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. 6303 _____ Доброхвалов М.О.

Преподаватель _____ Спицын А.В.

Санкт-Петербург
2019

Содержание

Цель работы 3

Постановка задачи 3

Индивидуальное задание 3

Ход работы 4

Выделение элементов архитектуры MVC 4

Отображение графических фигур 7

Работа с файлами 10

Выводы 12

Список использованных источников 13

ПРИЛОЖЕНИЕ А 14

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.
3) Добавить сериализацию в файл и десериализацию из файла.

Индивидуальное задание

1) Фигуры – окружность, эллипс, текст, текст в эллипсе.
2) Контейнер – бинарное дерево.

Ход работы

Полный исходный код программы представлен в приложении А.

Выделение элементов архитектуры MVC

В качестве элемента модели представлен класс FiguresScene. Класс хранит фигуры и моделирует взаимодействие графических объектов для передачи информации компоненту отображения. Класс представлен на рисунке 1.

```
namespace Ui {
class FiguresScene;
}

class FiguresScene : public QGraphicsScene
{
    Q_OBJECT

public:
    explicit FiguresScene(QObject *parent = nullptr);
    void setFigureType(QString newFigureType);
    void setFigureRadius1(int newRadius);
    void setFigureRadius2(int newRadius);
    void setFigureFontSize(int fontSz);
    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 = "circle";
    int radius_1 = 100;
    int radius_2 = 80;
    int fontSize = 12;
    QString figureText = "\"\"";
    Shape* shape;
    std::queue<Shape*> figuresQueue;
    int figuresCount = 0;

protected:
    void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event) override;
};
```

Рисунок 1 – Класс FiguresScene

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 2.

```
class ShapeView : public QGraphicsView
{

public:

    ShapeView(QWidget *parent = 0);

protected:

    void wheelEvent(QWheelEvent *event) override;

private:

    void scale_view(qreal scale_factor);

};
```

Рисунок 2 – Класс ShapeView

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.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:

    void on_chooseObject_currentIndexChanged(int index);

    void on_lineEditText_textChanged(const QString &arg1);
    void on_fontSize_textChanged(const QString &arg1);

    void on_radius_1_textChanged(const QString &arg1);
    void on_radius_2_textChanged(const QString &arg1);

    void on_deleteButton_clicked();
    void on_clearScene_clicked();
    void on_newSceneButton_clicked();
    void on_openAction_triggered();
    void on_saveAction_triggered();

private:

    void setEnabledFields(const std::string &figure);

    Ui::MainWindow *ui;
    FiguresScene* getCurrentScene();
    QList<FiguresScene*> getAllScenes();
};
```

Рисунок 3 – Класс MainWindow

Отображение графических фигур

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

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

Рисунок 4 – Класс Shape

```
class Circle : virtual public Shape{
public:
    Circle(double x = 0, double y = 0, double r = 0);
    Circle(QDataStream& stream);
    void saveToStream(QDataStream& stream) const override;

    void print(std::ostream& out) override;
    double getRadius();
protected:
    double radius;
private:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
};
```

Рисунок 5 – Класс Circle

```
class Ellipse : virtual public Shape{
public:
    Ellipse(double x = 0, double y = 0, double r_1 = 0, double r_2 = 0);
    Ellipse(QDataStream& stream);
    void saveToStream(QDataStream& stream) const override;

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

    double getRadius1();
    double getRadius2();

protected:
    double radius_1;
    double radius_2;
private:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
};
```

Рисунок 6 – Класс Ellipse

```
class Text : virtual public Shape{
public:
    Text(double x = 0, double y = 0, const QString& text = "", int fontSize = 0);
    Text(QDataStream& stream);
    void saveToStream(QDataStream& stream) const override;

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

    QString getText();
    int getFontSize();

protected:
    QString text;
    std::size_t length;
    int fontSize;
private:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
};
```

Рисунок 7 – Класс Text

```
class TextInEllipse : virtual public Text, virtual public Ellipse{
public:
    TextInEllipse(double x = 0, double y = 0, double r_1 = 0, double r_2 = 0, const QString& newText = "", int newFontSize = 0);
    TextInEllipse(QDataStream& stream);
    void saveToStream(QDataStream& stream) const override;
    void print(std::ostream& out) override;
protected:
private:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
};
```

Рисунок 8 – Класс TextInEllipse

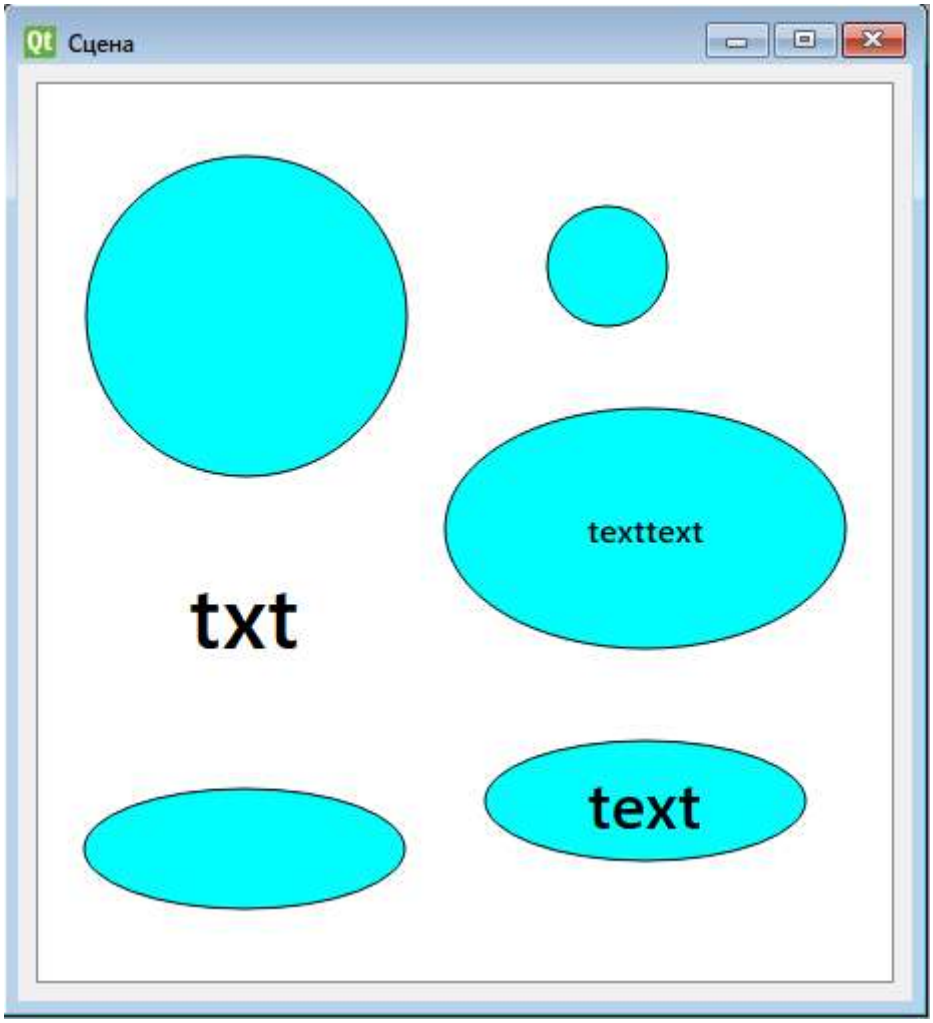

Рисунок 9 – Демонстрация отображения фигур

Для хранения фигур используется контейнер PriorityQueue из первой лабораторной работы.

Для изменения масштаба представления добавлена обработка событий колеса мыши. Обработка событий колеса мыши представлена на рисунке 10.

```
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);
}
```

Рисунок 10 – 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 11.

```
void FiguresScene::serialize(QDataStream &stream) {
    size_t cont_sz = figuresContainer.size();
    stream << cont_sz;
    while (cont_sz--) {
        Shape *fig = figuresContainer.pop()->elem();
        stream << *fig;
    }
}

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++;
            figuresContainer.push(new nodeType(figure));
        }
    }
}
```

Рисунок 11 – 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. Qt. Qt Documentation [Электронный ресурс]. URL: https://doc.qt.io/ (дата обращения: 01.12.2019).
2. Шлее М. Qt 5.10. Профессиональное программирование на C++. – СПб.: БХВ-Петербург. – 2018. – 1072 с.
3. Cppreference. Справка по C++ [Электронный ресурс]. URL: https://ru.cppreference.com/w/ (дата обращения: 02.12.2019).
4. Cplusplus. Справка по C++ [Электронный ресурс]. URL: http://www.cplusplus.com/ (дата обращения: 02.12.2019)

ПРИЛОЖЕНИЕ А

Исходный код

```
#ifndef SOFTWARE_DESIGN_BINARYTREE_H
#define SOFTWARE_DESIGN_BINARYTREE_H

#include <iostream>
#include <queue>
#include <deque>
#include "TreeNode.h"
#include "EmptyErrorBT.h"

template <typename T>
class BinaryTree {
   T *root_;

   void print_klp_(T *node);
public:
                BinaryTree();
   explicit BinaryTree(T *base);

//              BinaryTreeIterator<T> iterator();

                T* root() const;
   void print_klp();
   void clear(T* node);
   void clear();
                void push(T *);
                T* pop();

   size_t size();

   ~BinaryTree();
};

template <typename T>
BinaryTree<T>::BinaryTree(T *base) {
   if (base != nullptr)
      root_ = base;
   else
      throw EmptyErrorBT("pointer is empty");
}

template <typename T>
BinaryTree<T>::BinaryTree() {
                root_ = nullptr;
}

template <typename T>
T* BinaryTree<T>::root() const {
                return root_;
}

template <typename T>
void BinaryTree<T>::push(T *elem) {
                if (root_) {
                                std::queue<T*> current;
                                current.push(root_);
                                size_t queue_sz = 1;
                                while (queue_sz && (queue_sz & (queue_sz - 1)) == 0) {
                                                std::queue<T*> next;
                                                while (current.size()) {
                                                                auto el = current.front();
                                                                current.pop();
```

```
                                        if (el->left()) {
                                                next.push(el->left());
                                        }
                                        else {
                                                el->left(elem);
                                                return;
                                        }
                                        if (el->right()) {
                                                next.push(el->right());
                                        }
                                        else
                                        {
                                                el->right(elem);
                                                return;
                                        }
                                }
                                queue_sz = next.size();
                                current = next;
                        }
                }
                else {
                        root_ = elem;
                }
}

template <typename T>
T* BinaryTree<T>::pop() {
                if (root_) {
                        if (root_->left()) {
                                std::deque<T*> current;
                                current.push_back(root_);
                                while (true)
                                {
                                        std::deque<T*> previous;
                                        previous = current;
                                        current.clear();
                                        for (auto el : previous) {
                                                if (el->left()) {
                                                        current.push_back(el->left());
                                                }
                                                if (el->right()) {
                                                        current.push_back(el->right());
                                                }
                                        }
                                        size_t previous_sz = previous.size();
                                        size_t current_sz = current.size();

                                        if (current_sz != 2 * previous_sz) {
                                                if (current_sz % 2) {
                                                        previous[current_sz / 2]->left() = nullptr;
                                                }
                                                else {
                                                        previous[current_sz / 2]->right() = nullptr;
                                                }
                                                return current.back();
                                        }
                                        else if (!(current.front()->left())) {
                                                previous.back()->right() = nullptr;
                                                return current.back();
                                        }
                                }
                        }
                        else {
                                auto for_return = root_;
                                root_ = nullptr;
```

```
                                        return for_return;
                                }
                        }
                        else
                                throw EmptyErrorBT("pop from empty tree");
}

template <typename T>
void BinaryTree<T>::print_klp_(T* node)
{
   std::cout << node->elem() << ' ';
   if (node->left()) {
      print_klp_(node->left());
   }
   if (node->right()) {
      print_klp_(node->right());
   }
}

template <typename T>
void BinaryTree<T>::print_klp()
{
   print_klp_(root_);
}

template <typename T>
size_t BinaryTree<T>::size(){
   if(!root_)
      return 0;
   std::queue<T *> queue_;
   queue_.push(root_);
   size_t count = 0;
   while(queue_.size())
   {
      auto el = queue_.front();
      queue_.pop();
      ++count;
      if(el->left())
         queue_.push(el->left());
      if(el->right())
         queue_.push(el->right());
   }
   return count;
}

template <typename T>
void BinaryTree<T>::clear(T* node)
{
   if (node == nullptr)
      return;
   if (node->left()) {
      clear(node->left());
                                auto t = node->left();
                                t = nullptr;
   }
   if (node->right()){
      clear(node->right());
                                auto t = node->right();
                                t = nullptr;
   }
                delete node;
}

template <typename T>
void BinaryTree<T>::clear()
```

```
{
        if (root_)
                clear(root_);
        root_ = nullptr;
}

template <typename T>
BinaryTree<T>::~BinaryTree(){
  clear(root_);
        root_ = nullptr;
}

#endif //SOFTWARE_DESIGN_BINARYTREE_H

#ifndef SOFTWARE_DESIGN_BTEXCEPTION_H
#define SOFTWARE_DESIGN_BTEXCEPTION_H

#include <string>
#include <utility>
#include <exception>

class BTException {
public:
  BTException() = default;
  inline explicit BTException(const char *msg)
      : msg_(msg) {}

  const char *what() const {
    return msg_;
  }

protected:
  const char *msg_;
};

#endif //SOFTWARE_DESIGN_BTEXCEPTION_H


#ifndef CIRCLE_H
#define CIRCLE_H
#include "Shape.h"

class Circle : virtual public Shape{
public:
  Circle(double x = 0, double y = 0, double r = 0);
  Circle(QDataStream& stream);
  void saveToStream(QDataStream& stream) const override;

  double getRadius();
  void print(std::ostream& out) override;

protected:
  double radius;
private:
  void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
};

#endif // CIRCLE_H


#ifndef COLOUR_H
#define COLOUR_H

class Colour{
public:
  short r;
```

```
    short g;
    short b;
    Colour (short r, short g, short b):r(r),g(g),b(b){}
};

#endif


#ifndef ELLIPSE_H
#define ELLIPSE_H
#include "Shape.h"

class Ellipse : virtual public Shape{
public:
    Ellipse(double x = 0, double y = 0, double r_1 = 0, double r_2 = 0);
    Ellipse(QDataStream& stream);
    void saveToStream(QDataStream& stream) const override;

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

    double getRadius1();
    double getRadius2();

protected:
    double radius_1;
    double radius_2;
private:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
};

#endif // ELLIPSE_H

#ifndef SOFTWARE_DESIGN_EMPYERRORBT_H
#define SOFTWARE_DESIGN_EMPYERRORBT_H

#include "BTException.h"

class EmptyErrorBT : public BTException {
public:
    inline explicit EmptyErrorBT(const char *msg)
        : BTException(msg) {}
};

#endif //SOFTWARE_DESIGN_EMPYERRORBT_H
#ifndef FIGURESSCENE_H
#define FIGURESSCENE_H

#include <QWidget>
#include <QGraphicsScene>
#include "Shape.h"
#include "BinaryTree.h"
#include <queue>

typedef TreeNode<Shape *> nodeType;

namespace Ui {
class FiguresScene;
}

class FiguresScene : public QGraphicsScene
{
    Q_OBJECT

public:
    explicit FiguresScene(QObject *parent = nullptr);
    void setFigureType(QString newFigureType);
```

```
    void setFigureRadius1(int newRadius);
    void setFigureRadius2(int newRadius);
    void setFigureFontSize(int fontSz);
    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 = "circle";
    int radius_1 = 100;
    int radius_2 = 80;
    int fontSize = 12;
    QString figureText = "\"\"";
    Shape* shape;
//    std::queue<Shape*> figuresQueue;
    BinaryTree<nodeType> figuresContainer;
    int figuresCount = 0;

protected:
    void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event) override;
};

#endif // FIGURESSCENE_H


#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 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_chooseObject_currentIndexChanged(int index);

    void on_lineEditText_textChanged(const QString &arg1);
    void on_fontSize_textChanged(const QString &arg1);

    void on_radius_1_textChanged(const QString &arg1);
    void on_radius_2_textChanged(const QString &arg1);

    void on_deleteButton_clicked();
    void on_clearScene_clicked();
    void on_newSceneButton_clicked();
    void on_openAction_triggered();
    void on_saveAction_triggered();

private:

    void setEnabledFields(const std::string &figure);

    Ui::MainWindow *ui;
    FiguresScene* getCurrentScene();
    QList<FiguresScene*> getAllScenes();
};
#endif // MAINWINDOW_H


#ifndef POINT_H
#define POINT_H

class Point {
public:
    Point(double x, double y): x(x),y(y){}
    double x;
    double y;
};
#endif
#ifndef SHAPE_H
#define SHAPE_H
#include <QDataStream>
#include <QGraphicsObject>
#include <QGraphicsSceneMouseEvent>
#include "includes.h"

class FiguresScene;

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;
    Colour getColour() 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 = 0;
    Colour col = Colour(0,255,255);
    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 "includes.h"
#include "Shape.h"

class Text : virtual public Shape{
public:
    Text(double x = 0, double y = 0, const QString& text = "", int fontSize = 0);
    Text(QDataStream& stream);
    void saveToStream(QDataStream& stream) const override;
```

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

protected:
    QString text;
    std::size_t length;
    int fontSize;
private:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
};
#endif

#ifndef TEXT_IN_ELLIPSE_H
#define TEXT_IN_ELLIPSE_H
#include "Ellipse.h"
#include "Text.h"

class TextInEllipse : virtual public Text, virtual public Ellipse{
public:
    TextInEllipse(double x = 0, double y = 0, double r_1 = 0, double r_2 = 0, const QString& newText = "", int newFontSize = 0);
    TextInEllipse(QDataStream& stream);
    void saveToStream(QDataStream& stream) const override;
    void print(std::ostream& out) override;
protected:
private:
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) override;
};

#endif // TEXT_IN_ELLIPSE_H

#ifndef SOFTWARE_DESIGN_TREENODE_H
#define SOFTWARE_DESIGN_TREENODE_H

#include <utility>

template<typename T>
class TreeNode{
    T elem_;
    TreeNode *left_;
    TreeNode *right_;
public:

    explicit TreeNode(T base);
    TreeNode(T base, TreeNode *elem, bool is_right = false);
    TreeNode(T base, TreeNode* left, TreeNode* right);


    T &elem();
    const T &elem() const;
    void elem(T base);

    TreeNode *&left();
    void left(T left);
    void left(TreeNode *left);

    TreeNode *&right();
    void right(T right);
    void right(TreeNode *right);
     ~TreeNode();
};

template<typename T>
TreeNode<T>::TreeNode(T base) : elem_(base), left_(nullptr), right_(nullptr){}

template<typename T>
```

```
TreeNode<T>::TreeNode(T base, TreeNode* left, TreeNode* right) : elem_(base), left_(left), right_(right){}

template<typename T>
TreeNode<T>::TreeNode(T base, TreeNode *elem, bool is_right) : elem_(base) {
  if (is_right)
  {
    left_ = nullptr;
    right_ = elem;
  }
  else
  {
    left_ = elem;
    right_ = nullptr;
  }
}

template<typename T>
T &TreeNode<T>::elem() { return elem_; }

template<typename T>
const T &TreeNode<T>::elem() const { return elem_; }

template<typename T>
void TreeNode<T>::elem(T base) { elem_ = base; }

template<typename T>
TreeNode<T> *&TreeNode<T>::left() { return left_; }

template<typename T>
void TreeNode<T>::left(T left) { left_ = new TreeNode<T>(left); }

template<typename T>
void TreeNode<T>::left(TreeNode *left) { left_ = left;}

template<typename T>
TreeNode<T> *&TreeNode<T>::right() { return right_; }

template<typename T>
void TreeNode<T>::right(T right) { right_ = new TreeNode<T>(right); }

template<typename T>
void TreeNode<T>::right(TreeNode *right) {
  right_ = right;
}

template<typename T>
TreeNode<T>::~TreeNode() {}

#endif //SOFTWARE_DESIGN_TREENODE_H


#include "Circle.h"
#include "Shape.h"

Circle::Circle(double x, double y, double r) : Shape(x,y), radius(r){
  figureRect = QRectF(-r, -r, 2*r, 2*r);
}
Circle::Circle(QDataStream &stream)
  : Shape(stream) {
  stream >> radius;
}
void Circle::saveToStream(QDataStream &stream) const {
  stream << QString::fromStdString("circle");
  stream << figureRect;
  stream << QPoint(static_cast<int>(cent.x), static_cast<int>(cent.y));
```

```
    stream << scenePos();
    stream << radius;
}

void Circle::print(std::ostream& out){
    out<< "Радиус "<< radius <<"\n";
}

double Circle::getRadius() {
    return radius;
}

void Circle::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) {
    painter->setBrush(QColor(col.r, col.g, col.b));    // Устанавливаем кисть, которой будем отрисовывать объект
    painter->drawEllipse(figureRect);
    Q_UNUSED(option)
    Q_UNUSED(widget)
}

#include "Ellipse.h"
#include "Shape.h"

Ellipse::Ellipse(double x, double y, double r_1, double r_2) : Shape(x,y), radius_1(r_1), radius_2(r_2) {
    figureRect = QRectF(-r_1, -r_2, 2*r_1, 2*r_2);
}
Ellipse::Ellipse(QDataStream &stream)
    : Shape(stream) {
    stream >> radius_1 >> radius_2;
}
void Ellipse::saveToStream(QDataStream &stream) const {
    stream << QString("ellipse");
    stream << figureRect;
    stream << QPoint(static_cast<int>(cent.x), static_cast<int>(cent.y));
    stream << scenePos();
    stream << radius_1;
    stream << radius_2;
}

void Ellipse::print(std::ostream& out){
    out<< "Радиус 1 "<< radius_1 << "; Радиус 2: "<< radius_2 << "\n";
}

double Ellipse::getRadius1() {
    return radius_2;
}

double Ellipse::getRadius2() {
    return radius_1;
}

void Ellipse::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) {
    painter->setBrush(QColor(col.r, col.g, col.b));    // Устанавливаем кисть, которой будем отрисовывать объект
    painter->drawEllipse(figureRect);
    Q_UNUSED(option)
    Q_UNUSED(widget)
}

#include "FiguresScene.h"
#include "Text.h"
#include "Shape.h"
#include "Circle.h"
#include "Ellipse.h"
#include "TextInEllipse.h"
#include <QMouseEvent>
#include <QGraphicsSceneEvent>
```

```
#include <stack>

FiguresScene::FiguresScene(QObject *parent)
   : QGraphicsScene(parent) {
   setItemIndexMethod(QGraphicsScene::NoIndex);
}

FiguresScene::~FiguresScene(){
}

void FiguresScene::setFigureType(QString newFigureType) {
    typeFigure = newFigureType;
}

void FiguresScene::setFigureRadius1(int newRadius) {
   if (newRadius <= 0) {
      newRadius = 30;
   }
   radius_1 = newRadius;
}

void FiguresScene::setFigureRadius2(int newRadius) {
   if (newRadius <= 0) {
      newRadius = 40;
   }
   radius_2 = newRadius;
}

void FiguresScene::setFigureFontSize(int fontSz) {
   if (fontSz <= 12)
      fontSz = 12;
   fontSize = fontSz;
}

void FiguresScene::setFigureText(QString newText) {
   if (!newText.size()) {
      newText = "\"\"";
   }
   figureText = newText;
}

QString FiguresScene::getFigureType() const {
   return typeFigure;
}

void FiguresScene::popFigure() {
   try {
      auto fig = figuresContainer.pop()->elem();
      auto item = this->itemAt(fig->getCentCoords().x, fig->getCentCoords().y, QTransform::fromScale(1, 1));
      this->removeItem(item);
      figuresCount--;
   } catch (std::exception& e) {
      std::cout << e.what() << std::endl;
   }
}

void FiguresScene::clearSFiguresScene() {
   this->clear();
   size_t cont_sz = figuresContainer.size();
   while (cont_sz--) {
      figuresContainer.pop();
   }
}

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

```
    size_t cont_sz = figuresContainer.size();
    stream << cont_sz;
    while (cont_sz--) {
        Shape *fig = figuresContainer.pop()->elem();
        stream << *fig;
    }
}

void FiguresScene::deserialize(QDataStream &stream) {
    std::size_t figuresToLoadCount;
    stream >> figuresToLoadCount;
    if (figuresToLoadCount > 0) {
        clearSFiguresScene();
    } else {
        return;
    }
    std::stack<Shape *> st;
    for (size_t i = 0; i < figuresToLoadCount; i++) {
       Shape* figure = Shape::loadFigure(stream);
        if (figure) {
            st.push(figure);
        }
    }
    while(st.size())
    {
        Shape* figure = st.top();
        st.pop();
        this->addItem(figure);
        figuresCount++;
        figuresContainer.push(new nodeType(figure));
        std::cout << figuresCount;
    }
}


void FiguresScene::mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event) {
    auto item = this->itemAt(event->scenePos(), QTransform::fromScale(1, 1));
    if(!item) {
        auto type = this->getFigureType().toStdString();
        if (type == "text") {
            shape = new Text(event->scenePos().rx(),event->scenePos().ry(),this->figureText, this->fontSize);
        } else if (type == "circle") {
            shape = new Circle(event->scenePos().rx(),event->scenePos().ry(),this->radius_1);
        } else if (type == "ellipse") {
            shape = new Ellipse(event->scenePos().rx(), event->scenePos().ry(), this->radius_1, this->radius_2);
        } else if (type == "textInEllipse") {
            shape = new TextInEllipse(event->scenePos().rx(), event->scenePos().ry(), this->radius_1, this->radius_2, this->figureText,
this->fontSize);
        }

        shape->setPos(event->scenePos());
        this->addItem(shape);
        figuresCount++;
        figuresContainer.push(new nodeType(shape));
    }
}

#include "FiguresSceneMDI.h"
#include "ui_figuresscene.h"
#include "ShapeView.h"
#include "Text.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 "includes.h"
#include "Point.h"
#include "Colour.h"
#include "Shape.h"
#include "Text.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>

#include <iostream>

MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);
   this->setWindowTitle("Лабораторная №2");
   auto newWindow = new FiguresSceneMdi(ui->mdiArea);
   auto newFiguresScene = ui->mdiArea->addSubWindow(newWindow);
   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;
}

std::string get_figure_type(int index){
    switch (index) {
    case 0:
        return "circle";
    case 1:
        return "ellipse";
    case 2:
        return "text";
    case 3:
        return "textInEllipse";
    }
    throw std::logic_error("unknown object");
}

void MainWindow::setEnabledFields(const std::string &figure) {
    if (figure == std::string("circle"))
    {
        ui->radius_1->setEnabled(true);
        ui->radius_2->setEnabled(false);
        ui->lineEditText->setEnabled(false);
        ui->fontSize->setEnabled(false);
    }
    else if (figure == std::string("ellipse"))
    {
        ui->radius_1->setEnabled(true);
        ui->radius_2->setEnabled(true);
        ui->lineEditText->setEnabled(false);
        ui->fontSize->setEnabled(false);
    }
    else if (figure == std::string("text"))
    {
        ui->radius_1->setEnabled(false);
        ui->radius_2->setEnabled(false);
        ui->lineEditText->setEnabled(true);
        ui->fontSize->setEnabled(true);
    }
    else if (figure == std::string("textInEllipse"))
    {
        ui->radius_1->setEnabled(true);
        ui->radius_2->setEnabled(true);
        ui->lineEditText->setEnabled(true);
        ui->fontSize->setEnabled(true);
    }
    else
        throw std::logic_error("unknown object");

}

void MainWindow::on_chooseObject_currentIndexChanged(int index){

    auto scenes = getAllScenes();
    auto figure = get_figure_type(index);
```

```
    setEnabledFields(figure);

    for (auto scene : scenes)
        scene->setFigureType(QString::fromStdString(figure));
}

void MainWindow::on_radius_1_textChanged(const QString &arg1){
    auto scenes = getAllScenes();
    for (auto scene : scenes) {
        scene->setFigureRadius1(arg1.toInt());
    }
}

void MainWindow::on_radius_2_textChanged(const QString &arg1){
    auto scenes = getAllScenes();
    for (auto scene : scenes) {
        scene->setFigureRadius2(arg1.toInt());
    }
}

void MainWindow::on_lineEditText_textChanged(const QString &arg1)
{
    auto scenes = getAllScenes();
    for (auto scene : scenes) {
        scene->setFigureText(arg1);
    }
}

void MainWindow::on_fontSize_textChanged(const QString &arg1){
    auto scenes = getAllScenes();
    for (auto scene : scenes) {
        scene->setFigureFontSize(arg1.toInt());
    }
}

void MainWindow::on_deleteButton_clicked()
{
    auto scene = getCurrentScene();
    scene->popFigure();
}

void MainWindow::on_newSceneButton_clicked()
{
    auto newWindow = new FiguresSceneMdi(ui->mdiArea);
    auto newFiguresScene = ui->mdiArea->addSubWindow(newWindow);
    newFiguresScene->setWindowTitle("Сцена");

    std::string figure = get_figure_type(ui->chooseObject->currentIndex());
    newWindow->figureScene->setFigureType(QString::fromStdString(figure));

    newWindow->figureScene->setFigureRadius1(ui->radius_1->displayText().toInt());
    newWindow->figureScene->setFigureRadius2(ui->radius_2->displayText().toInt());
    newWindow->figureScene->setFigureText(ui->lineEditText->displayText());
    newWindow->figureScene->setFigureFontSize(ui->fontSize->displayText().toInt());
    setEnabledFields(figure);

    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_clearScene_clicked(){
    auto currentScene = getCurrentScene();
    currentScene->clearSFiguresScene();
}
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 "Shape.h"
#include "Text.h"
#include "Circle.h"
#include "Ellipse.h"
#include "TextInEllipse.h"
#include <QGraphicsSceneEvent>
#include <QCursor>
#include <stdexcept>

Shape::Shape(double x, double y): ang(0), cent(x,y), col(0,255,255){
    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 == "text") {
        return new Text(stream);
    } else if (type == "circle") {
        return new Circle(stream);
    } else if (type == "ellipse") {
        return new Ellipse(stream);
    } else if (type == "textInEllipse") {
        std::cout << "type: " << type.toStdString() << std::endl;
        return new TextInEllipse(stream);
    } else {
        throw std::logic_error("Incorrect figure type");
    }
}
Point Shape::getCentCoords() const {
    return cent;
}
QRectF Shape::boundingRect() const {
    return figureRect;
}
Colour Shape::getColour() const{
    return col;
}
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 QString& text, int fontSize) : Shape(x,y), text(text), fontSize(fontSize){
   length = text.length();
   figureRect = QRectF(-10/1.5,-10/2, (10*1.2)*(fontSize/12)*length, 30*(fontSize/12));
}
Text::Text(QDataStream &stream)
   : Shape(stream) {
   QString qtext;
   stream >> qtext;
   text = qtext;
   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 << text;
   stream << fontSize;
}
void Text::print(std::ostream& out){
   out<<"Длина текста "<< length<<"\n";
   out<<"Размер шрифта "<< fontSize<<"\n";
   out<<"Текст "<< text.toStdString() <<"\n";
```

```
}
void Text::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) {
    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(fontSize);
    painter->setFont(font);
    painter->drawText(figureRect, text);


    Q_UNUSED(option)
    Q_UNUSED(widget)
}
#include "TextInEllipse.h"
#include "Shape.h"

TextInEllipse::TextInEllipse(double x, double y, double r_1, double r_2, const QString& newText, int newFontSize) : Shape(x,y) {
    radius_1 = r_1;
    radius_2 = r_2;
    length = text.length();
    text = newText;
    fontSize = newFontSize;
    figureRect = QRectF(-r_1, -r_2, 2*r_1, 2*r_2);
}
TextInEllipse::TextInEllipse(QDataStream &stream)
    : Shape(stream), Ellipse(stream), Text(stream) {}


void TextInEllipse::saveToStream(QDataStream &stream) const {
    stream << QString::fromStdString("textInEllipse");
    stream << figureRect;
    stream << QPoint(static_cast<int>(cent.x), static_cast<int>(cent.y));
    stream << scenePos();
    stream << radius_1;
    stream << radius_2;
    stream << text;
    stream << fontSize;
}

void TextInEllipse::print(std::ostream& out){
    out<< "Радиус 1 "<< radius_1 << "; Радиус 2: "<< radius_2 << "\n";
    out<< "Длина текста "<< length <<"\n";
    out<< "Размер шрифта "<< fontSize<<"\n";
    out<< "Текст "<< text.toStdString() <<"\n";
}
void TextInEllipse::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget) {
    painter->setBrush(QColor(col.r, col.g, col.b));
    painter->drawEllipse(figureRect);
    painter->setPen(QPen(Qt::black, 3, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));
    auto font = painter->font();
    font.setPointSize(fontSize);
    painter->setFont(font);
    painter->drawText(figureRect, Qt::AlignCenter, text);

    Q_UNUSED(option)
    Q_UNUSED(widget)
}
```