Правительство Российской Федерации

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

Кафедра компьютерной безопасности

Отчёт к лабораторной работе №1

по дисциплине

«Языки программирования»

Работу выполнил

студент группы СКБ223 _____ А. Я. Сорочайкин
подпись, дата

Работу проверил _____ С. А. Булгаков
подпись, дата

Москва 2023

Содержание

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 курсором мыши. При нажатии на фигуру правой кнопкой мыши выводить контекстное меню позволяющее повернуть или изменить размер фигуры.
Для реализации фигуры использовать класс QWidget и возможности класса QPainter.

1. Алгоритм решения задачи

Для решения задачи, был разработан основной класс Widget, а также классы геометрических фигур, которые можно вращать, масштабировать и перемещать. В основной функции создается объект класса Widget, на котором можно добавить геометрические фигуры.

2. Решение задачи

Класс Widget разработан на основе класса QWidget. Все классы геометрических фигур были разработаны по одному принципу. Ниже, приведен разбор структуры для класса circleFigure.

UML диаграмма зависимостей классов представлена на рисунке 1.

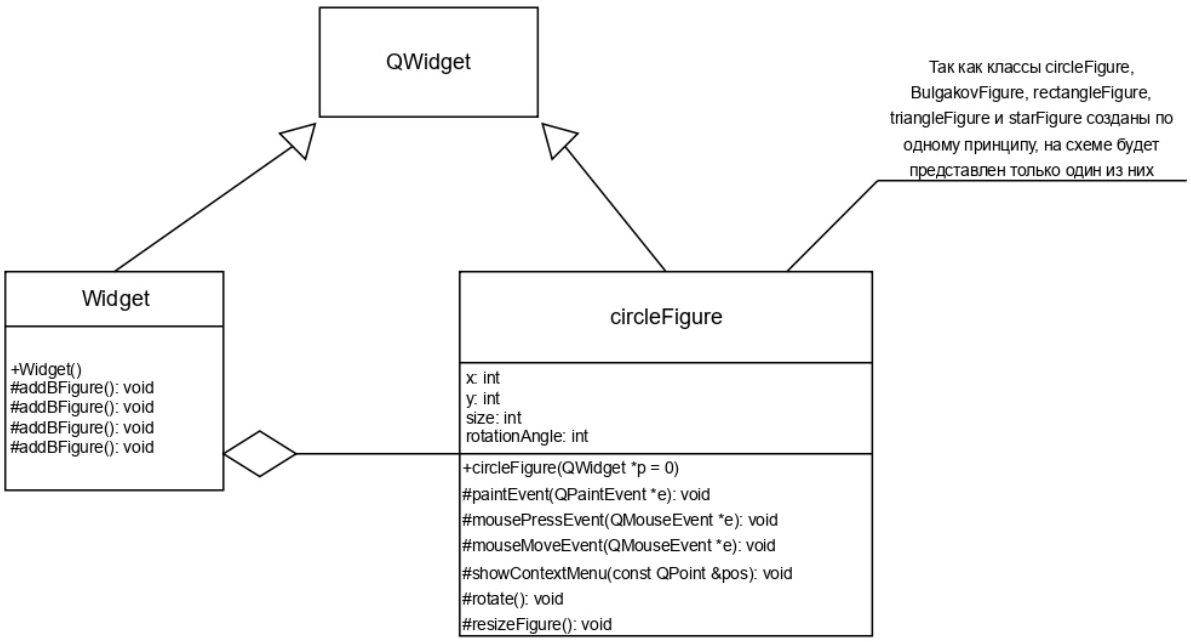

Рис. 1

2.1 Класс circleFigure

Класс состоит из, полей int x, int y, отвечающих за координаты середины виджета, полей int size и int rotationAngle – отвечающих за размер и наклон виджета соответственно, конструктора, защищенных методов void paintEvent(QPainEvent *e), mousePressEvent(QMouseEvent *e), mouseMoveEvent(QMouseEvent *e), а также приватных слотов void showContextMenu(const QPoint &pos), void rotate(), void resizeFigure().

2.1.1 Конструктор

Инициализирует поля класса, устанавливает политику контекстного меню, позволяющую пользователю определить собственное контекстное меню для данного виджета. Также, соединяем сигнал customContextMenuRequested и слотом showContextMenu

2.1.2 Метод paintEvent

Отвечает за отрисовку данной геометрической фигуры

2.1.3 Метод mousePressEvent

При нажатии на виджет, устанавливает координаты нажатия.

2.1.4 Метод mouseMoveEvent

При перемещении меняет координаты фигуры на новые.

2.1.5 Слот showContextMenu

Создаем контекстное меню фигуры, с возможными действиями: rotate и resize, также, соединяет поля контекстного меню с соответствующими функциями.

2.1.6 Слот rotate

Отвечает за поворот фигуры, на указанную пользователем в диалоговом окне, величину.

2.1.7 Слот resizeFigure

Отвечает за изменение размеров фигуры, до указанных пользователем в диалоговом окне, размеров.

2.2. Класс Widget

Класс состоит из конструктора, а также приватных слотов addBfigure, addCircle, addRectangle, addTriangle, addStar, разработанных по одному принципу. Далее, принцип их разработки будет рассмотрен на примере addCircle.

2.2.1 Конструктор

Устанавливает политику контекстного меню, позволяющую пользователю определить собственное контекстное меню для данного виджета. Также, для каждой геометрической фигуры создает действие ее создания и добавляет его в контекстное меню, а также соединяет между сигналом выбора пользователем фигуры в контекстном меню и соответствующим слотом создания геометрической фигуры.

2.2.2 Слот addCircle

Создает и отображает экземпляр соответствующего класса геометрической фигуры Circle.

3. Тестирование программы

Общий вид окна программы с отображенным контекстным меню выбора геометрической фигуры показан на рисунке 2.

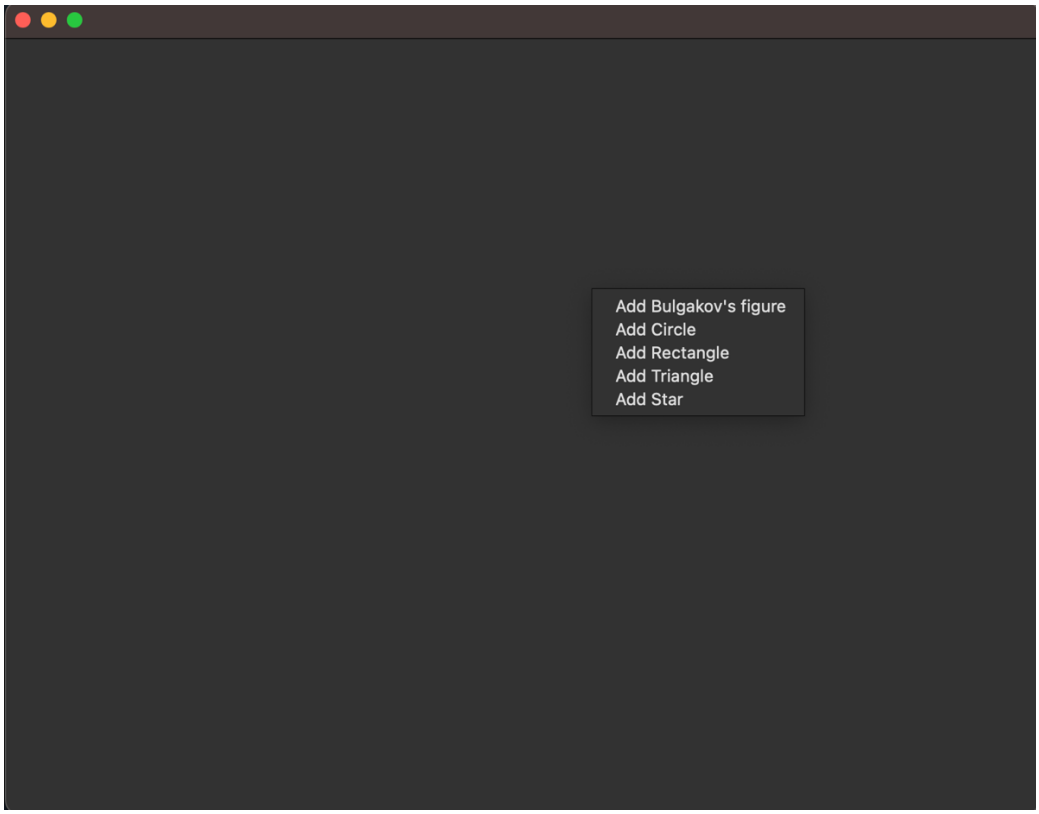

Рис. 2

Примеры всех отображенных фигур показаны на рисунке 3.

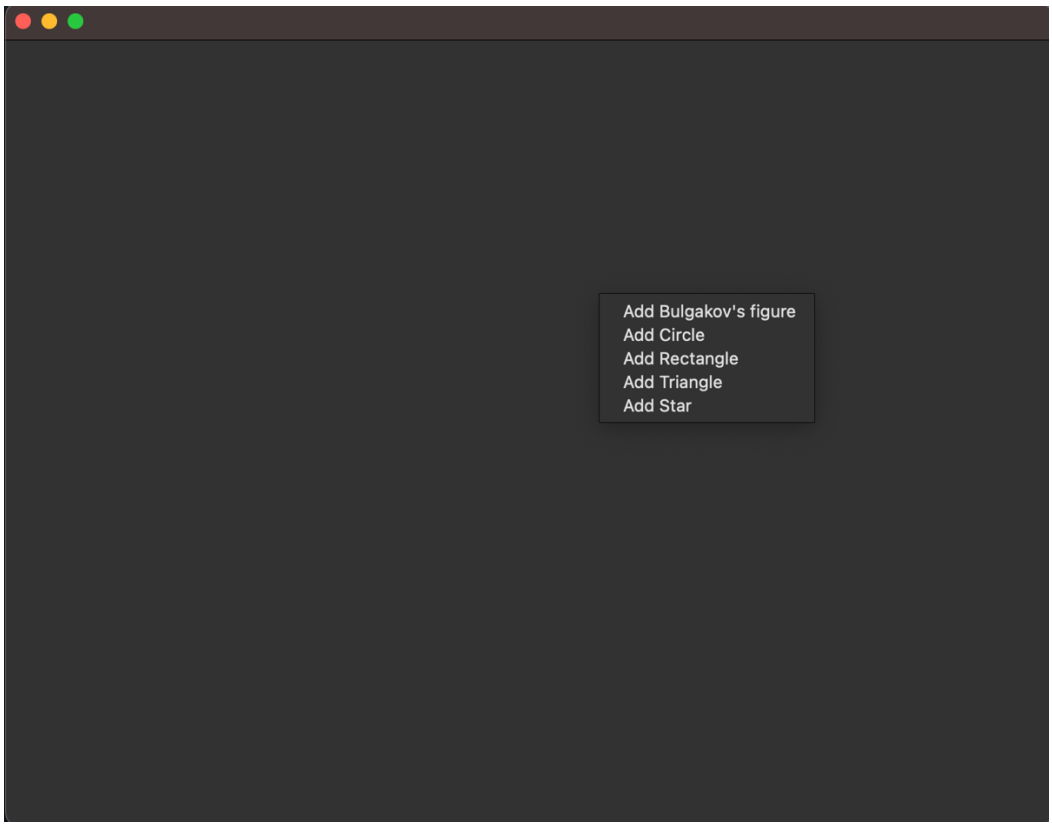

Рис. 3

На рисунке 4 представлено контекстное меню взаимодействия с фигурами, а также результаты поворота и изменения размера фигур.

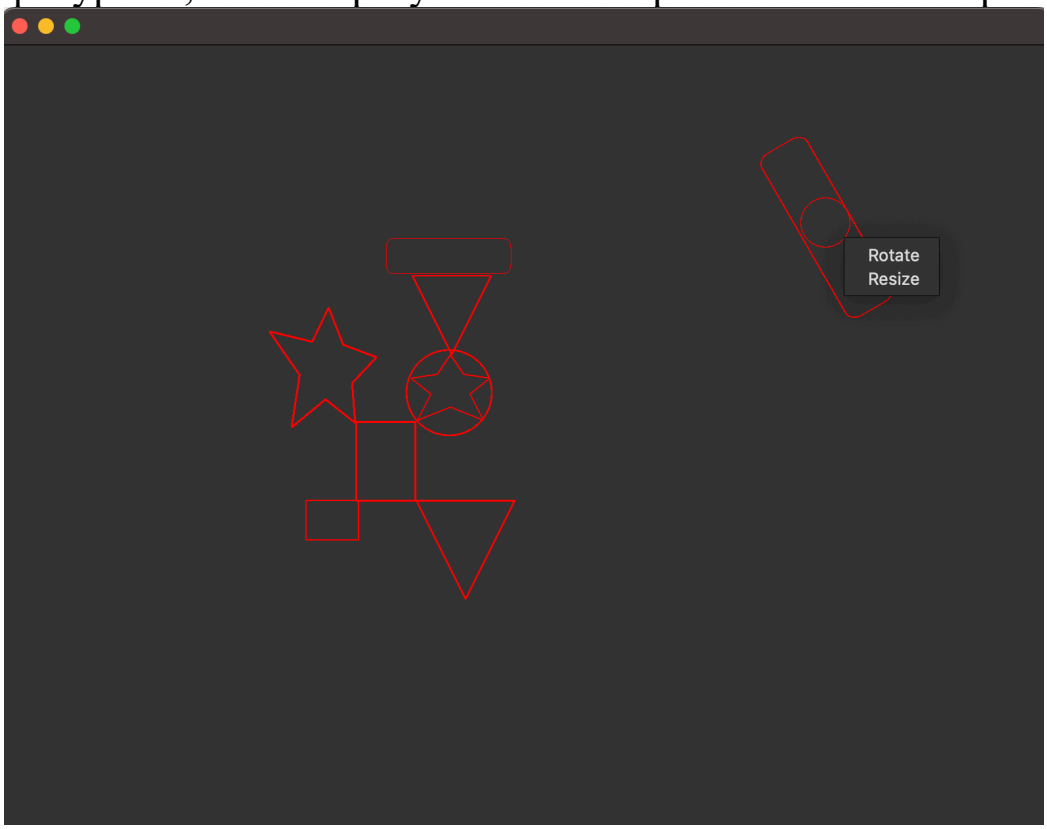

Рис. 4

Приложение А

А1. Исходный код main.cpp

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

int main(int argc, char **argv){

  QApplication app(argc,argv);

  Widget w;
  w.setMinimumSize(800,600);
  w.show();

  return app.exec();
}
```

А2. Исходный код Widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QMenu>
#include <QAction>
#include <QInputDialog>

class Widget: public QWidget
{
    Q_OBJECT
public:
    Widget(QWidget *parent = nullptr);
private slots:
    void addBFigure();
    void addCircle();
    void addRectangle();
    void addTriangle();
    void addStar();
};

#endif
```

А3. Исходный код Widget.cpp

```
#include "Widget.h"
#include "figures.h"

#include <QMouseEvent>
#include <QColor>
#include <QPainter>
#include <QAction>
#include <QInputDialog>

Widget::Widget(QWidget *parent)
        : QWidget(parent)
{
    setMinimumSize(800, 600);

    QMenu *contextMenu = new QMenu(this);
    //Bulgakov's Figure
    QAction *addBFigureAction = new QAction("Add Bulgakov's figure", this);
    connect(addBFigureAction, &QAction::triggered, this, &Widget::addBFigure);
    contextMenu->addAction(addBFigureAction);
    //Circle
    QAction *addCircleAction = new QAction("Add Circle", this);
    connect(addCircleAction, &QAction::triggered, this, &Widget::addCircle);
    contextMenu->addAction(addCircleAction);
    //Rectangle
    QAction *addRectangleAction = new QAction("Add Rectangle", this);
    connect(addRectangleAction, &QAction::triggered, this,
```

```
&Widget::addRectangle);
    contextMenu->addAction(addRectangleAction);
    //Triangle
    QAction *addTriangleAction = new QAction("Add Triangle", this);
    connect(addTriangleAction, &QAction::triggered, this,
&Widget::addTriangle);
    contextMenu->addAction(addTriangleAction);
    //Star
    QAction *addStarAction = new QAction("Add Star", this);
    connect(addStarAction, &QAction::triggered, this, &Widget::addStar);
    contextMenu->addAction(addStarAction);

    setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, &QWidget::customContextMenuRequested, [this,
contextMenu](const QPoint &pos) {
        contextMenu->popup(mapToGlobal(pos));
    });
}

void Widget::addBFigure()
{
    BulgakovFigure *figure = new BulgakovFigure(this);
    figure->show();
}

void Widget::addCircle()
{
    circleFigure *figure = new circleFigure(this);
    figure->show();
}

void Widget::addRectangle()
{
    rectangleFigure *figure = new rectangleFigure(this);
    figure->show();
}

void Widget::addTriangle()
{
    triangleFigure *figure = new triangleFigure(this);
    figure->show();
}

void Widget::addStar()
{
    starFigure *figure = new starFigure(this);
    figure->show();
}
```

A4. Исходный код figures.h

```
#ifndef FIGURES_H
#define FIGURES_H

#include <QWidget>
#include <QMouseEvent>

class BulgakovFigure: public QWidget
{
    Q_OBJECT
    int x,y;
    int size;
    int rotationAngle;

protected:
    void paintEvent(QPaintEvent *e);
    void mousePressEvent(QMouseEvent *e);
    void mouseMoveEvent(QMouseEvent *e);
```

```
public:
    BulgakovFigure(QWidget *p = 0);

private slots:
    void showContextMenu(const QPoint &pos);
    void rotate();
    void resizeFigure();
};

class circleFigure: public QWidget
{
    Q_OBJECT
    int x,y;
    int size;
    int rotationAngle;


protected:
    void paintEvent(QPaintEvent *e);
    void mousePressEvent(QMouseEvent *e);
    void mouseMoveEvent(QMouseEvent *e);

public:
    circleFigure(QWidget *p = 0);

private slots:
    void showContextMenu(const QPoint &pos);
    void rotate();
    void resizeFigure();
};

class rectangleFigure: public QWidget
{
    Q_OBJECT
    int x,y;
    int size;
    int rotationAngle;

protected:
    void paintEvent(QPaintEvent *e);
    void mousePressEvent(QMouseEvent *e);
    void mouseMoveEvent(QMouseEvent *e);

public:
    rectangleFigure(QWidget *p = 0);

private slots:
            void showContextMenu(const QPoint &pos);
    void rotate();
    void resizeFigure();
};

class triangleFigure: public QWidget
{
    Q_OBJECT
    int x,y;
    int size;
    int rotationAngle;

protected:
    void paintEvent(QPaintEvent *e);
    void mousePressEvent(QMouseEvent *e);
    void mouseMoveEvent(QMouseEvent *e);

public:
    triangleFigure(QWidget *p = 0);

private slots:
```

```
            void showContextMenu(const QPoint &pos);
    void rotate();
    void resizeFigure();
};

class starFigure: public QWidget
{
    Q_OBJECT
    int x,y;
    int size;
    int rotationAngle;

protected:
    void paintEvent(QPaintEvent *e);
    void mousePressEvent(QMouseEvent *e);
    void mouseMoveEvent(QMouseEvent *e);

public:
    starFigure(QWidget *p = 0);

private slots:
            void showContextMenu(const QPoint &pos);
    void rotate();
    void resizeFigure();
};
#endif
```

A5. Исходный код figures.cpp

```
#include "figures.h"

#include <QPainter>
#include <QMenu>
#include <QInputDialog>
#include <QTransform>

BulgakovFigure::BulgakovFigure(QWidget *p) : QWidget(p)
{
    setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, &QWidget::customContextMenuRequested, this,
&BulgakovFigure::showContextMenu);
    size = 100;
    rotationAngle = 0;
}

void BulgakovFigure::paintEvent(QPaintEvent *e)
{
    QTransform transform;
    transform.translate(width() / 2, height() / 2);
    transform.rotate(rotationAngle);
    transform.scale(size / 200.0, size / 150.0);

    setMinimumSize(350,350);
    QPainter p(this);

    p.setPen(QColor(255,0,0));
    p.setTransform(transform);

    p.drawLine(20, 10,190,10); /* o--o */
    p.drawLine(200,20,200,40); /*    | */
    p.drawLine(190,50, 20,50); /* o--o */
    p.drawLine(10, 40, 10,20); /* |    */
    p.drawArc(10,10,20,20,90*16,90*16);
    p.drawArc(10,30,20,20,180*16,90*16);
    p.drawArc(180,30,20,20,270*16,90*16);
    p.drawArc(180,10,20,20,  0*16,90*16);
}

void BulgakovFigure::mousePressEvent(QMouseEvent *e)
```

```
{
    x = e->pos().x();
    y = e->pos().y();
}

void BulgakovFigure::mouseMoveEvent(QMouseEvent *e)
{
    int delta_x = e->pos().x() - x;
    int delta_y = e->pos().y() - y;
    move(pos().x()+delta_x, pos().y()+delta_y);
}

void BulgakovFigure::showContextMenu(const QPoint &pos) {
    QMenu contextMenu(this);
    QAction rotateAction("Rotate", this);
    QAction resizeAction("Resize", this);

    connect(&rotateAction, &QAction::triggered, this, &BulgakovFigure::rotate);
    connect(&resizeAction, &QAction::triggered, this,
&BulgakovFigure::resizeFigure);

    contextMenu.addAction(&rotateAction);
    contextMenu.addAction(&resizeAction);
    contextMenu.exec(mapToGlobal(pos));
}

void BulgakovFigure::rotate()
{
    bool ok;
    int angle = QInputDialog::getInt(this, "Rotate", "Enter rotation angle
(degrees) (from -360 to 360):", 0, -360, 360, 1, &ok);
    if (ok)
    {
        rotationAngle = angle;
        update();
    }
}

void BulgakovFigure::resizeFigure()
{
    bool ok;
    int newSize = QInputDialog::getInt(this, "Resize", "Enter new size (from 10
to 150):", size, 10, 150, 1, &ok);
    if (ok)
    {
        size = newSize;
        update();
    }
}

circleFigure::circleFigure(QWidget *p) : QWidget(p)
{
    size = 100;
    rotationAngle = 0;
    setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, &QWidget::customContextMenuRequested, this,
&circleFigure::showContextMenu);
}

void circleFigure::paintEvent(QPaintEvent *e)
{

    QTransform transform;
    transform.translate(width() / 2, height() / 2);
    transform.rotate(rotationAngle);
    transform.scale(size / 100.0, size / 100.0);

    setMinimumSize(350,350);
```

```
    QPainter p(this);

    p.setPen(QColor(255,0,0));
    p.setTransform(transform);

    p.setPen(QColor(255,0,0));
    int circleX = 30;
    int circleY = 30;
    int circleRadius = 25;
    p.drawEllipse(circleX - circleRadius, circleY - circleRadius, 2 *
circleRadius, 2 * circleRadius);
}

void circleFigure::mousePressEvent(QMouseEvent *e)
{
    x = e->pos().x();
    y = e->pos().y();
}

void circleFigure::mouseMoveEvent(QMouseEvent *e)
{
    int delta_x = e->pos().x() - x;
    int delta_y = e->pos().y() - y;
    move(pos().x()+delta_x, pos().y()+delta_y);
}

void circleFigure::showContextMenu(const QPoint &pos) {
    QMenu contextMenu(this);
    QAction rotateAction("Rotate", this);
    QAction resizeAction("Resize", this);

    connect(&rotateAction, &QAction::triggered, this, &circleFigure::rotate);
    connect(&resizeAction, &QAction::triggered, this,
&circleFigure::resizeFigure);

    contextMenu.addAction(&rotateAction);
    contextMenu.addAction(&resizeAction);
    contextMenu.exec(mapToGlobal(pos));
}

void circleFigure::rotate()
{
    bool ok;
    int angle = QInputDialog::getInt(this, "Rotate", "Enter rotation angle
(degrees) (from -360 to 360):", 0, -360, 360, 1, &ok);
    if (ok)
    {
        rotationAngle = angle;
        update();
    }
}

void circleFigure::resizeFigure()
{
    bool ok;
    int newSize = QInputDialog::getInt(this, "Resize", "Enter new size (from 10
to 150):", size, 10, 150, 1, &ok);
    if (ok)
    {
        size = newSize;
        update();
    }
}

rectangleFigure::rectangleFigure(QWidget *p) : QWidget(p)
{
    size = 100;
    rotationAngle = 0;
```

```
    setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, &QWidget::customContextMenuRequested, this,
&rectangleFigure::showContextMenu);
}

void rectangleFigure::paintEvent(QPaintEvent *e)
{

    QTransform transform;
    transform.translate(width() / 2, height() / 2);
    transform.rotate(rotationAngle);
    transform.scale(size / 100.0, size / 100.0);

    setMinimumSize(350,350);
    QPainter p(this);

    p.setPen(QColor(255,0,0));
    p.setTransform(transform);

    p.setPen(QColor(255,0,0));
    int rectX = 50;
    int rectY = 50;
    int rectWidth = 40;
    int rectHeight = 30;
    p.drawRect(rectX, rectY, rectWidth, rectHeight);
}

void rectangleFigure::mousePressEvent(QMouseEvent *e)
{
    x = e->pos().x();
    y = e->pos().y();
}

void rectangleFigure::mouseMoveEvent(QMouseEvent *e)
{
    int delta_x = e->pos().x() - x;
    int delta_y = e->pos().y() - y;
    move(pos().x()+delta_x, pos().y()+delta_y);
}

void rectangleFigure::showContextMenu(const QPoint &pos) {
    QMenu contextMenu(this);
    QAction rotateAction("Rotate", this);
    QAction resizeAction("Resize", this);

    connect(&rotateAction, &QAction::triggered, this,
&rectangleFigure::rotate);
    connect(&resizeAction, &QAction::triggered, this,
&rectangleFigure::resizeFigure);

    contextMenu.addAction(&rotateAction);
    contextMenu.addAction(&resizeAction);
    contextMenu.exec(mapToGlobal(pos));
}

void rectangleFigure::rotate()
{
    bool ok;
    int angle = QInputDialog::getInt(this, "Rotate", "Enter rotation angle
(degrees) (from -360 to 360):", 0, -360, 360, 1, &ok);
    if (ok)
    {
        rotationAngle = angle;
        update();
    }
}

void rectangleFigure::resizeFigure()
```

```
{
    bool ok;
    int newSize = QInputDialog::getInt(this, "Resize", "Enter new size (from 10
to 150):", size, 10, 150, 1, &ok);
    if (ok)
    {
        size = newSize;
        update();
    }
}


triangleFigure::triangleFigure(QWidget *p) : QWidget(p)
{
    size = 100;
    rotationAngle = 0;
    setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, &QWidget::customContextMenuRequested, this,
&triangleFigure::showContextMenu);
}

void triangleFigure::paintEvent(QPaintEvent *e)
{
    QTransform transform;
    transform.translate(width() / 2, height() / 2);
    transform.rotate(rotationAngle);
    transform.scale(size / 100.0, size / 100.0);

    setMinimumSize(350,350);
    QPainter p(this);

    p.setPen(QColor(255,0,0));
    p.setTransform(transform);

    p.setPen(QColor(255,0,0));
    int x = 10;
    int y = 10;
    int se = 50;
    QPolygon triangle;
    triangle << QPoint(x, y) << QPoint(x + se, y) << QPoint(x + se / 2, y +
se);
    p.drawPolygon(triangle);
}

void triangleFigure::mousePressEvent(QMouseEvent *e)
{
    x = e->pos().x();
    y = e->pos().y();
}

void triangleFigure::mouseMoveEvent(QMouseEvent *e)
{
    int delta_x = e->pos().x() - x;
    int delta_y = e->pos().y() - y;
    move(pos().x()+delta_x, pos().y()+delta_y);
}

void triangleFigure::showContextMenu(const QPoint &pos) {
    QMenu contextMenu(this);
    QAction rotateAction("Rotate", this);
    QAction resizeAction("Resize", this);

    connect(&rotateAction, &QAction::triggered, this, &triangleFigure::rotate);
    connect(&resizeAction, &QAction::triggered, this,
&triangleFigure::resizeFigure);

    contextMenu.addAction(&rotateAction);
    contextMenu.addAction(&resizeAction);
```

```
    contextMenu.exec(mapToGlobal(pos));
}

void triangleFigure::rotate()
{
    bool ok;
    int angle = QInputDialog::getInt(this, "Rotate", "Enter rotation angle
(degrees) (from -360 to 360):", 0, -360, 360, 1, &ok);
    if (ok)
    {
        rotationAngle = angle;
        update();
    }
}

void triangleFigure::resizeFigure()
{
    bool ok;
    int newSize = QInputDialog::getInt(this, "Resize", "Enter new size (from 10
to 150):", size, 10, 150, 1, &ok);
    if (ok)
    {
        size = newSize;
        update();
    }
}

starFigure::starFigure(QWidget *p) : QWidget(p)
{
    size = 100;
    rotationAngle = 0;
    setContextMenuPolicy(Qt::CustomContextMenu);
    connect(this, &QWidget::customContextMenuRequested, this,
&starFigure::showContextMenu);
}

void starFigure::paintEvent(QPaintEvent *e)
{

    QTransform transform;
    transform.translate(width() / 2, height() / 2);
    transform.rotate(rotationAngle);
    transform.scale(size / 100.0, size / 100.0);

    setMinimumSize(350,350);
    QPainter p(this);

    p.setPen(QColor(255,0,0));
    p.setTransform(transform);

    p.setPen(QColor(255,0,0));

    int x = 10;
    int y = 10;

    QPolygon star;
    star << QPoint(x + 50, y) << QPoint(x + 60, y + 15) << QPoint(x + 80, y +
18) << QPoint(x + 65, y + 30)
         << QPoint(x + 75, y + 50) << QPoint(x + 50, y + 40) << QPoint(x + 25,
y + 50) << QPoint(x + 35, y + 30)
         << QPoint(x + 20, y + 18) << QPoint(x + 40, y + 15);
    p.drawPolygon(star);
}

void starFigure::mousePressEvent(QMouseEvent *e)
{
    x = e->pos().x();
    y = e->pos().y();
```

```
}

void starFigure::mouseMoveEvent(QMouseEvent *e)
{
    int delta_x = e->pos().x() - x;
    int delta_y = e->pos().y() - y;
    move(pos().x()+delta_x, pos().y()+delta_y);
}

void starFigure::showContextMenu(const QPoint &pos) {
    QMenu contextMenu(this);
    QAction rotateAction("Rotate", this);
    QAction resizeAction("Resize", this);

    connect(&rotateAction, &QAction::triggered, this, &starFigure::rotate);
    connect(&resizeAction, &QAction::triggered, this,
&starFigure::resizeFigure);

    contextMenu.addAction(&rotateAction);
    contextMenu.addAction(&resizeAction);
    contextMenu.exec(mapToGlobal(pos));
}

void starFigure::rotate()
{
    bool ok;
    int angle = QInputDialog::getInt(this, "Rotate", "Enter rotation angle
(degrees) (from -360 to 360):", 0, -360, 360, 1, &ok);
    if (ok)
    {
        rotationAngle = angle;
        update();
    }
}

void starFigure::resizeFigure()
{
    bool ok;
    int newSize = QInputDialog::getInt(this, "Resize", "Enter new size (from 10
to 150):", size, 10, 150, 1, &ok);
    if (ok)
    {
        size = newSize;
        update();
    }
}
```