPoint, QSize(size, size));  
 painter.drawRect(square);  
 } else if (m\_drawMode == Rectangle) {  
 QRectF rectangle(m\_tempStartPoint, m\_tempEndPoint);  
 painter.drawRect(rectangle);  
 } else if (m\_drawMode == Circle) {  
 QPointF center((m\_tempStartPoint.x() + m\_tempEndPoint.x()) / 2, (m\_tempStartPoint.y() + m\_tempEndPoint.y()) / 2);  
 double radius = std::hypot(m\_tempEndPoint.x() - m\_tempStartPoint.x(), m\_tempEndPoint.y() - m\_tempStartPoint.y()) / 2.0;  
 QRectF circle(center.x() - radius, center.y() - radius, radius \* 2, radius \* 2);  
 painter.drawEllipse(circle);  
 } else if (m\_drawMode == Triangle) {  
 QPolygon triangle;  
 int sideLength = std::hypot(m\_tempEndPoint.x() - m\_tempStartPoint.x(), m\_tempEndPoint.y() - m\_tempStartPoint.y());  
 triangle << m\_tempStartPoint  
 << m\_tempEndPoint  
 << QPoint(m\_tempStartPoint.x() + sideLength / 2, m\_tempStartPoint.y() - sideLength);  
 painter.drawPolygon(triangle);  
 newShapeTriangle.triangle = triangle;  
 newShapeTriangle.name = Triangle;  
 newShapeTriangle.penSize = m\_penSize;  
 newShapeTriangle.color = m\_penColor;  
 m\_isDrawingTriangle = true;  
 }  
  
 }  
  
 // Рисуем все сохраненные фигуры  
  
 for (const auto& shape : shapes) {  
 painter.setPen(QPen(shape.color, shape.penSize));  
 // Проверяем, является ли фигура кругом  
 if (shape.name == Circle) {  
 painter.drawEllipse(shape.m\_shapes); // Рисуем круг  
 } else if (shape.name == Triangle) {  
 painter.drawPolygon(shape.triangle); // Рисуем треугольник  
 } else {  
 painter.drawRect(shape.m\_shapes); // Рисуем прямоугольник или квадрат  
 }  
 }  
}  
void QPaintWidget::mousePressEvent(QMouseEvent \*event)  
{  
 if (event->button() == Qt::LeftButton) {  
 m\_currentLine.clear();  
 m\_currentLine << event->pos();  
 m\_isDrawing = true;  
  
 m\_tempStartPoint = event->pos();  
 m\_tempEndPoint = event->pos();  
 m\_isDrawingShape = true; // Начинаем рисовать фигуру  
 }  
}  
  
void QPaintWidget::mouseMoveEvent(QMouseEvent \*event)  
{  
 if (m\_isDrawing) {  
 m\_currentLine << event->pos();  
 update(); // Перерисовываем виджет  
 }  
  
 if (m\_isDrawingShape) {  
 m\_tempEndPoint = event->pos(); // Обновляем конечную точку для фигуры  
 update(); // Перерисовываем виджет  
 }  
}  
  
void QPaintWidget::mouseReleaseEvent(QMouseEvent \*event)  
{  
 if (event->button() == Qt::LeftButton) {  
 if(m\_drawMode == Triangle && m\_isDrawingTriangle){  
 shapes.push\_back(newShapeTriangle);  
 m\_isDrawingTriangle = false;  
  
 }  
 }  
 needSave = true;  
 shapesDeleted.clear();  
  
 Shaps\_ newShape;  
 if (m\_isDrawing) {  
 if (m\_drawMode == Line\_) {  
 if(m\_currentLine.length() > 1){  
 newShape.name = Line\_;  
 newShape.points = m\_currentLine;  
 newShape.color = m\_penColor;  
 newShape.penSize = m\_penSize;  
 shapes.push\_back(newShape);  
 }  
 } else {  
 // Получаем координаты начала и конца рисования  
 QPoint startPoint = m\_currentLine.first();  
 QPoint endPoint = event->pos();  
  
 if (m\_drawMode == Square) {  
 // Рисуем квадрат  
 int width = endPoint.x() - startPoint.x();  
 int height = endPoint.y() - startPoint.y();  
 int size = std::min(width, height); // Используем минимальное значение  
 QRectF square(startPoint, QSize(size, size));  
 newShape.m\_shapes = square;  
 newShape.name = Square;  
 newShape.penSize = m\_penSize;  
 newShape.color = m\_penColor;  
 shapes.push\_back(newShape);  
  
 } else if (m\_drawMode == Rectangle) {  
 // Рисуем прямоугольник  
 QRectF rectangle(startPoint, endPoint);  
 newShape.m\_shapes = rectangle;  
 newShape.name = Rectangle;  
 newShape.penSize = m\_penSize;  
 newShape.color = m\_penColor;  
 shapes.push\_back(newShape);  
 } else if (m\_drawMode == Circle) {  
 // Рисуем круг  
 QPointF center((startPoint.x() + endPoint.x()) / 2, (startPoint.y() + endPoint.y()) / 2);  
 double radius = std::hypot(endPoint.x() - startPoint.x(), endPoint.y() - startPoint.y()) / 2.0;  
 QRectF circle(center.x() - radius, center.y() - radius, radius \* 2, radius \* 2);  
 newShape.m\_shapes = circle;  
 newShape.name = Circle;  
 newShape.penSize = m\_penSize;  
 newShape.color = m\_penColor;  
 shapes.push\_back(newShape);  
 }  
 }  
 m\_currentLine.clear();  
 m\_isDrawing = false;  
 m\_isDrawingShape = false;  
 update();  
 }  
}  
  
void QPaintWidget::keyPressEvent(QKeyEvent \*event) {  
 if (event->key() == Qt::Key\_Z && event->modifiers() & Qt::ControlModifier) {  
 EditBack();  
 }  
 else if(event->key() == Qt::Key\_Y && event->modifiers() & Qt::ControlModifier){  
 EditForward();  
 }  
 else if(event->key() == Qt::Key\_S && event->modifiers() & Qt::ControlModifier){  
 saveFile();  
 }  
}  
  
void QPaintWidget::chooseColor() {  
 // Открываем диалог выбора цвета  
 QColor color = QColorDialog::getColor(m\_penColor, this, "Выберите цвет");  
 if (color.isValid()) {  
 m\_penColor = color; // Обновляем цвет пера  
 update(); // Перерисовываем виджет  
 }  
}  
void QPaintWidget::setPenSize() {  
 bool ok;  
 int size = QInputDialog::getInt(this, tr("Выберите размер пера"), tr("Размер пера:"), m\_penSize, 1, 100, 1, &ok);  
 if (ok) { // Проверяем, нажал ли пользователь "ОК"  
 m\_penSize = size; // Устанавливаем новый размер пера  
 update(); // Обновляем виджет  
 }  
}  
  
  
  
void QPaintWidget::setDrawMode(DrawMode mode) {  
 m\_drawMode = mode;  
 // Сброс временных точек при смене режима  
 m\_tempStartPoint = QPoint(); // или (0, 0)  
 m\_tempEndPoint = QPoint(); // или (0, 0)  
 update(); // Перерисовываем виджет, чтобы обновить режим рисования  
}  
  
void QPaintWidget::EditBack(){  
 if(shapes.length() != 0){  
 needSave = true;  
 shapesDeleted.push\_back(shapes.back());  
 shapes.pop\_back();  
 lenShapes = shapes.length();  
 update();  
 }  
}  
  
void QPaintWidget::EditForward(){  
 if(lenShapes == shapes.length()){  
 if(shapesDeleted.length() != 0){  
 needSave = true;  
 shapes.push\_back(shapesDeleted.back());  
 shapesDeleted.pop\_back();  
 lenShapes++;  
 update();  
 }  
 }  
}  
  
  
void QPaintWidget::saveFile() {  
  
 QString fileName;  
 if (!currentFileName.isEmpty()) {  
 fileName = currentFileName;  
 }  
 else{  
 fileName = QFileDialog::getSaveFileName(this, tr("Сохранить файл"), "", tr("Text Files (\*.txt)"));  
 if (fileName.isEmpty()) return;  
 }  
  
 QFile file(fileName);  
 if (!file.open(QIODevice::WriteOnly)) {  
 QMessageBox::warning(this, tr("Ошибка"), tr("Не удалось открыть файл для записи."));  
 return;  
 }  
  
 QDataStream out(&file);  
 out.setVersion(QDataStream::Qt\_5\_0);  
 out.setByteOrder(QDataStream::LittleEndian);  
  
 // Сохраняем количество фигур  
 out << (qint32)shapes.size();  
  
 for (const Shaps\_& shape : shapes) {  
 // Сохраняем основные параметры  
 out << (qint8)shape.name;  
 out << shape.color.rgba();  
 out << (qint32)shape.penSize;  
  
 // Сохраняем специфичные данные для каждого типа фигуры  
 switch (shape.name) {  
 case Line\_:  
 out << shape.points; // Для линии - массив точек  
 break;  
 case Square:  
 case Rectangle:  
 case Circle:  
 out << shape.m\_shapes; // Для геометрических фигур - QRectF  
 break;  
 case Triangle:  
 out << shape.triangle; // Для треугольника - QPolygon  
 break;  
 }  
 }  
 file.close();  
 needSave = false;  
}  
  
void QPaintWidget::openFile() {  
 if(needSave){  
 promptSaveFile();  
 }  
 QString fileName = QFileDialog::getOpenFileName(this, tr("Открыть файл"), "", tr("Text Files (\*.txt)"));  
 if (fileName.isEmpty()) return;  
  
 QFile file(fileName);  
 if (!file.open(QIODevice::ReadOnly)) {  
 QMessageBox::warning(this, tr("Ошибка"), tr("Не удалось открыть файл для чтения."));  
 return;  
 }  
  
 QDataStream in(&file);  
 in.setVersion(QDataStream::Qt\_5\_0);  
 in.setByteOrder(QDataStream::LittleEndian);  
  
 shapes.clear();  
  
 qint32 shapeCount;  
 in >> shapeCount;  
 currentFileName = fileName;  
 for (qint32 i = 0; i < shapeCount; ++i) {  
 Shaps\_ shape;  
  
 // Читаем основные параметры  
 qint8 shapeType;  
 in >> shapeType;  
 shape.name = (DrawMode)shapeType;  
  
 QRgb color;  
 in >> color;  
 shape.color = QColor::fromRgba(color);  
  
 qint32 penSize;  
 in >> penSize;  
 shape.penSize = penSize;  
  
 // Читаем специфичные данные для каждого типа фигуры  
 switch (shape.name) {  
 case Line\_:  
 in >> shape.points;  
 break;  
 case Square:  
 case Rectangle:  
 case Circle:  
 in >> shape.m\_shapes;  
 break;  
 case Triangle:  
 in >> shape.triangle;  
 break;  
 }  
  
 shapes.append(shape);  
 }  
  
 file.close();  
 update();  
}  
  
void QPaintWidget::creatNewFile(){  
 if(needSave){  
 promptSaveFile();  
 }  
 shapes.clear();  
 update();  
 currentFileName.clear();  
}  
  
  
void QPaintWidget::promptSaveFile(smart\_ptr<bool>& flag) {  
 QMessageBox::StandardButton reply = QMessageBox::question(  
 this,  
 tr("Сохранить файл"),  
 tr("Вы хотите сохранить изменения перед закрытием?"),  
 QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel  
 );  
  
 if (reply == QMessageBox::Save) {  
 saveFile();  
 } else {  
 \*flag = false;  
 }  
}  
  
  
void QPaintWidget::promptSaveFile() {  
 QMessageBox::StandardButton reply = QMessageBox::question(  
 this,  
 tr("Сохранить файл"),  
 tr("Вы хотите сохранить изменения перед закрытием?"),  
 QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel  
 );  
  
 if (reply == QMessageBox::Save) {  
 saveFile();  
 }  
}  
  
  
void QPaintWidget::close()  
{  
 smart\_ptr<bool> flag(new bool(true));  
 if(needSave){  
 promptSaveFile(flag);  
 }  
 if(\*flag){  
 QWidget::close();  
 }  
}

main.cpp

#include "mainwindow.h"  
  
#include <QApplication>  
  
int main(int argc, char \*argv[])  
{  
 QApplication a(argc, argv);  
 QPaintWidget w;  
 w.show();  
 return a.exec();  
}

[Ссылка на гит.](https://gitlab.com/asitnikov770/2025_pv233_sitnikovalekseipavlovich_13/-/tree/main?ref_type=heads)

Изображение выглядит как снимок экрана, диаграмма, дизайн

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, программное обеспечение, Значок на компьютере, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

**Вывод:** в ходе проделанной работы я преобрёл навыки в работе с QT.