Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Звіт

з лабораторної роботи № 4 з дисципліни

«Вдосконалення структури коду графічного редактора об’єктів на С++»

Варіант 29

Виконав студент ІМ-11, Пономаренко Маргарита Альбертівна
(шифр, прізвище, ім'я, по батькові)

Перевірив Порєв В. М.
(прізвище, ім'я, по батькові)

Київ 2022

Мета роботи: отримати вміння та навички проектування класів, виконавши модернізацію коду графічного редактора в об’єктно-орієнтованому стилі для забезпечення зручного додавання нових типів об'єктів.

Завдання:

1. Створити у середовищі Qt Creator проект з ім’ям Lab4.

2. Написати вихідний текст програми згідно варіанту завдання.

3. Скомпілювати вихідний текст і отримати виконуваний файл програми.

4. Перевірити роботу програми. Налагодити програму.

5. Проаналізувати та прокоментувати результати та вихідний текст програми.

6. Оформити звіт.

Варіант 29:

Студенти з непарним номером (1, 3, 5, . . .) програмують глобальний статичний об'єкт класу MyEditor.

Текст головного файлу (main.cpp)

```
#include "mainwindow.h"

#include <QApplication>

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

Текст заголовку головного вікна (mainwindow.h)

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
```

```
#include "myeditorview.h"
#include "toolbar.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_actionNew_triggered();

  void on_drawPoint_triggered();

  void on_drawLine_triggered();

  void on_drawRect_triggered();

  void on_drawElipse_triggered();

  void on_drawLineWithElipse_triggered();

  void on_drawCube_triggered();

private:
  Ui::MainWindow *ui;
  MyEditorView *myEditorView;
  Toolbar *toolBar;
  void setUpToolBar();

};
#endif // MAINWINDOW_H
```

Текст реалізації головного вікна (mainwindow.cpp)

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "scenewindow.h"
#include "toolbar.h"

#include <QToolBar>

MainWindow::MainWindow(QWidget *parent)
  : QMainWindow(parent)
  , ui(new Ui::MainWindow)
```

```
{
  ui->setupUi(this);
  QApplication::instance()->setAttribute(Qt::AA_DontShowIconsInMenus, true);

  toolBar = new Toolbar(this);
  addToolBar(toolBar);
  setUpToolBar();
}

MainWindow::~MainWindow()
{
  delete ui;
}

void MainWindow::on_actionNew_triggered()
{
  myEditorView = createEditor(this);
}

void MainWindow::on_drawPoint_triggered()
{
  setWindowTitle("Режим малювання: крапка");
  ui->drawLine->setChecked(false);
  ui->drawRect->setChecked(false);
  ui->drawElipse->setChecked(false);
  ui->drawLineWithElipse->setChecked(false);
  ui->drawCube->setChecked(false);

  myEditorView->drawShape(DrawType::POINT);
}

void MainWindow::on_drawLine_triggered()
{
  setWindowTitle("Режим малювання: лінія");
  ui->drawPoint->setChecked(false);
  ui->drawRect->setChecked(false);
  ui->drawElipse->setChecked(false);
  ui->drawLineWithElipse->setChecked(false);
  ui->drawCube->setChecked(false);

  myEditorView->drawShape(DrawType::LINE);
}

void MainWindow::on_drawRect_triggered()
{
  setWindowTitle("Режим малювання: прямокутник");
  ui->drawPoint->setChecked(false);
  ui->drawLine->setChecked(false);
  ui->drawElipse->setChecked(false);
  ui->drawLineWithElipse->setChecked(false);
  ui->drawCube->setChecked(false);
```

```
  myEditorView->drawShape(DrawType::RECT);
}

void MainWindow::on_drawElipse_triggered()
{
  setWindowTitle("Режим малювання: еліпс");
  ui->drawPoint->setChecked(false);
  ui->drawLine->setChecked(false);
  ui->drawRect->setChecked(false);
  ui->drawLineWithElipse->setChecked(false);
  ui->drawCube->setChecked(false);

  myEditorView->drawShape(DrawType::ELIPSE);
}

void MainWindow::on_drawLineWithElipse_triggered()
{
  setWindowTitle("Режим малювання: лінія з еліпсами");
  ui->drawPoint->setChecked(false);
  ui->drawLine->setChecked(false);
  ui->drawRect->setChecked(false);
  ui->drawElipse->setChecked(false);
  ui->drawCube->setChecked(false);

  myEditorView->drawShape(DrawType::LINEWITHELIPSE);
}

void MainWindow::on_drawCube_triggered()
{
  setWindowTitle("Режим малювання: куба");
  ui->drawPoint->setChecked(false);
  ui->drawLine->setChecked(false);
  ui->drawRect->setChecked(false);
  ui->drawElipse->setChecked(false);
  ui->drawLineWithElipse->setChecked(false);

  myEditorView->drawShape(DrawType::CUBE);
}

void MainWindow::setUpToolBar()
{
  toolBar->addAction(ui->actionNew);
  toolBar->addSeparator();
  toolBar->addAction(ui->drawPoint);
  toolBar->addAction(ui->drawLine);
  toolBar->addAction(ui->drawRect);
  toolBar->addAction(ui->drawElipse);
  toolBar->addAction(ui->drawLineWithElipse);
  toolBar->addAction(ui->drawCube);
}
```

Текст заголовку файлу класу Shape (shape.h)

```
#ifndef SHAPE_H
#define SHAPE_H
#include <QGraphicsItem>

enum class DrawType{
  POINT = 0,
  LINE,
  RECT,
  ELIPSE,
  LINEWITHELIPSE,
  CUBE,
};

class Shape:public virtual QGraphicsItem
{
public:
  Shape();
  virtual ~Shape();
  void Set(int x1, int y1, int x2, int y2);
  virtual void Show(QColor color) = 0;

  int getXs1() const;
  int getYs1() const;
  int getXs2() const;
  int getYs2() const;

  static Shape * createShape(DrawType type);
  virtual void startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene * scene) = 0;
  virtual void drawRubberFootprint(QGraphicsSceneMouseEvent *event) {}
  virtual void endDrawing() {}

protected:
  int xs1=0;
  int ys1=0;
  int xs2=0;
  int ys2=0;

  QGraphicsScene * scene;
  QColor color = Qt::black;
};

#endif // SHAPE_H
```

Текст реалізації класу Shape (shape.cpp)

```
#include "shape.h"
#include "pointshape.h"
```

```
#include "lineshape.h"
#include "rectshape.h"
#include "elipseshape.h"
#include "linewithelipseshape.h"
#include "cubeshape.h"

Shape::Shape()
{

}

Shape::~Shape()
{
}

void Shape::Set(int x1, int y1, int x2, int y2)
{
  xs1 = x1;
  ys1 = y1;
  xs2 = x2;
  ys2 = y2;
}

int Shape::getXs1() const
{
  return xs1;
}

int Shape::getYs1() const
{
  return ys1;
}

int Shape::getXs2() const
{
  return xs2;
}

int Shape::getYs2() const
{
  return ys2;
}

Shape *Shape::createShape(DrawType type)
{
  switch (type){
  case DrawType::POINT:
     return new PointShape;
     break;
  case DrawType::LINE:
     return new LineShape;
     break;
```

```
  case DrawType::RECT:
    return new RectShape;
    break;
  case DrawType::ELIPSE:
    return new ElipseShape;
    break;
  case DrawType::LINEWITHELIPSE:
    return (ElipseShape *)new LineWithElipseShape;
    break;
  case DrawType::CUBE:
    return (LineShape *)new CubeShape;
    break;
  }
}
```

Текст заголовку файлу класу PointShape (pointshape.h)

```
#ifndef POINTSHAPE_H
#define POINTSHAPE_H
#include "shape.h"

class PointShape: public Shape
{
public:
  PointShape();
  void Show(QColor color) override;
  void startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene * scene) override;
  void drawRubberFootprint(QGraphicsSceneMouseEvent *event) override;
  void endDrawing() override;

  QRectF boundingRect() const override;
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
override;  //малювання фігури
  QPainterPath shape() const override;
};

#endif // POINTSHAPE_H
```

Текст реалізації класу PointShape (pointshape.cpp)

```
#include "pointshape.h"

#include <QPainter>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsScene>

PointShape::PointShape()
{
```

```
}

void PointShape::Show(QColor color)
{
  this->color = color;
  this->setPos(xs1, ys1);
}

void PointShape::startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene * scene)
{
  this->scene = scene;
  int x = event->scenePos().x();  //ініціалізація позиції x та y курсора
  int y = event->scenePos().y();
  Set(x, y, x, y);
  scene->addItem(this);  //додавання об'єкту крапка
  Show(Qt::blue);
}

void PointShape::drawRubberFootprint(QGraphicsSceneMouseEvent *event)
{
}

void PointShape::endDrawing()
{
}

QRectF PointShape::boundingRect() const  //виділенян на сцені прямокутної області для
відображення фігури
{
  return QRectF(0,0,2,2);
}

void PointShape::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)  //метод малювання крапки
{
  qDebug()<<"Paint Point"<< xs1<<", "<<ys1;
  Q_UNUSED(option);
  Q_UNUSED(widget);
  painter->setBrush(Qt::black);
  painter->setPen(Qt::NoPen);
  painter->drawEllipse(0,0,2,2);
}

QPainterPath PointShape::shape() const
{
  QPainterPath path;
  path.addRect(0,0,2,2);
  return path;
}
```

Текст заголовку файлу класу LineShape (lineshape.h)

```
#ifndef LINESHAPE_H
#define LINESHAPE_H
#include "shape.h"

class LineShape: public Shape
{
public:
  LineShape();
  virtual ~LineShape(){}
  void Show(QColor color) override;
  virtual void startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene * scene)
override;
  virtual void drawRubberFootprint(QGraphicsSceneMouseEvent *event) override;
  virtual void endDrawing() override;

  QRectF boundingRect() const override;
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
override;
  QPainterPath shape() const override;

protected:
  Qt::PenStyle style = Qt::SolidLine;
  int x1,x2,y1,y2;
  int xSize() const;
  int ySize() const;
};

#endif // LINESHAPE_H
```

Текст реалізації класу LineShape (lineshape.cpp)

```
#include "lineshape.h"
#include <QPainter>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsScene>

LineShape::LineShape()
{

}

void LineShape::Show(QColor color)
{
  this->color = color;
  this->setPos(std::min(xs1,xs2), std::min(ys1,ys2));
}
```

```
void LineShape::startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene *scene)
{
  int x = event->scenePos().x();  //ініціалізація позиції x та y курсора
  int y = event->scenePos().y();
  Set(x, y, x, y);
  scene->addItem(this);  //додавання об'єкту лінія
  Show(Qt::blue);
}

void LineShape::drawRubberFootprint(QGraphicsSceneMouseEvent *event)
{
  int x1 = getXs1();
  int y1 = getYs1();
  int x2 = event->scenePos().x();
  int y2 = event->scenePos().y();
  Set(x1, y1, x2, y2);
  style = Qt::DashLine;
  Show(Qt::blue);
}

void LineShape::endDrawing()
{
  style = Qt::SolidLine;
  Show(Qt::black);
}

QRectF LineShape::boundingRect() const
{
//    qreal adjust = 0.5; before
  qreal adjust = 10;
  QPointF point1(0 - adjust, 0 - adjust);
  QPointF point2(xSize() + adjust, ySize() + adjust);
  //qDebug()<<"Bounding rect"<<xSize()<<"; "<<ySize();
  qDebug()<<"Bounds"<<point1<<"; "<<point2;
  return QRectF(point1, point2);
}

void LineShape::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
  Q_UNUSED(option);
  Q_UNUSED(widget);
  qDebug()<<"Paint Line"<< xs1<<"; "<<ys1<<"; "<<xs2<<"; "<<ys2;


  if ((xs2 >xs1 ) && (ys2>ys1)){
     x1 = 0;
     y1 = 0;
     x2 = xSize();
     y2 = ySize();
  }
  if ((xs2 <= xs1 ) && (ys2>ys1)){
```

```
        x1 = xSize();
        y1 = 0;
        x2 = 0;
        y2 = ySize();
    }
    if ((xs2 >xs1 ) && (ys2<=ys1)){
        x1 = 0;
        y1 = ySize();
        x2 = xSize();
        y2 = 0;
    }
    if ((xs2 <=xs1 ) && (ys2<=ys1)){
        x1 = xSize();
        y1 = ySize();
        x2 = 0;
        y2 = 0;
    }

    QPointF point1(x1, y1);
    QPointF point2(x2, y2);

    painter->setPen(QPen(color, 2, style));
    //painter->setBrush(color);
    painter->drawLine(point2, point1);
}

QPainterPath LineShape::shape() const
{
    QPainterPath path;
    QPointF point1(0, 0);
//    QPointF point2(xSize(), ySize()); before
    QPointF point2(xSize() + 20, ySize() + 20);
    //path.addRect(QRectF(point1, point2)); before
    path.addRect(QRectF(point1, point2));
    return path;
}

int LineShape::xSize() const
{
    return abs(xs1 - xs2);
}

int LineShape::ySize() const
{
    return abs(ys1 - ys2);
}
```

Текст заголовку файлу класу RectShape (rectshape.h)

```
#ifndef RECTSHAPE_H
#define RECTSHAPE_H
```

```
#include "shape.h"

class RectShape: public Shape
{
public:
  RectShape();
  void Show(QColor color) override;
  void startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene * scene) override;
  void drawRubberFootprint(QGraphicsSceneMouseEvent *event) override;
  void endDrawing() override;

  QRectF boundingRect() const override;
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
override;
  QPainterPath shape() const override;

private:
  Qt::PenStyle style = Qt::SolidLine;
  int xSize() const;
  int ySize() const;
};

#endif // RECTSHAPE_H
```

Текст реалізації класу RectShape (rectshape.cpp)

```
#include "rectshape.h"
#include <QPainter>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsScene>

RectShape::RectShape()
{

}

void RectShape::Show(QColor color)
{
  this->color = color;
  this->setPos(std::min(xs1,xs2), std::min(ys1,ys2));
}

void RectShape::startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene *scene)
{
  int x = event->scenePos().x();  //ініціалізація позиції x та y курсора
  int y = event->scenePos().y();
  Set(x, y, x, y);
  scene->addItem(this);  //додавання об'єкту лінія
  Show(Qt::blue);
}
```

```
void RectShape::drawRubberFootprint(QGraphicsSceneMouseEvent *event)
{
  int x1 = getXs1();
  int y1 = getYs1();
  int x2 = event->scenePos().x();
  int y2 = event->scenePos().y();
  Set(x1, y1, x2, y2);
  style = Qt::DashLine;
  Show(Qt::blue);
}

void RectShape::endDrawing()
{
  style = Qt::SolidLine;
  Show(Qt::black);
}

QRectF RectShape::boundingRect() const
{
  qreal adjust = 0.5;
  QPointF point1(0 - adjust - xSize() / 2, 0 - adjust - ySize() / 2);
  QPointF point2(xSize() / 2 + adjust, ySize() / 2 + adjust);
  qDebug()<<"Bounding rect"<<xSize()<<"; "<<ySize();
  return QRectF(point1, point2);
}

void RectShape::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
  Q_UNUSED(option);
  Q_UNUSED(widget);
  qDebug()<<"Paint Line"<< xs1<<"; "<<ys1<<"; "<<xs2<<"; "<<ys2;
  int x1,x2,y1,y2;

  if ((xs2 >xs1 ) && (ys2>ys1)){
     qDebug()<<"1 path";
     x1 = -xSize();
     y1 = -ySize();
     x2 = xSize() + xSize() / 2;
     y2 = ySize() + ySize() / 2;
  }
  if ((xs2 <= xs1 ) && (ys2>ys1)){
     qDebug()<<"2 path";
     x1 = 0;
     y1 = -ySize();
     x2 = xSize();
     y2 = ySize() + ySize() / 2;
  }
  if ((xs2 >xs1 ) && (ys2<=ys1)){
     qDebug()<<"3 path";
     x1 = -xSize();
```

```
        y1 = 0;
        x2 = xSize() + xSize() / 2;
        y2 = ySize();
    }
    if ((xs2 <=xs1 ) && (ys2<=ys1)){
        qDebug()<<"4 path";
        x1 = 0;
        y1 = 0;
        x2 = xSize();
        y2 = ySize();
    }

    QPointF point1(x1, y1);
    QPointF point2(x2, y2);

    if(color == Qt::blue){
        painter->setPen(QPen(color, 2, style));

    }
    else{
        painter->setPen(QPen(color, 2, style));
        painter->setBrush(Qt::white);
    }
    painter->drawRect(x1, y1, x2, y2);
}

QPainterPath RectShape::shape() const
{
    QPainterPath path;
    QPointF point1(0, 0);
    QPointF point2(xSize(), ySize());
    path.addRect(QRectF(point1, point2));
    return path;
}

int RectShape::xSize() const
{
    return abs(xs1 - xs2) * 2;
}

int RectShape::ySize() const
{
    return abs(ys1 - ys2) * 2;
}
```

Текст заголовку файлу класу ElipseShape (elipseshape.h)

```
#ifndef ELIPSESHAPE_H
#define ELIPSESHAPE_H
```

```
#include "shape.h"

class ElipseShape: public Shape
{
public:
  ElipseShape();
  virtual ~ElipseShape(){}
  void Show(QColor color) override;
  virtual void  startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene * scene)
override;
  virtual void drawRubberFootprint(QGraphicsSceneMouseEvent *event) override;
  virtual void endDrawing() override;


  QRectF boundingRect() const override;
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
override;
  QPainterPath shape() const override;

protected:
  Qt::PenStyle style = Qt::SolidLine;
  int xSize() const;
  int ySize() const;
};

#endif // ELIPSESHAPE_H
```

Текст реалізації класу ElipseShape (elipseshape.cpp)

```
#include "elipseshape.h"
#include <QPainter>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsScene>

ElipseShape::ElipseShape()
{

}

void ElipseShape::Show(QColor color)
{
  this->color = color;
  this->setPos(std::min(xs1,xs2), std::min(ys1,ys2));
}

void ElipseShape::startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene *scene)
{
  int x = event->scenePos().x();  //ініціалізація позиції x та y курсора
  int y = event->scenePos().y();
  Set(x, y, x, y);
```

```
  scene->addItem(this);
  Show(Qt::blue);
}

void ElipseShape::drawRubberFootprint(QGraphicsSceneMouseEvent *event)
{
  int x1 = getXs1();
  int y1 = getYs1();
  int x2 = event->scenePos().x();
  int y2 = event->scenePos().y();
  Set(x1, y1, x2, y2);
  style = Qt::DashLine;
  Show(Qt::blue);

}

void ElipseShape::endDrawing()
{
  style = Qt::SolidLine;
  Show(Qt::black);
}

QRectF ElipseShape::boundingRect() const
{
  qreal adjust = 0.5;
  QPointF point1(0 - adjust, 0 - adjust);
  QPointF point2(xSize() + adjust, ySize() + adjust);
  qDebug()<<"Bounding rect"<<xSize()<<"; "<<ySize();
  return QRectF(point1, point2);
}

void ElipseShape::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
  Q_UNUSED(option);
  Q_UNUSED(widget);
  qDebug()<<"Paint Elipse"<< xs1<<"; "<<ys1<<"; "<<xs2<<"; "<<ys2;
  int x1,x2,y1,y2;

  x1 = 0;
  y1 = 0;
  x2 = xSize();
  y2 = ySize();

  QPointF point1(x1, y1);
  QPointF point2(x2, y2);

  painter->setPen(QPen(color, 2, style));
  painter->drawEllipse(x1, y1, xSize(), ySize());
}

QPainterPath ElipseShape::shape() const
```

```
{
  QPainterPath path;
  QPointF point1(0, 0);
  QPointF point2(xSize(), ySize());
  path.addRect(QRectF(point1, point2));
  return path;
}

int ElipseShape::xSize() const
{
  return abs(xs1 - xs2);
}

int ElipseShape::ySize() const
{
  return abs(ys1 - ys2);
}
```

Текст заголовку файлу класу LineWithElipseShape (linewithelipseshape.h)

```
#ifndef LINEWITHELIPSESHAPE_H
#define LINEWITHELIPSESHAPE_H
#include "lineshape.h"
#include "elipseshape.h"

class LineWithElipseShape:  public LineShape
{
public:
  LineWithElipseShape();

  void startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene * scene) override;
  void drawRubberFootprint(QGraphicsSceneMouseEvent *event) override;
  void endDrawing() override;
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
override;


private:
  int phaze = 0;
};

#endif // LINEWITHELIPSESHAPE_H
```

Текст реалізації класу LineWithElipseShape (linewithelipseshape.cpp)

```
#include "linewithelipseshape.h"
#include <QPainter>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsScene>
```

```
LineWithElipseShape::LineWithElipseShape()
{

}

void LineWithElipseShape::startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene
*scene)
{
  phaze = 0;
  LineShape::startDrawing(event, scene);
  scene->addItem(this);
  Show(Qt::red);

}

void LineWithElipseShape::drawRubberFootprint(QGraphicsSceneMouseEvent *event)
{
  LineShape::drawRubberFootprint(event);
  phaze = 1;
}

void LineWithElipseShape::endDrawing()
{
  LineShape::endDrawing();
  phaze = 2;
}

void LineWithElipseShape::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
  Q_UNUSED(option);
  Q_UNUSED(widget);

  if(phaze == 0){
     painter->setPen(QPen(Qt::blue, 2));
     painter->drawEllipse(-5, -5,  10, 10);
     qDebug()<<"xs1: "<<xs1<<" ys1: "<<ys1<<"; "<<boundingRect();
  }
  if(phaze == 1){
     LineShape::paint(painter, option, widget);
     painter->setPen(QPen(color, 2, Qt::DashLine));
     painter->setBrush(Qt::white);
     painter->drawEllipse(x1 - 5, y1 - 5,  10,  10);
     painter->drawEllipse(x2 - 5, y2 - 5,  10,  10);
  }
  if(phaze == 2){
     LineShape::paint(painter, option, widget);
     painter->setPen(QPen(color, 2));
     painter->setBrush(Qt::white);
     painter->drawEllipse(x1 - 5, y1 - 5,  10,  10);
     painter->drawEllipse(x2 - 5, y2 - 5,  10,  10);
```

```
  }
}
```

Текст заголовку файлу класу CubeShape (cubeshape.h)

```
#ifndef CUBESHAPE_H
#define CUBESHAPE_H
#include "lineshape.h"
#include "rectshape.h"

class CubeShape:  public LineShape
{
public:
  CubeShape();

  void startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene * scene) override;
  void drawRubberFootprint(QGraphicsSceneMouseEvent *event) override;
  void endDrawing() override;
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
override;

private:
  int phaze = 0;
};

#endif // CUBESHAPE_H
```

Текст реалізації класу CubeShape (cubeshape.cpp)

```
#include "cubeshape.h"
#include <QPainter>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsScene>

CubeShape::CubeShape()
{

}

void CubeShape::startDrawing(QGraphicsSceneMouseEvent *event, QGraphicsScene *scene)
{
  phaze = 0;
  LineShape::startDrawing(event, scene);
  scene->addItem(this);
  Show(Qt::red);
}

void CubeShape::drawRubberFootprint(QGraphicsSceneMouseEvent *event)
```

```
{
  LineShape::drawRubberFootprint(event);
  phaze = 1;
}

void CubeShape::endDrawing()
{
  LineShape::endDrawing();
  phaze = 2;
}

void CubeShape::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget
*widget)
{
  Q_UNUSED(option);
  Q_UNUSED(widget);
  if(phaze == 0){
     painter->setPen(QPen(Qt::blue, 2));
     painter->drawRect(0, 0, xSize(), ySize());
  }
  if(phaze == 1){
     painter->setPen(QPen(color, 2, Qt::DashLine));
     painter->drawRect(0, 0, xSize(), ySize());

     qDebug()<<"x1: "<<x1<<" y1: "<<y1;
     qDebug()<<"x2: "<<x2<<" y2: "<<y2;

     x1 = 0;
     y1 = 0;
     x2 = xSize();
     y2 = ySize();
     painter->drawLine(x1, y1,  x1 + 50, y1 - 50);
     painter->drawLine(x1 + 50, y1 - 50, x2 + 50, y1 - 50);
     painter->drawLine(x2, y1, x2 + 50, y1 - 50);
     painter->drawLine(x2, y2, x2 + 50, y2 - 50);
     painter->drawLine(x2 + 50, y1 - 50, x2 + 50, y2 - 50);
     painter->drawLine(x1, y2, x1 + 50, y2 - 50);
     painter->drawLine(x1 + 50, y2 - 50, x2 + 50, y2 - 50);
     painter->drawLine(x1 + 50, y1 - 50, x1 + 50, y2 - 50);

  }
  if(phaze == 2){
     painter->setPen(QPen(color, 2, Qt::SolidLine));
     painter->drawRect(0, 0, xSize(), ySize());
     x1 = 0;
     y1 = 0;
     x2 = xSize();
     y2 = ySize();
     painter->drawLine(x1, y1,  x1 + 50, y1 - 50);
     painter->drawLine(x1 + 50, y1 - 50, x2 + 50, y1 - 50);
     painter->drawLine(x2, y1, x2 + 50, y1 - 50);
     painter->drawLine(x2, y2, x2 + 50, y2 - 50);
```

```
        painter->drawLine(x2 + 50, y1 - 50, x2 + 50, y2 - 50);
        painter->drawLine(x1, y2, x1 + 50, y2 - 50);
        painter->drawLine(x1 + 50, y2 - 50, x2 + 50, y2 - 50);
        painter->drawLine(x1 + 50, y1 - 50, x1 + 50, y2 - 50);

    }
}
```

Текст заголовку файлу класу MyEditor (myeditor.h)

```
#ifndef MYEDITOR_H
#define MYEDITOR_H
#include <QGraphicsScene>
#include "shape.h"

const int capasity = 129;

class MyEditor: public QGraphicsScene
{
public:
    MyEditor();

    void selectShape(DrawType shapeType);

private:
    DrawType drawType = DrawType::POINT;
    bool drawStatus = false;
    Shape *currentShape;

    int size = 0;
    Shape *objects [capasity];  //масив вказівників на об'єкти типу Shape


    // QGraphicsScene interface
protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseMoveEvent(QGraphicsSceneMouseEvent *event) override;

};

#endif // MYEDITOR_H
```

Текст реалізації класу MyEditor (myeditor.cpp)

```
#include "myeditor.h"
#include "shape.h"

#include <QGraphicsSceneMouseEvent>
```

```
MyEditor::MyEditor()
{

}

void MyEditor::selectShape(DrawType shapeType)
{
  drawType = shapeType;
}

void MyEditor::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
  if(size < capasity){
     currentShape = Shape::createShape(drawType);
     currentShape->startDrawing(event, this);
     objects[size] = currentShape;
     size++;
     //this->items()
  }
}

void MyEditor::mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
{
  currentShape->endDrawing();
  update();
}

void MyEditor::mouseMoveEvent(QGraphicsSceneMouseEvent *event)
{
  currentShape->drawRubberFootprint(event);
  update();
}
```

Текст заголовку файлу класу MyEditorView (myeditorview.h)

```
#ifndef MYEDITORVIEW_H
#define MYEDITORVIEW_H
#include <QGraphicsView>
#include <QWidget>
#include "myeditor.h"

class MyEditorView:public QGraphicsView
{
public:
  MyEditorView(QWidget *parent);
  void setEditor(MyEditor *editor);

  void drawShape(DrawType shapeType);

private:
  MyEditor *editor;
```

```
};

#endif // MYEDITORVIEW_H
```

Текст реалізації класу MyEditorView (myeditorview.cpp)

```
#include "myeditorview.h"

MyEditorView::MyEditorView(QWidget *parent):QGraphicsView(parent)
{
}

void MyEditorView::setEditor(MyEditor *editor)
{
  this->editor = editor;
  this->setScene(editor);
}

void MyEditorView::drawShape(DrawType shapeType)
{
  editor->selectShape(shapeType);
}
```

Результати тестування програми

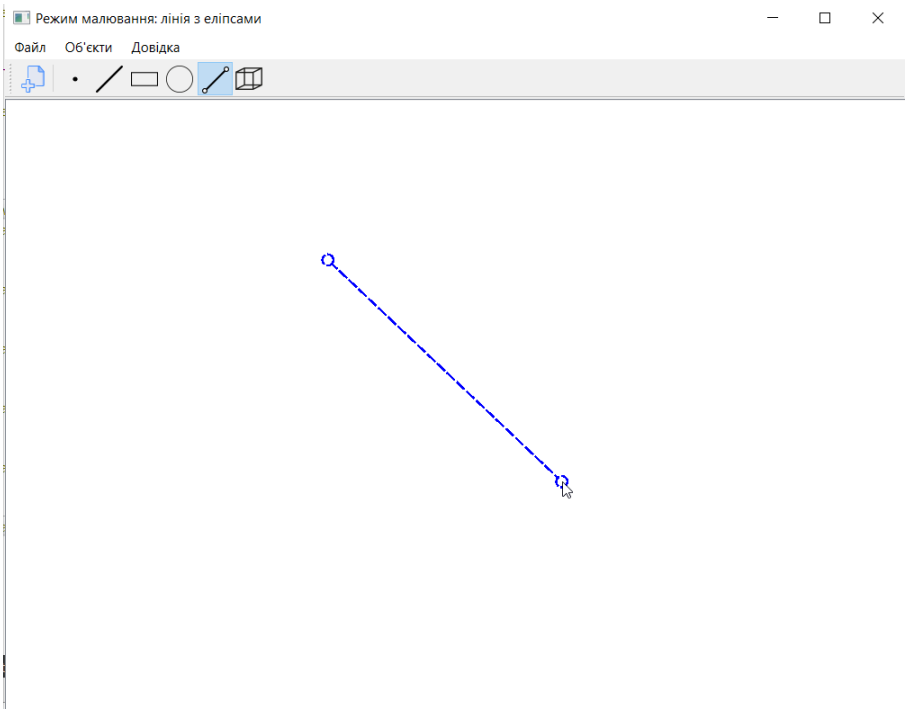


Рис 1.1: Демонстрація роботи інструменту Лінія з еліпсами, а саме малювання пунктирного гумового сліду

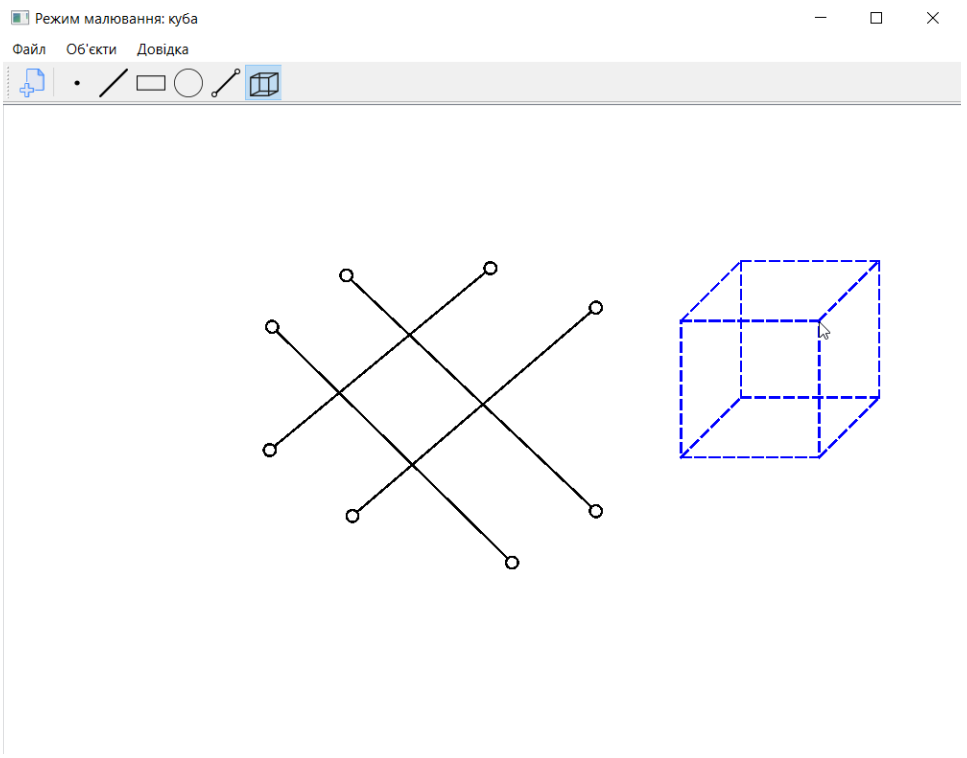


Рис 1.2: Демонстрація роботи інструменту Куб, а саме малювання пунктирного гумового сліду

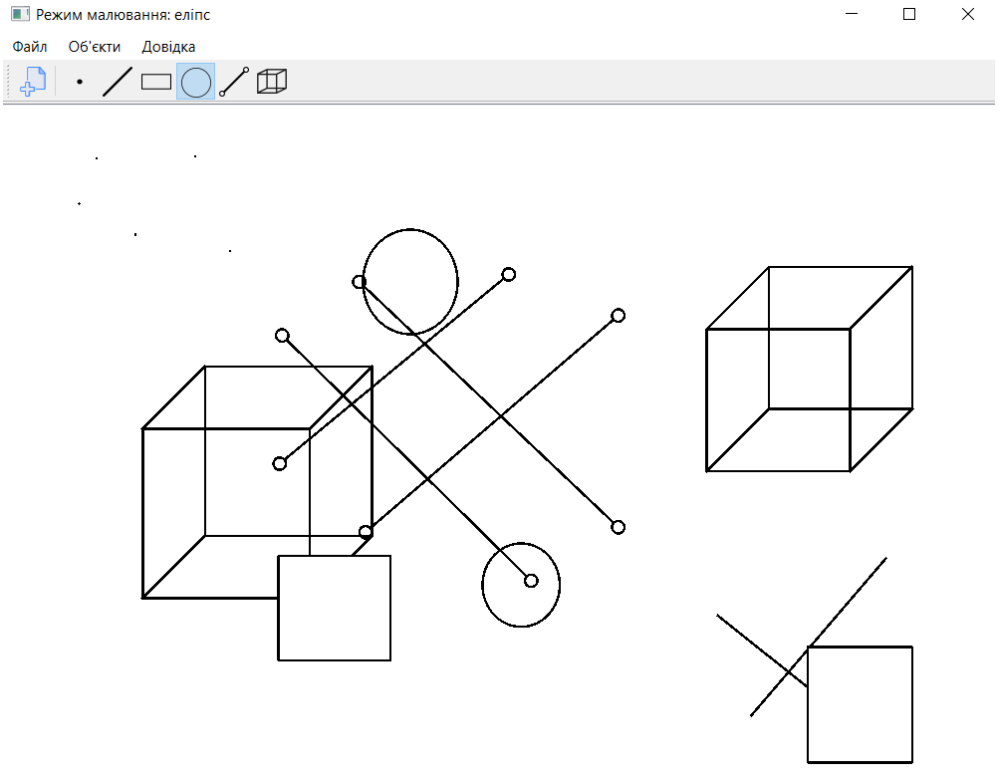
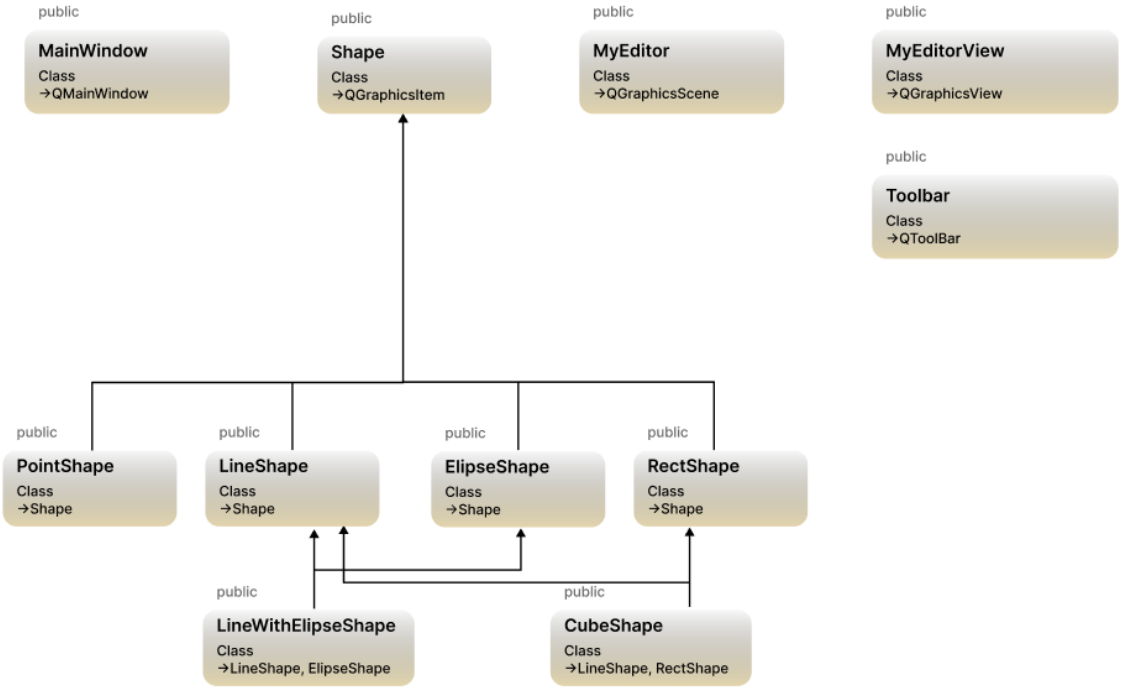


Рис 1.3: Композиція, що демонструє роботу усіх інструментів малювання

Діаграма класів

*з урахуванням особливості розробки в середовищі Qt Creator



Висновок

У цій лабораторній роботи отримано вміння та навички проектування класів, виконано модернізацію коду графічного редактора в об’єктно-орієнтованому стилі для забезпечення зручного додавання нових типів об'єктів. Додано нові типи об’єктів такі, як лінія з кружечками та каркас куба з використання мультинаслідування.