Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Кафедра «Компьютерная безопасность»

Отчёт к лабораторной работе №3 (2 год)

по дисциплине

«Языки программирования»

Работу выполнил

Студент группы СКБ221 _____ М. И. Нугманов

подпись, дата

Работу проверил _____ С. А. Булгаков

подпись, дата

Москва 2023

Содержание

Постановка задачи ... 3
1. Алгоритм решения ... 4
2. Реализация и её отличие от Лабораторной работы №2 ... 4
2.1 Методы класса и отличие их реализации от Лабораторной работы №2 ... 5
2.1.1 paintEvent() ... 5
2.1.2 drawRectangle() ... 5
2.1.3 drawCircle() ... 6
2.1.4 InBorderRectangle(const QPoint& clickPoint) ... 6
2.1.5 InBorderCircle(const QPoint& clickPoint) ... 6
2.1.6 mousePressEvent(QMouseEvent *event) ... 7
2.1.7 ShowContextMenu(const QPoint &pos) ... 7
2.1.8 mouseMoveEvent(QMouseEvent *event) ... 7
2.1.9 mouseReleaseEvent(QMouseEvent *event) ... 7
2.1.10 addNewRectangle() ... 8
2.1.11 addNewCircle() ... 8
2.1.12 changeAngle() ... 8
2.1.13 rotate(QPainter& painter, QPainterPath& path, const qreal angle) ... 9
2.1.14 scale() ... 9
2.2 Контекстное меню ... 10
2.2.1 Нажатие ПКМ вне границ фигуры или во внутренней ее части ... 10
2.2.2 Нажатие ПКМ на границе фигуры ... 10
3. Тестирование программы ... 11
Листинг 1 ... 15
1.1 Исходный код widget.h ... 15
1.2 Исходный код widget.cpp ... 16
1.3 Исходный код main.cpp ... 20

Постановка задачи

Разработать программу с использованием библиотеки *Qt*. В окне программы реализовать возможность добавления геометрической фигуры посредством контекстного меню. Реализовать перемещение фигуры в рамках окна при перетаскивании ее за границу курсором мыши. При нажатии на границу фигуры правой кнопкой мыши выводить контекстное меню позволяющее повернуть ее или изменить размер. При наведении курсора на внутреннюю часть фигуры подсвечивать ее полупрозрачным цветом.

Для реализации фигуры использовать класс *QWidget* и возможности классов *QPainter* и *QPainterPath*.

1. Алгоритм решения

Для решения поставленной задачи был создан класс *Widget* – наследник класса *QWidget*, в котором реализованы методы для создания фигур, их поворота на 15 градусов, увеличения размера на 20%, а также вывода контекстного меню, позволяющего пользователю выбирать вышеперечисленные действия.

2. Реализация и её отличие от Лабораторной работы №2

В классе *Widget* реализованы все необходимые методы для взаимодействия с фигурами. Отдельно была создана структура *Shape*, содержащая информацию об фигуре. Также реализовано контекстное меню при помощи соединения *customContextMenuRequested* со слотом *ShowContextMenu*. Меню позволяет создавать и изменять фигуры. Подробнее с реализацией программы можно ознакомиться в Листинге 1.

Отличия в реализации от Лабораторной работы №2 заключаются в том, что для отрисовки фигуры используется не только возможности класса *QPainter*, но и возможности класса *QPainterPath*, который позволяет задать путь для отрисовки фигуры, который уже в дальнейшем может быть нарисован при помощи класса *QPainter*. Так, в структуре *Shape* информация о типе фигуры представлена с помощью *QPainterPath*, в котором может быть описан любой контур фигуры, а не *QRect*, который изначально представляет из себя прямоугольник, но может быть отрисован в иную фигуру с помощью класса *QPainter*. Из-за того, что в классе *QPainterPath* реализованы другие методы для работы с фигурами, методы из Лабораторной работы №2 были доработаны. Далее будут рассмотрены все методы класса *Widget*, и отличие их реализации от Лабораторной работы №2.

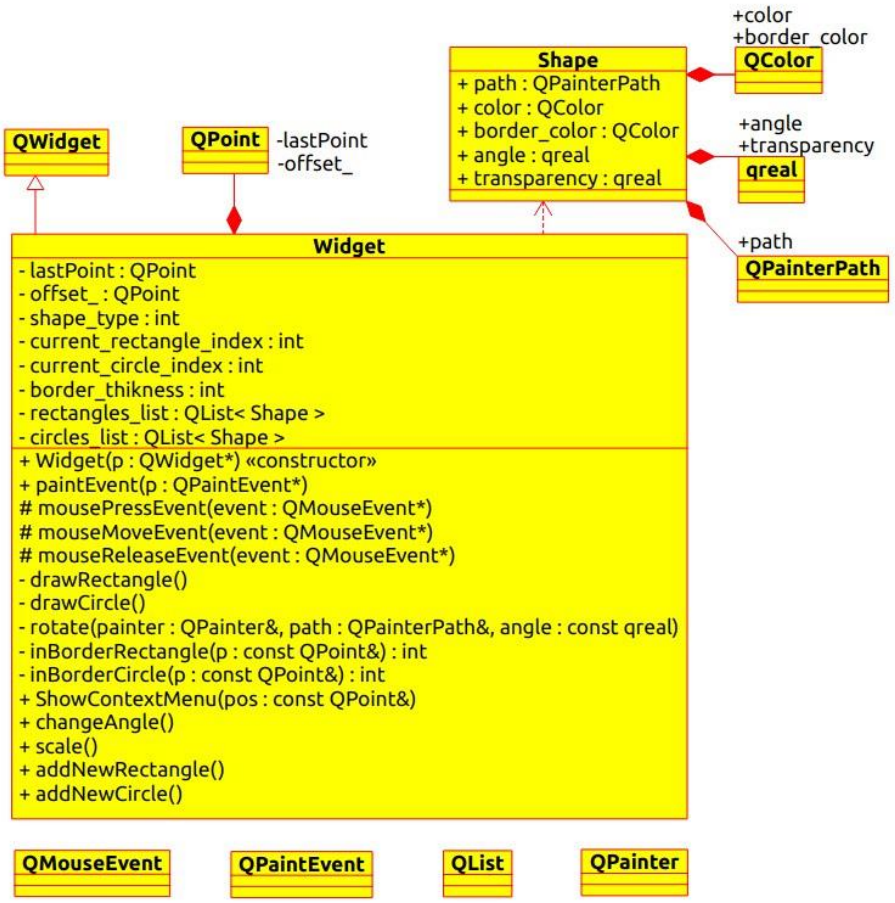

Рисунок 1 - UML-диаграмма widget.h

2.1 Методы класса и отличие их реализации от Лабораторной работы №2

2.1.1 paintEvent()

Событие отрисовки фигур. Вызывает методы *drawRectangle()* и *drawCircle()* для отображения фигур на виджете.

2.1.2 drawRectangle()

Предназначен для отрисовки всех прямоугольников из списка *rectangles_list*. Использует объект *QPainter* для отрисовки пути фигуры, представляюшего из себя прямоугольник, с учетом их цвета, угла и координат.

Отличия заключаются в следующем: метод проверяет положение курсора относительно каждой фигуры из списка – если курсор находится внутри фигуры, то

она станет полупрозрачной, если курсор находится на границе фигуры, то граница подсветится черным цветом.

2.1.3 drawCircle()

Предназначен для отрисовки всех кругов из списка *circles_list*. Использует объект *QPainter* для отрисовки пути фигуры, представляющего из себя круг, с учетом их цвета, угла и координат.

Отличия заключаются в следующем: метод проверяет положение курсора относительно каждой фигуры из списка – если курсор находится внутри фигуры, то она станет полупрозрачной, если курсор находится на границе фигуры, то граница подсветится черным цветом.

2.1.4 InBorderRectangle(const QPoint& clickPoint)

Проверяет, находится ли точка *clickPoint* на границе какого-либо прямоугольника из списка *rectangles_list*. Возвращает индекс прямоугольника, если точка находится на его границе, или -1, если не находится.

Отличие от метода *inRectangle(QPoint clickPoint)* из Лабораторной работы №2 заключается в том, что последний проверяет, находится ли точка *clickPoint* внутри самой фигуры.

2.1.5 InBorderCircle(const QPoint& clickPoint)

Проверяет, находится ли точка *clickPoint* на границе какого-либо круга из списка *circles_list*. Возвращает индекс круга, если точка находится на ее границе, или -1, если не находится.

Отличие от метода *inCircle(QPoint clickPoint)* из Лабораторной работы №2 заключается в том, что последний проверяет, находится ли точка *clickPoint* внутри самой фигуры.

2.1.6 mousePressEvent(QMouseEvent *event)

Обрабатывает событие нажатия кнопки мыши. Определяет, на границу какого типа фигуры указывает точка *clickPoint* и сохраняет этот тип в *shape_type*. Запоминает индекс выбранной фигуры, если таковая имеется.

Отличие заключается в том, что теперь тип фигуры сохраняется только если точка *clickPoint* находится именно на границе какой-либо фигуры.

2.1.7 ShowContextMenu(const QPoint &pos)

Показывает контекстное меню, связанное с выбранной фигурой (*rectangles_list* или *circles_list*) или меню для создания новой фигуры, если фигура не выбрана. Добавляет действия для вращения и изменения размера фигур в меню.

Отличие заключается в том, что контекстное меню с параметрами изменения фигуры показывается только в случае, если правая кнопка мыши была нажата на границе фигуры. В противном случае вызывается контекстное меню с параметрами добавления новых фигур.

2.1.8 mouseMoveEvent(QMouseEvent *event)

Обрабатывает событие перемещения мыши с зажатой кнопкой. Перемещает выбранную фигуру (прямоугольник или круг) на величину изменения координат указателя мыши.

Отличие заключается в том, что перемещение возможно только в случае, если курсор мыши находится на границе фигуры.

2.1.9 mouseReleaseEvent(QMouseEvent *event)

Обрабатывает событие отпускания кнопки мыши. Сбрасывает *shape_type* после того, как кнопка мыши была отпущена.

2.1.10 addNewRectangle()

Добавляет новый объект структуры Shape в виде прямоугольника в список *rectangles_list* с начальными параметрами, такими как координаты, размер и случайный цвет. Вызывает *update(),* чтобы обновить отображение.

Отличие заключается в том, что теперь в структуре *Shape* теперь следующие поля: *path* (объект класса *QPainterPath,* хранит контур фигуры, в последствии отрисовывается с помощью объекта класса *QPainter*); *color* (объект класса *QColor*, хранит информацию о цвете внутренней части фигуры); *border_color* (объект класса *QColor*, хранит информацию о цвете границы фигуры); *angle* (число типа *qreal* (*typedef* для *double*), хранит информацию об угле, на который повернута фигура); *transparency* (число типа *qreal* (*typedef* для *double*), хранит информацию о коэффициенте прозрачности цвета фигуры).

2.1.11 addNewCircle()

Добавляет новый круг в список *circles_list* с начальными параметрами, такими как координаты, размер и случайный цвет. Вызывает *update(),* чтобы обновить отображение.

Отличие заключается в том, что теперь в структуре *Shape* теперь следующие поля: *path* (объект класса *QPainterPath,* хранит контур фигуры, в последствии отрисовывается с помощью объекта класса *QPainter*); *color* (объект класса *QColor*, хранит информацию о цвете внутренней части фигуры); *border_color* (объект класса *QColor*, хранит информацию о цвете границы фигуры); *angle* (число типа *qreal* (*typedef* для *double*), хранит информацию об угле, на который повернута фигура); *transparency* (число типа *qreal* (*typedef* для *double*), хранит информацию о коэффициенте прозрачности цвета фигуры).

2.1.12 changeAngle()

Изменяет угол поворота выбранной фигуры (прямоугольника или круга) на 15 градусов. Вызывает *update(),* чтобы обновить отображение.

2.1.13rotate(QPainter& painter, QPainterPath& path, const qreal angle)

Производит вращение фигуры, находящейся внутри *painter*, вокруг ее центра на заданный угол. Вызывает *update(),* чтобы обновить отображение.

Отличие заключается в том, что поворот производится для объекта класса *QPainterPath.*

2.1.14 scale()

Увеличивает размер выбранной фигуры (прямоугольника или круга), добавляя 20% к ее ширине и высоте. Вызывает *update(),* чтобы обновить отображение.

Отличие заключается в том, что изменяется в размерах объект класса *QPainterPath,* при помощи объекта класса *QTransform.*

2.2 Контекстное меню

2.2.1 Нажатие ПКМ вне границ фигуры или во внутренней ее части

При нажатии ПКМ вне границ фигуры или во внутренней ее части появляется контекстное меню с возможность добавить новый прямоугольник или новый круг.

2.2.2 Нажатие ПКМ на границе фигуры

При нажатии ПКМ на границе фигуры появляется контекстное меню с возможностью повернуть выбраннную фигуру на 15 градусов по часовой стрелке или увеличить Высоту и Ширину выбранной фигуры на 20%.

3. Тестирование программы

На рисунках 2–9 приведены результаты тестирования программы:

Рисунок 2 - Начальный вид программы при ее запуске

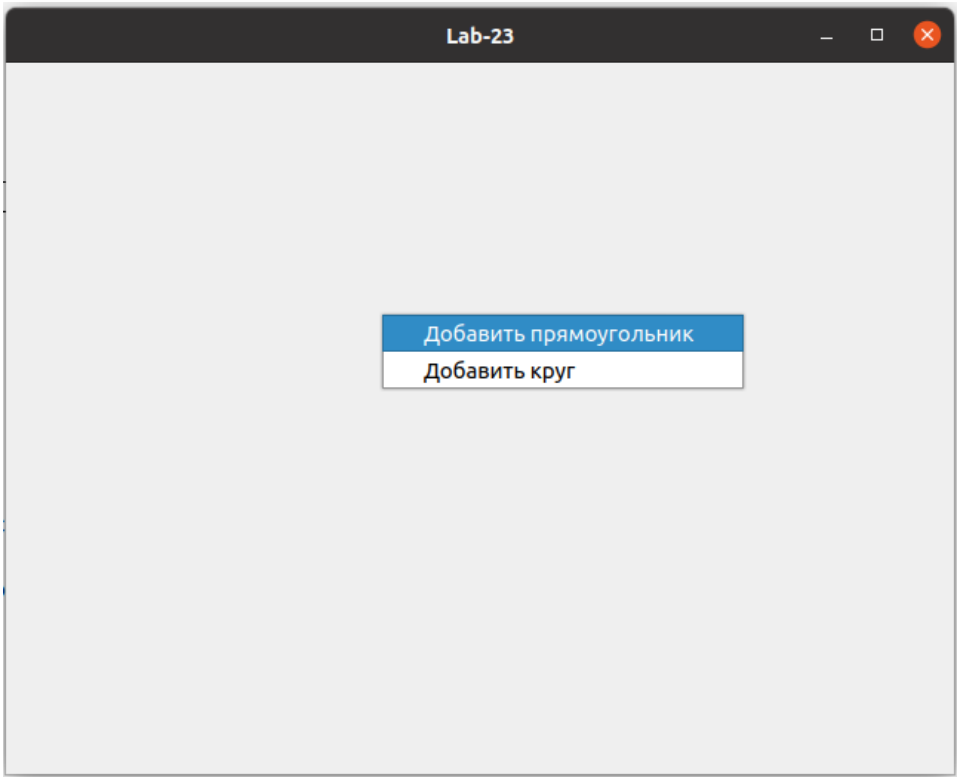

Рисунок 3 - Отображение контекстного меню для создания новой фигуры

Рисунок 4 - Результат нажатия на поле "Добавить круг"

Рисунок 5 - Вид, если курсор вне фигуры

Рисунок 6 - Вид, если курсор внутри фигуры

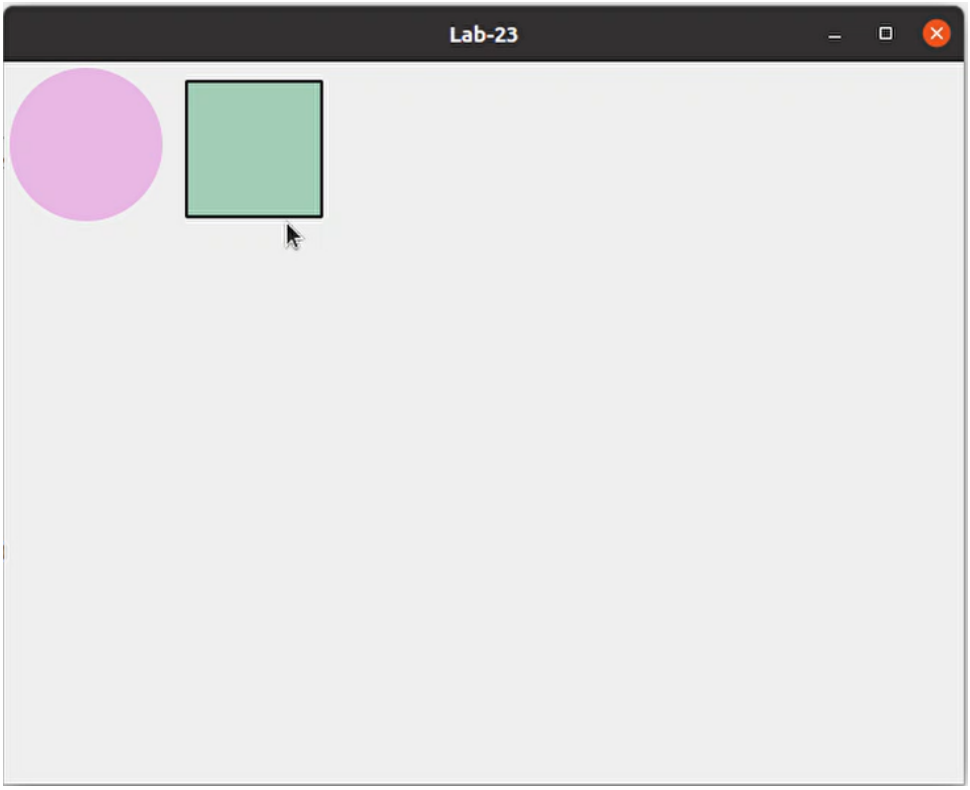

Рисунок 7 - Вид, если курсор на границе фигуры

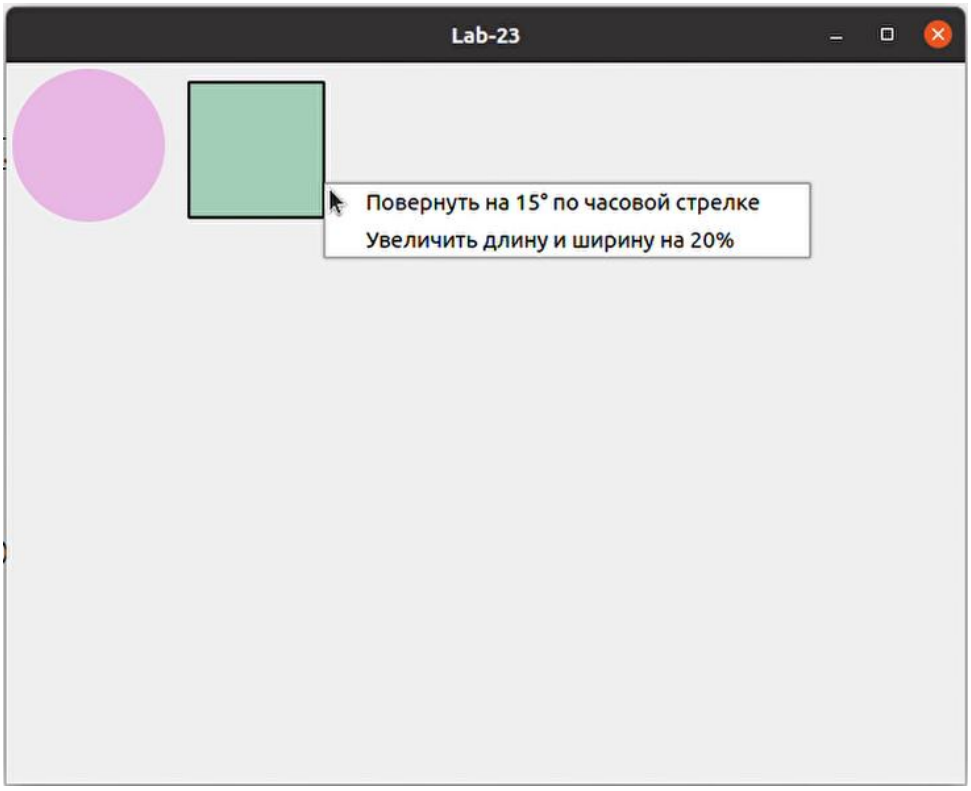

Рисунок 8 - Отображение контекстного меню для изменения фигуры при нажатии ПКМ на границе фигуры

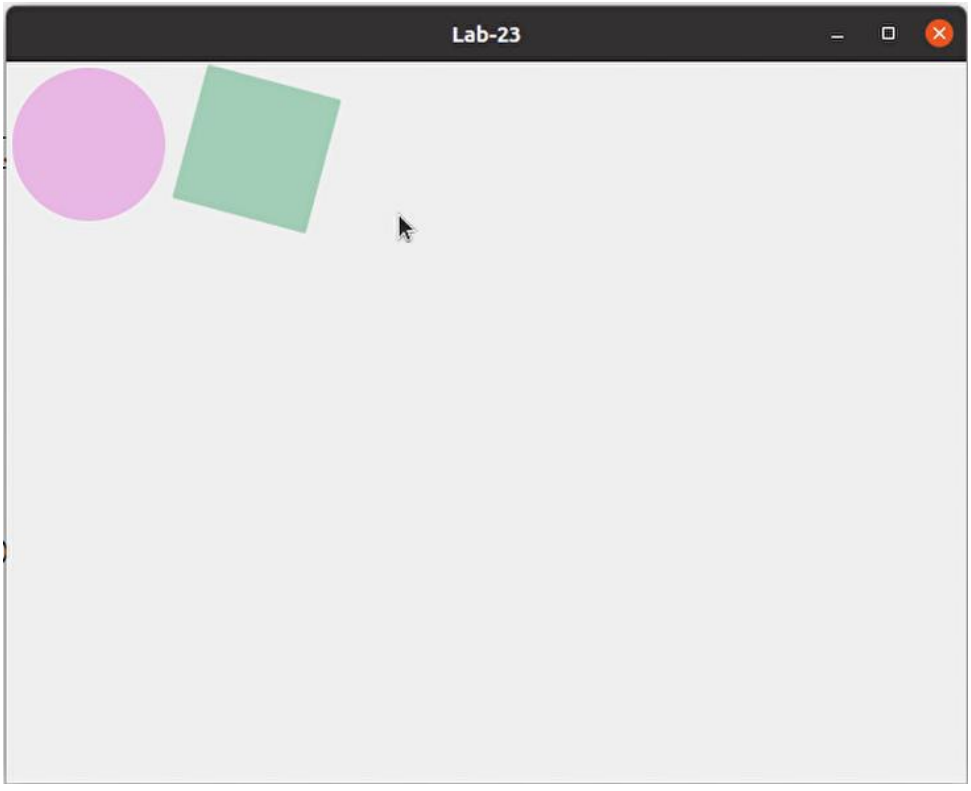

Рисунок 9 - Результат применения действия "Повернуть на 15° по часовой стрелке"

Листинг 1

1.1 Исходный код widget.h

```
#include <QWidget>
#include <QPainter>
#include <QMouseEvent>

struct Shape {
    QPainterPath path;
    QColor       color;
    QColor       border_color = color;
    qreal        angle        = 0;
    qreal        transparency = 1.0;
};

class Widget: public QWidget {
    Q_OBJECT
    public:
        Widget(QWidget* p = nullptr);
        void paintEvent(QPaintEvent *p = nullptr);

    protected:
        void mousePressEvent(QMouseEvent *event)   override;
        void mouseMoveEvent(QMouseEvent *event)    override;
        void mouseReleaseEvent(QMouseEvent *event) override;

    private:
        void drawRectangle();
        void drawCircle();
        void rotate(QPainter& painter, QPainterPath& path, const qreal angle);
        int inBorderRectangle(const QPoint& p);
        int inBorderCircle(const    QPoint& p);

        QPoint lastPoint;
        QPoint offset_;
        int shape_type;
        int current_rectangle_index = 0;
        int current_circle_index    = 0;
        int border_thikness         = 2;

        QList<Shape> rectangles_list = {};
        QList<Shape> circles_list    = {};

    public slots:
        void ShowContextMenu(const QPoint &pos);
        void changeAngle();
        void scale();
        void addNewRectangle();
        void addNewCircle();
};
```

1.2 Исходный код widget.cpp

```
#include "widget.h"
#include <QPainterPath>
#include <QDebug>
#include <QMenu>
#include <QAction>
#include <QRandomGenerator>

Widget::Widget(QWidget *parent): QWidget(parent) {
    this->setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, SIGNAL(customContextMenuRequested(const QPoint &)),
        this, SLOT(ShowContextMenu(const QPoint &)));
}

void Widget::paintEvent(QPaintEvent *) {
    drawRectangle();
    drawCircle();
}

void Widget::drawRectangle() {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);
    for(Shape &shape : rectangles_list) {
        painter.save();
        rotate(painter, shape.path, shape.angle);
        QPainterPathStroker stroker;
        stroker.setWidth(border_thikness);
        QPainterPath borderedPath = stroker.createStroke(shape.path);
        if(shape.path.contains(mapFromGlobal(QCursor::pos()))) {
            shape.transparency = 0.5;
            shape.border_color = shape.color;
            shape.border_color.setAlphaF(0.5);
        }
        else if(borderedPath.contains(mapFromGlobal(QCursor::pos()))) {
            shape.transparency = 1.0;
            shape.border_color = Qt::black;
        }
        else {
            shape.transparency = 1.0;
            shape.border_color = shape.color;
            shape.border_color.setAlphaF(1.0);
        }
        QColor color = shape.color;
        color.setAlphaF(shape.transparency);
        painter.setBrush(color);
        painter.setPen(QPen(shape.border_color, border_thikness));
        painter.drawPath(shape.path);
        painter.restore();
        update();
    }
}

void Widget::drawCircle() {
    QPainter painter(this);
    painter.setRenderHint(QPainter::Antialiasing);
    for(Shape &shape : circles_list) {
        painter.save();
        rotate(painter, shape.path, shape.angle);
        QPainterPathStroker stroker;
        stroker.setWidth(border_thikness);
        QPainterPath borderedPath = stroker.createStroke(shape.path);
```

```
        if(shape.path.contains(mapFromGlobal(QCursor::pos()))) {
            shape.transparency = 0.5;
            shape.border_color = shape.color;
            shape.border_color.setAlphaF(0.5);
        }
        else if(borderedPath.contains(mapFromGlobal(QCursor::pos()))) {
            shape.transparency = 1.0;
            shape.border_color = Qt::black;
        }
        else {
            shape.transparency = 1.0;
            shape.border_color = shape.color;
            shape.border_color.setAlphaF(1.0);
        }
        QColor color = shape.color;
        color.setAlphaF(shape.transparency);
        painter.setBrush(color);
        painter.setPen(QPen(shape.border_color, border_thikness));
        painter.drawPath(shape.path);
        painter.restore();
        update();
    }
}

void Widget::addNewRectangle() {
    Shape newRectangle;
    newRectangle.path.addRect(10, 10, 90, 90);
    newRectangle.color = QColor(QRandomGenerator::global()->bounded(256),
QRandomGenerator::global()->bounded(256), QRandomGenerator::global()->bounded(256));
    rectangles_list.append(newRectangle);
    update();
}

void Widget::addNewCircle() {
    Shape newCircle;
    newCircle.path.addEllipse(QPoint(35, 35), 50, 50);
    newCircle.color = QColor(QRandomGenerator::global()->bounded(256),
QRandomGenerator::global()->bounded(256), QRandomGenerator::global()->bounded(256));
    circles_list.append(newCircle);
    update();
}

int Widget::inBorderRectangle(const QPoint& clickPoint) {
    for (int i = 0; i < rectangles_list.size(); ++i) {
        QPainterPathStroker stroker;
        stroker.setWidth(border_thikness);
        QPainterPath borderedPath = stroker.createStroke(rectangles_list[i].path);
        if (borderedPath.contains(clickPoint))
            return i;
    }
    return -1;
}

int Widget::inBorderCircle(const QPoint& clickPoint) {
    for (int i = 0; i < circles_list.size(); ++i) {
        QPainterPathStroker stroker;
        stroker.setWidth(border_thikness);
        QPainterPath borderedPath = stroker.createStroke(circles_list[i].path);
        if (borderedPath.contains(clickPoint))
            return i;
    }
    return -1;
```

```
}

void Widget::ShowContextMenu(const QPoint &pos) {
    if(inBorderRectangle(pos) != -1)
        shape_type = 1;
    else if(inBorderCircle(pos) != -1)
        shape_type = 2;
    else
        shape_type = 0;

    if(shape_type) {
        QMenu contextMenu(tr("Контекстное меню"), this);

        QAction action1("Повернуть на 15° по часовой стрелке", this);
        connect(&action1, SIGNAL(triggered()), this, SLOT(changeAngle()));

        QAction action2("Увеличить длину и ширину на 20%", this);
        connect(&action2, SIGNAL(triggered()), this, SLOT(scale()));

        contextMenu.addAction(&action1);
        contextMenu.addAction(&action2);
        contextMenu.exec(mapToGlobal(pos));
    }
    else {
        QMenu contextMenu(tr("Контекстное меню"), this);

        QAction action3("Добавить прямоугольник", this);
        connect(&action3, SIGNAL(triggered()), this, SLOT(addNewRectangle()));

        QAction action4("Добавить круг", this);
        connect(&action4, SIGNAL(triggered()), this, SLOT(addNewCircle()));

        contextMenu.addAction(&action3);
        contextMenu.addAction(&action4);
        contextMenu.exec(mapToGlobal(pos));
    }
}

void Widget::mousePressEvent(QMouseEvent *event) {
    QPoint clickPoint = event->pos();
    if(inBorderRectangle(clickPoint) != -1) {
        shape_type = 1;
        current_rectangle_index = inBorderRectangle(clickPoint);
    }
    else if(inBorderCircle(clickPoint) != -1) {
        shape_type = 2;
        current_circle_index = inBorderCircle(clickPoint);
    }
    else
        shape_type = 0;

    if(shape_type && event->button() == Qt::LeftButton) {
        lastPoint = clickPoint;
        offset_ = QPoint(0, 0);
    }
}

void Widget::mouseMoveEvent(QMouseEvent *event) {
    if (shape_type) {
        int delta_x = event->pos().x() - lastPoint.x();
        int delta_y = event->pos().y() - lastPoint.y();
        lastPoint = event->pos();
```

```
        offset_ += QPoint(delta_x, delta_y);
        switch (shape_type) {
        case 1:
            rectangles_list[current_rectangle_index].path.translate(offset_);
            break;
        case 2:
            circles_list[current_circle_index].path.translate(offset_);
            break;
        default:
            break;
        }
        offset_ = QPoint(0, 0);
        update();
    }
}

void Widget::mouseReleaseEvent(QMouseEvent *event) { if (event->button() ==
Qt::LeftButton && shape_type) shape_type = 0; }

void Widget::changeAngle() {
    switch (shape_type) {
    case 1:
        rectangles_list[current_rectangle_index].angle += 15;
        break;
    case 2:
        circles_list[current_circle_index].angle += 15;
        break;
    default:
        break;
    }
}

void Widget::rotate(QPainter &painter, QPainterPath& path, const qreal angle) {
    painter.translate(path.boundingRect().center());
    painter.rotate(angle);
    painter.translate(-path.boundingRect().center());
}

void Widget::scale() {
    QTransform transform;
    switch (shape_type) {
    case 1:

transform.translate(rectangles_list[current_rectangle_index].path.boundingRect().cent
er().x(),

rectangles_list[current_rectangle_index].path.boundingRect().center().y());
        transform.scale(1.2, 1.2);
        transform.translate(-
rectangles_list[current_rectangle_index].path.boundingRect().center().x(),
                            -
rectangles_list[current_rectangle_index].path.boundingRect().center().y());
        rectangles_list[current_rectangle_index].path =
transform.map(rectangles_list[current_rectangle_index].path);
        rectangles_list[current_rectangle_index].path =
rectangles_list[current_rectangle_index].path.simplified();
        break;
    case 2:

transform.translate(circles_list[current_circle_index].path.boundingRect().center().x
(),
```

```
circles_list[current_circle_index].path.boundingRect().center().y());
        transform.scale(1.2, 1.2);
        transform.translate(-
circles_list[current_circle_index].path.boundingRect().center().x(),
                            -
circles_list[current_circle_index].path.boundingRect().center().y());
        circles_list[current_circle_index].path =
transform.map(circles_list[current_circle_index].path);
        circles_list[current_circle_index].path =
circles_list[current_circle_index].path.simplified();
        break;
    default:
        break;
    }
    update();
}
```

1.3 Исходный код main.cpp

```
#include "widget.h"
#include <QApplication>

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    Widget w;
    w.show();
    return a.exec();
}
```