

**МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И.УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ**

**ОТЧЁТ
по лабораторной работе №2
по дисциплине «Конструирование ПО»
Тема: Разработка приложений**

Студент гр. 6304

Преподаватель

Корытов П.В.

Спицын А.В.

Санкт-Петербург

2019

СОДЕРЖАНИЕ

1	Постановка задачи	2
1.1	Цель работы	2
1.2	Формулировка задания	2
1.3	Индивидуальное задание	2
2	Ход работы	3
2.1	Реализация MVC	3
2.1.1	Отображение содержимого контейнера	3
2.2	Отображение графических фигур	4
2.3	Сериализация	5
3	Выводы	6
	Список литературы	7
	Приложение А. Код классов иерархии фигур	8
	Приложение Б. Код контейнера	19
	Приложение В. Код классов графического интерфейса	24
	Приложение Г. Пример сериализации	32

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Цель работы

Создание приложения с графическим интерфейсом пользователя. Изучение паттерна MVC.

1.2. Формулировка задания

- Визуализировать графические фигуры из ЛР1
- Визуализировать контейнер из ЛР1
- Использовать при визуализации паттерн MVC (Model-View-Controller)
- Реализовать сериализацию и десериализацию графических фигур
- Реализовать многооконность

1.3. Индивидуальное задание

- Фигуры — пентаграмма, кусок арктангенса, текст, текст в пентаграмме.
- Контейнер — хэш-таблица на базе списка.

2. ХОД РАБОТЫ

Использованное ПО и технологии:

- **Qt Framework** [4] [3];
- **Qt Creator** — IDE для C/C++ [2];
- **X_YLA_TEX**, **neovim** — сборка и написание отчёта [1].

2.1. Реализация MVC

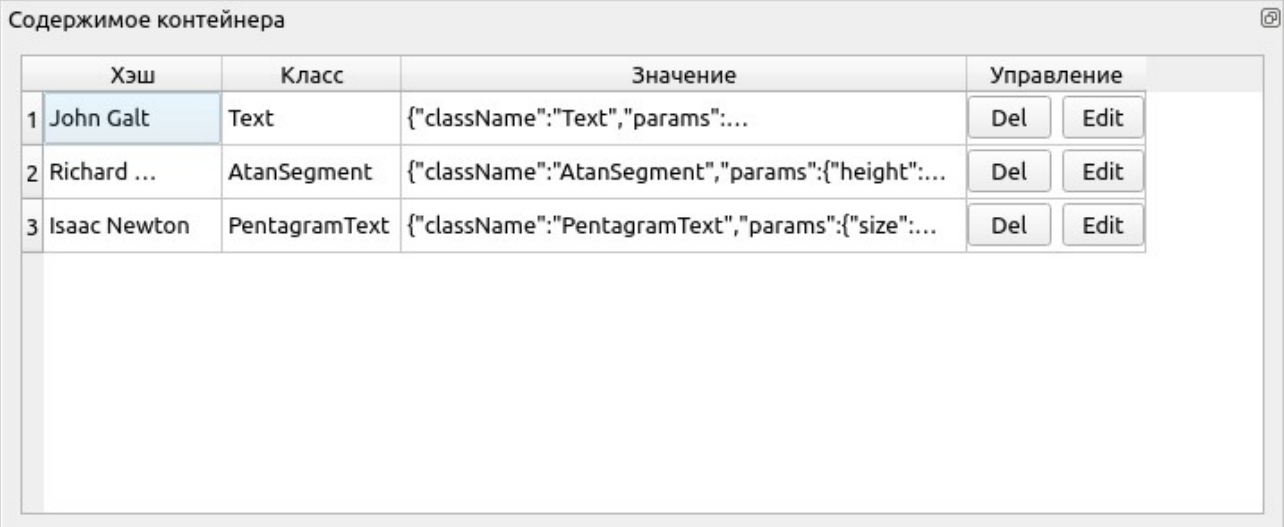
2.1.1. Отображение содержимого контейнера

Структура MVC, использованная для отображения содержимого контейнера:

- **Model** — контейнер из LP1 HashMap (код в приложении Б). Реализует логику хэш-таблицы на основе списка
- **View** — `QTableWidget`. Отображает параметры фигур, хранящихся в хэш-таблице
- **Controller** — `MainWindow`. Изменяет хэш-таблицу соответственно командам View. Код в приложении В.

В работе тип ключа — `QString`, тип значения — `QGraphicsItem*`. Хэш `QString` считается с помощью `qHash`.

Отображение контейнера представлено на рис. 1:



	Хэш	Класс	Значение	Управление	
1	John Galt	Text	{"className": "Text", "params": ...	Del	Edit
2	Richard ...	AtanSegment	{"className": "AtanSegment", "params": {"height": ...	Del	Edit
3	Isaac Newton	PentagramText	{"className": "PentagramText", "params": {"size": ...	Del	Edit

Рисунок 1 – Отображение контейнера

Добавление элемента в контейнер показано на рис. 2:

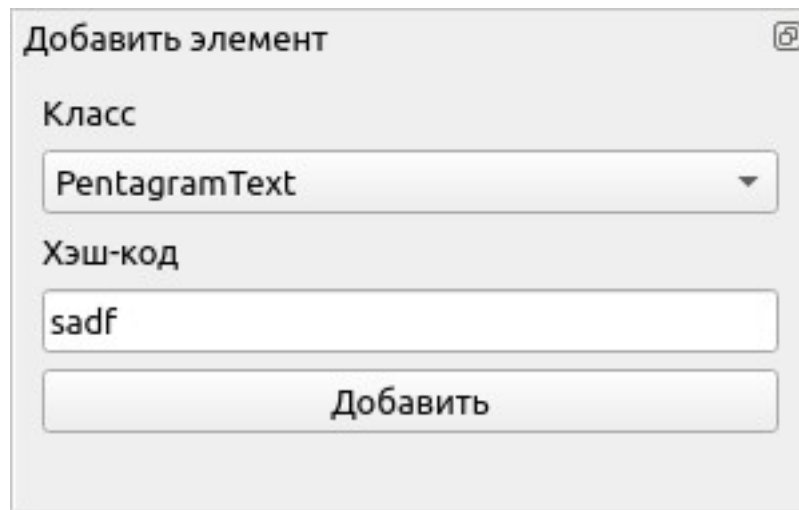


Рисунок 2 – Добавление элемента в контейнер

Настройка параметров добавляемой фигуры представлена на рис. 3:

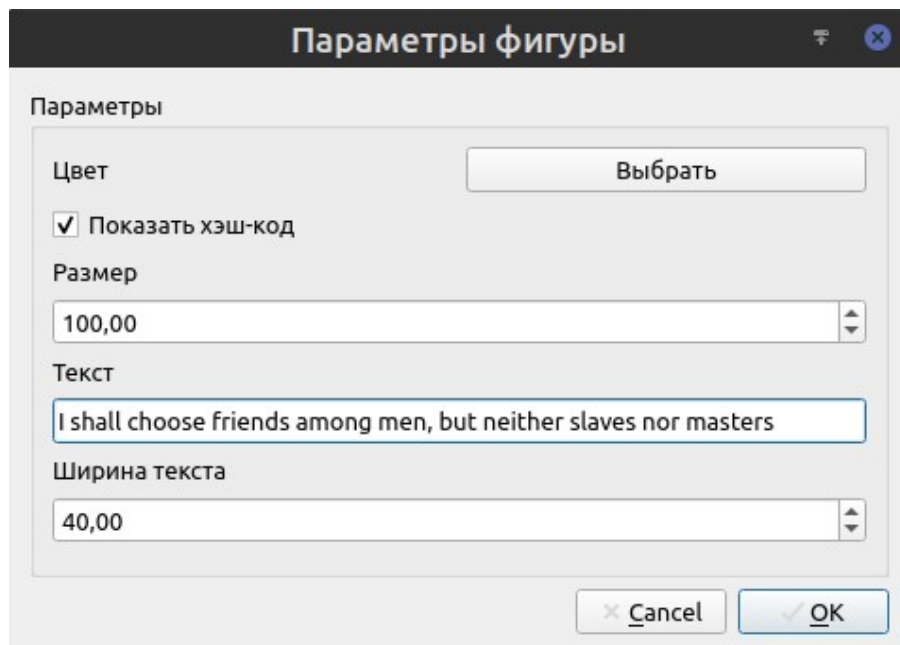


Рисунок 3 – Настройка параметров фигуры

Код диалога, представленного на рис. 3 — в приложении В.

2.2. Отображение графических фигур

Иерархия графических фигур LP1 модифицирована — класс `Shape` унаследован от `QGraphicsItem`, а `Point` — от `QPointF`. Избыточные методы — уже реализованные в классах Qt — убраны.

Методы рисования переопределены в наследниках. Коды фигур в приложении А.

`QGraphicsItem` настроен, чтобы поддерживать drag&drop.

Для отображения унаследован класс QGraphicsWidget (код в приложении В). В последнем установлен экземпляр QGraphicsScene, который композит и управляет отображением QGraphicsItem'ом.

MainWindow синхронизирует состояние контейнера и сцены, там же находятся фабрики фигур.

Пример отображения фигур представлен на 4.

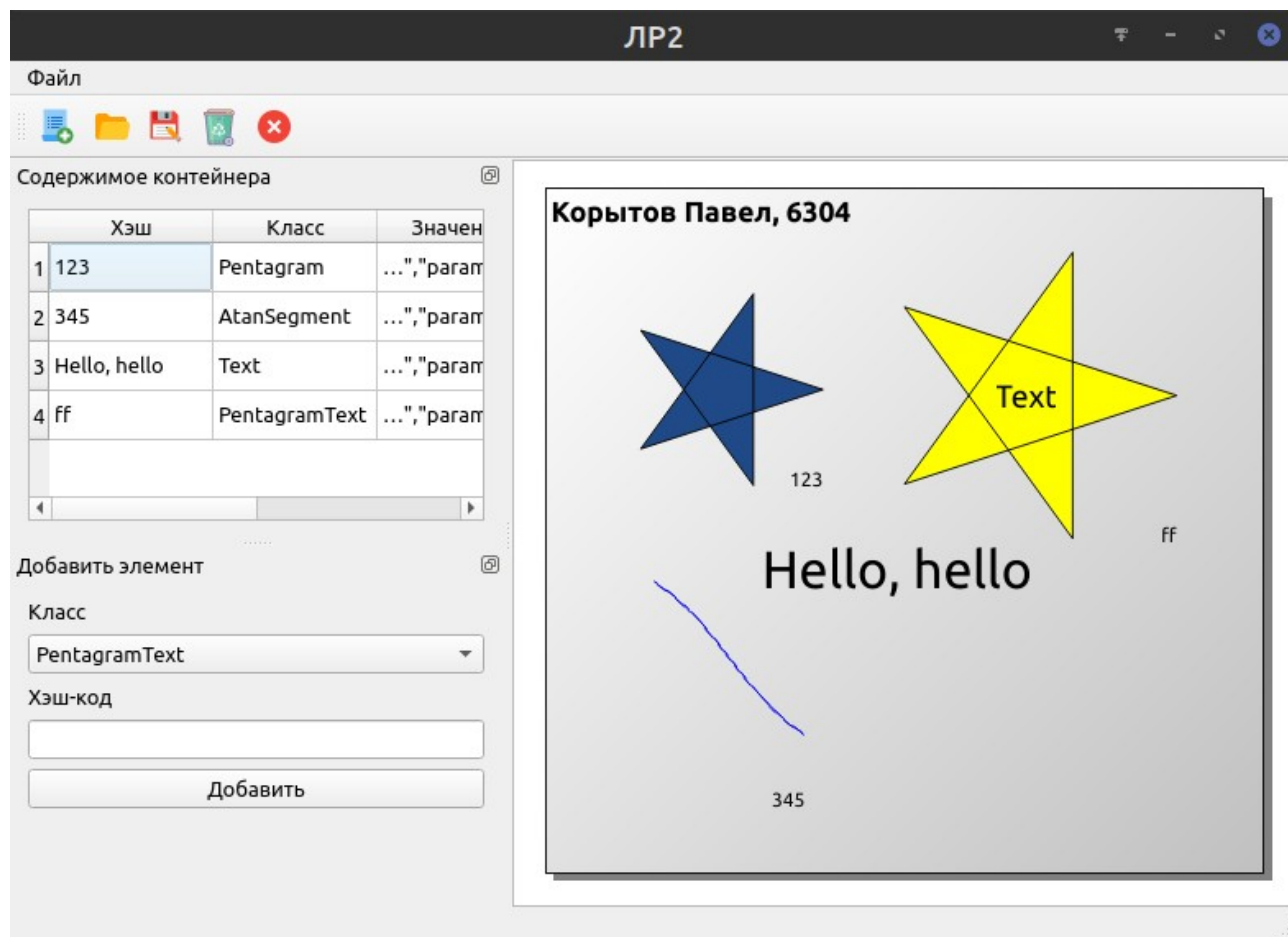


Рисунок 4 – Отображение фигур

2.3. Сериализация

Для сериализации и десериализации выбран формат JSON в реализации QJson.

У фигур определен метод toJSON, конвертирующий экземпляр класса в JSON-объект. При сохранении в ходе итерации по контейнеру вызывается этот метод, общий результат сохраняется в JSON-файл. Пример такого файла представлен в приложении Г.

При десериализации содержимое контейнера и сцены восстанавливается по файлу.

3. ВЫВОДЫ

Произведено создание графического приложения с использованием фреймворка Qt. Изучена реализация паттерна MVC в Qt, использование компонентов отображения и работы с файлами.

СПИСОК ЛИТЕРАТУРЫ

1. LaTeX - Wikibooks [Электронный ресурс]. — URL: <https://en.wikibooks.org/wiki/LaTeX> (дата обр. 16.10.2019).
2. Qt Documentation. Qt Creator Manual [Электронный ресурс]. — URL: <https://doc.qt.io/qtcreator/index.html> (дата обр. 27.11.2019).
3. Qt Documentation. Qt Reference [Электронный ресурс]. — URL: <https://doc.qt.io/qt-5/reference-overview.html> (дата обр. 27.11.2019).
4. Шлее М. Профессиональное программирование на C++ [Текст]. — СПб. : БЖХ-Петербург, 2018. — 1073 с.

ПРИЛОЖЕНИЕ А

Код классов иерархии фигур

Листинг 1. Код ../src/point.h

```
1  #ifndef POINT_H
2  #define POINT_H
3
4  #include <QPointF>
5  #include <iostream>
6
7  class Point : public QPointF
8  {
9      using QPointF::QPointF;
10 public:
11     [[nodiscard]] double getR() const;
12     [[nodiscard]] double getPhi() const;
13     void setR(double newR);
14     void setPhi(double newPhi);
15
16     friend std::ostream& operator<< (std::ostream& os, const Point& p);
17 };
18
19 #endif // POINT_H
```

Листинг 2. Код ../src/point.cpp

```
1  #include "point.h"
2
3  #include <QtMath>
4
5  double Point::getPhi() const {
6      return qAtan2(y(), x());
7  }
8
9  double Point::getR() const {
10     return qSqrt(x()*x() + y()*y());
11 }
12
13 void Point::setR(double newR) {
14     double phi = getPhi();
15     setX(newR * qCos(phi));
16     setY(newR * qSin(phi));
17 }
18
19 void Point::setPhi(double newPhi) {
20     double r = getR();
21     setX(r * qCos(newPhi));
22     setY(r * qSin(newPhi));
23 }
```

Листинг 3. Код ../src/figures/shape.h

```
1  #ifndef SHAPE_H
2  #define SHAPE_H
3
4  #include "graphwidget.h"
5  #include "point.h"
6  #include <QGraphicsItem>
7  #include <QJsonObject>
8
9  class Shape : virtual public QGraphicsItem
10 {
11 public:
12     explicit Shape();
13     virtual ~Shape() override = default;
```

```

14
15 // QGraphicsItem interface
16 virtual QRectF boundingRect() const override = 0;
17 virtual QPainterPath shape() const override = 0;
18 virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
    *option, QWidget *widget) override;
19
20 friend std::ostream &operator<<(std::ostream &os, const Shape &shape);
21
22 [[nodiscard]] virtual QString toString();
23 [[nodiscard]] virtual QJsonObject toJSON();
24 [[nodiscard]] QString getHashKey() const;
25 void setHashKey(const QString &value);
26
27 [[nodiscard]] bool getDrawHashKey() const;
28 void setDrawHashKey(bool value);
29
30 void setColor(const QColor &value);
31 QColor getColor() const;
32
33 protected:
34 virtual QColor primaryColor(const QStyleOptionGraphicsItem* option);
35 virtual void print(std::ostream &o) const = 0;
36
37 void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
38 void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override;
39 void mouseMoveEvent(QGraphicsSceneMouseEvent *event) override;
40
41 static QPolygonF getPolygon(QVector<Point> points);
42 static void mergeJsons(QJsonObject& doc, const QJsonObject& second);
43
44 private:
45     QColor color = Qt::yellow;
46     QString hashKey;
47     bool drawHashKey = true;
48
49 };
50
51 inline std::ostream &operator<<(std::ostream &os, const Shape &shape) {
52     shape.print(os);
53     return os;
54 }
55
56 #endif // SHAPE_H

```

Листинг 4. Код ../src/figures/shape.cpp

```

1 #include "shape.h"
2
3 #include <QStyleOptionGraphicsItem>
4 #include <QJsonObject>
5 #include <QJsonDocument>
6
7 Shape::Shape()
8 {
9     setFlag(ItemIsMovable);
10    setFlag(ItemSendsGeometryChanges);
11    setCacheMode(DeviceCoordinateCache);
12    setZValue(-1);
13 }
14
15 void Shape::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
    QWidget *widget)
16 {
17     if (drawHashKey) {
18         auto width = painter->fontMetrics().horizontalAdvance(hashKey);
19         auto height = painter->fontMetrics().height();
20         auto size = QSizeF(-width, -height);
21         auto rect = QRectF(boundingRect().bottomRight(), size);
22         rect = rect.normalized();
23     }

```

```

24         painter->drawText(rect, Qt::AlignRight | Qt::AlignBottom, hashKey);
25     }
26 }
27
28 QString Shape::toString()
29 {
30     QJsonDocument doc(toJSON());
31     return QString(doc.toJson(QJsonDocument::Compact));
32 }
33
34 QJsonObject Shape::toJSON()
35 {
36     return QJsonObject {
37         {"scenePos", QJsonObject {
38             {"x", scenePos().x()},
39             {"y", scenePos().y()}},
40     }
41 };
42 };
43 }
44
45 QColor Shape::primaryColor(const QStyleOptionGraphicsItem *option)
46 {
47     QColor col = QColor(color);
48     if (option->state & QStyle::State_Sunken) {
49         col = col.darker(120);
50     }
51     return col;
52 }
53
54 void Shape::mousePressEvent(QGraphicsSceneMouseEvent *event)
55 {
56     update();
57     QGraphicsItem::mousePressEvent(event);
58 }
59
60 void Shape::mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
61 {
62     update();
63     QGraphicsItem::mouseReleaseEvent(event);
64 }
65
66 void Shape::mouseMoveEvent(QGraphicsSceneMouseEvent *event)
67 {
68     QGraphicsItem::mouseMoveEvent(event);
69     auto rect = scene()->sceneRect();
70     if (x() < rect.x() - boundingRect().x()) {
71         setPos(rect.x() - boundingRect().x(), y());
72     } else if (x() > rect.width() + rect.x() - boundingRect().width() -
73         boundingRect().x()) {
74         setPos(rect.width() + rect.x() - boundingRect().width() -
75             boundingRect().x(), y());
76     }
77     if (y() < rect.y() - boundingRect().y()) {
78         setPos(x(), rect.y() - boundingRect().y());
79     } else if (y() > rect.height() + rect.y() - boundingRect().height() -
80         boundingRect().y()) {
81         setPos(x(), rect.height() + rect.y() - boundingRect().height() -
82             boundingRect().y());
83     }
84 }
85
86 QPolygonF Shape::getPolygon(QVector<Point> points)
87 {
88     auto polygon = QPolygonF();
89     for (auto point: points){
90         polygon << point;
91     }
92     return polygon;
93 }
94
95 void Shape::mergeJsons(QJsonObject &doc, const QJsonObject &second)

```

```

92 {
93     for (auto key: second.keys()) {
94         doc[key] = second[key];
95     }
96 }
97
98 QColor Shape::getColor() const
99 {
100     return color;
101 }
102
103 void Shape::setColor(const QColor &value)
104 {
105     color = value;
106 }
107
108 bool Shape::getDrawHashKey() const
109 {
110     return drawHashKey;
111 }
112
113 void Shape::setDrawHashKey(bool value)
114 {
115     drawHashKey = value;
116 }
117
118 QString Shape::getHashKey() const
119 {
120     return hashKey;
121 }
122
123 void Shape::setHashKey(const QString &value)
124 {
125     hashKey = value;
126     drawHashKey = true;
127 }

```

Листинг 5. Код ../src/figures/atansegment.h

```

1  #ifndef ATANSEGMENT_H
2  #define ATANSEGMENT_H
3
4  #include "shape.h"
5
6  class AtanSegment : virtual public Shape
7  {
8  public:
9      explicit AtanSegment(uint precision = 100, double width = 100, double
        height = 100);
10
11     // QGraphicsItem interface
12     QRectF boundingRect() const override;
13     QPainterPath shape() const override;
14     void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
        QWidget *widget) override;
15
16     // Shape interface
17     QJsonObject toJSON() override;
18
19     [[nodiscard]] uint getPrecision() const;
20     void setPrecision(const uint &value);
21
22     [[nodiscard]] double getWidth() const;
23     void setWidth(double value);
24
25     [[nodiscard]] double getHeight() const;
26     void setHeight(double value);
27
28 protected:
29     // Shape interface
30     QColor primaryColor(const QStyleOptionGraphicsItem *option) override;
31     void print(std::ostream &o) const override;

```

```

32
33 private:
34     [[nodiscard]] QVector<Point> getPath() const;
35     uint precision;
36     double width;
37     double height;
38
39     double atanStart = -1;
40     double atanStop = 1;
41     double lineWidth = 1;
42 };
43
44
45 #endif // ATANSEGMENT_H

```

Листинг 6. Код ../src/figures/atansegment.cpp

```

1  #include "atansegment.h"
2  #include <QJsonObject>
3  #include <QtMath>
4  #include <QStyleOptionGraphicsItem>
5  #include <cmath>
6
7
8  AtanSegment::AtanSegment(uint precision, double width, double height)
9      :precision(precision), width(width), height(height)
10 {
11
12 }
13
14 QRectF AtanSegment::boundingRect() const
15 {
16     qreal adjust = 2;
17     return QRectF(-width / 2 - adjust, -height / 2 - adjust,
18                   width + adjust, height + adjust);
19 }
20
21 QPainterPath AtanSegment::shape() const
22 {
23     QPainterPath path;
24     path.addRect(boundingRect());
25     return path;
26 }
27
28 void AtanSegment::paint(QPainter *painter, const QStyleOptionGraphicsItem
29 *option, QWidget *widget)
30 {
31     Shape::paint(painter, option, widget);
32     auto points = getPath();
33     QBrush brush(primaryColor(option));
34     QPen pen(brush, lineWidth);
35     painter->setPen(pen);
36     painter->setBrush(brush);
37     for (int i = 0; i < points.length() - 1; i++) {
38         painter->drawLine(points[i], points[i+1]);
39     }
40     // painter->drawRect(boundingRect());
41 }
42
43 QJsonObject AtanSegment::toJSON()
44 {
45     QJsonObject object {
46         {"className", "AtanSegment"},
47         {"params", QJsonObject {
48             {"precision", (int)precision},
49             {"width", width},
50             {"height", height}
51         }}
52     };
53     mergeJsons(object, Shape::toJSON());
54     return object;

```

```

55 }
56
57 void AtanSegment::print(std::ostream &o) const
58 {
59     o << "AtanSegment"; // TODO
60 }
61
62 QVector<Point> AtanSegment::getPath() const
63 {
64     auto path = QVector<Point>();
65     auto x = atanStart;
66     auto delta = (atanStop - atanStart) / precision;
67
68     for (uint i = 0; i < precision; i++) {
69         auto y = qAtan(x) * 2 / M_PI;
70         auto p = Point(x * width / 2, y * height / 2);
71         path.push_back(p);
72         x += delta;
73     }
74     return path;
75 }
76
77 double AtanSegment::getHeight() const
78 {
79     return height;
80 }
81
82 void AtanSegment::setHeight(double value)
83 {
84     height = value;
85 }
86
87 double AtanSegment::getWidth() const
88 {
89     return width;
90 }
91
92 void AtanSegment::setWidth(double value)
93 {
94     width = value;
95 }
96
97 uint AtanSegment::getPrecision() const
98 {
99     return precision;
100 }
101
102 void AtanSegment::setPrecision(const uint &value)
103 {
104     precision = value;
105 }
106
107 QColor AtanSegment::primaryColor(const QStyleOptionGraphicsItem *option)
108 {
109     if (option->state & QStyle::State_Sunken) {
110         return Qt::black;
111     }
112     return Qt::blue;
113 }

```

Листинг 7. Код ../src/figures/pentagram.h

```

1  #ifndef PENTAGRAM_H
2  #define PENTAGRAM_H
3
4  #include <QVector>
5  #include "figures/shape.h"
6  #include "point.h"
7
8  class Pentagram : virtual public Shape
9  {
10 public:

```

```

11     Pentagonam(double size = 100);
12
13     // QGraphicsItem interface
14     virtual QRectF boundingRect() const override;
15     virtual QPainterPath shape() const override;
16     virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
        *option, QWidget *widget) override;
17
18     double getSize() const;
19     void setSize(double value);
20
21     virtual QJsonObject toJson() override;
22
23 protected:
24     virtual void print(std::ostream &o) const override;
25     double size;
26
27 private:
28     [[nodiscard]] QVector<Point> getPath() const;
29     [[nodiscard]] QVector<Point> getPoints() const;
30 };
31
32 #endif // PENTAGRAM_H

```

Листинг 8. Код ../src/figures/pentagram.cpp

```

1  #include "pentagram.h"
2  #include <cmath>
3  #include <QJsonObject>
4  #include <QPainter>
5  #include <QStyleOption>
6  #include <QPolygonF>
7
8  Pentagonam::Pentagram(double size)
9      :size(size)
10 {
11 }
12
13 QRectF Pentagonam::boundingRect() const
14 {
15     qreal adjust = 2;
16     return QRectF(-size - adjust, -size - adjust,
17                   size * 2 + adjust, size * 2 + adjust);
18 }
19
20 QPainterPath Pentagonam::shape() const
21 {
22     QPainterPath path;
23     path.addPolygon(getPolygon(getPoints()));
24     return path;
25 }
26
27 void Pentagonam::paint(QPainter *painter, const QStyleOptionGraphicsItem
    *option, QWidget *widget)
28 {
29     Shape::paint(painter, option, widget);
30     painter->setPen(QPen(Qt::black));
31     painter->setBrush(primaryColor(option));
32     auto path = getPath();
33     auto polygon = getPolygon(path);
34     painter->drawPolygon(polygon, Qt::WindingFill);
35 }
36
37 void Pentagonam::print(std::ostream &o) const
38 {
39     o << "Pentagram";
40 }
41
42 double Pentagonam::getSize() const
43 {
44

```

```

45     return size;
46 }
47
48 void Pentagon::setSize(double value)
49 {
50     size = value;
51 }
52
53 QJsonObject Pentagon::toJSON()
54 {
55     QJsonObject object {
56         {"className", "Pentagram"},
57         {"params", QJsonObject {
58             {"size", size}
59         }}
60     };
61     mergeJsons(object, Shape::toJSON());
62     return object;
63 }
64
65 QVector<Point> Pentagon::getPoints() const
66 {
67     auto points = QVector<Point>();
68     for (int i = 0; i < 5; i++) {
69         auto p = Point {size, 0};
70         p.setPhi(i * M_PI * 2 / 5);
71         points.push_back(p);
72     }
73     return points;
74 }
75
76 QVector<Point> Pentagon::getPath() const
77 {
78     auto points = getPoints();
79     return QVector<Point> {
80         points[0],
81         points[2],
82         points[4],
83         points[1],
84         points[3],
85         points[0]
86     };
87 }
88

```

Листинг 9. Код ../src/figures/text.h

```

1  #ifndef TEXT_H
2  #define TEXT_H
3
4  #include "shape.h"
5
6  class Text : virtual public Shape
7  {
8  public:
9      explicit Text(QString string = "", double width = 40);
10
11      // QGraphicsItem interface
12      virtual QRectF boundingRect() const override;
13      virtual QPainterPath shape() const override;
14      virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
          *option, QWidget *widget) override;
15
16      [[nodiscard]] QString getString() const;
17      void setString(const QString &value);
18
19      [[nodiscard]] double getWidth() const;
20      void setWidth(double value);
21
22      [[nodiscard]] double getHeight() const;
23      void setHeight(double value);

```



```

24
25     virtual QJsonObject toJSON() override;
26 protected:
27     virtual void print(std::ostream &o) const override;
28
29     QString string;
30     double width;
31     double height;
32
33 private:
34     QColor textColor(const QStyleOptionGraphicsItem* option);
35
36 };
37
38 #endif // TEXT_H

```

Листинг 10. Код ../src/figures/text.cpp

```

1  #include "text.h"
2  #include <QStyleOptionGraphicsItem>
3  #include <QJsonObject>
4
5  Text::Text(QString string, double width)
6      :string(string), width(width)
7  {
8      height = width / string.length() * 8;
9  }
10
11 QRectF Text::boundingRect() const
12 {
13     qreal adjust = 2;
14     return QRectF(-adjust - width / 2, -adjust - height / 2,
15                  width + adjust, height + adjust);
16 }
17
18 QPainterPath Text::shape() const
19 {
20     QPainterPath path;
21     path.addRect(boundingRect());
22     return path;
23 }
24
25 void Text::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
26                 QWidget *widget)
27 {
28     Shape::paint(painter, option, widget);
29     auto factor = width / painter->fontMetrics().horizontalAdvance(string);
30     auto font = painter->font();
31     if (string.length() == 1) {
32         factor *= 0.5;
33     }
34     font.setPointSizeF(font.pointSizeF() * factor);
35     height = painter->fontMetrics().height() * font.pointSizeF() / 5;
36     QRectF rect(-width / 2, -height / 4, width, height / 2);
37
38     painter->setFont(font);
39     painter->setPen(QPen(textColor(option)));
40     painter->drawText(rect, Qt::AlignHCenter | Qt::AlignVCenter, string);
41     // painter->drawRect(rect);
42 }
43
44 void Text::print(std::ostream &o) const
45 {
46     o << "Text"; // TODO
47 }
48
49 double Text::getHeight() const
50 {
51     return height;
52 }
53
54 void Text::setHeight(double value)

```

```

54 {
55     height = value;
56 }
57
58 QJsonObject Text::toJSON()
59 {
60     QJsonObject object {
61         {"className", "Text"},
62         {"params", QJsonObject {
63             {"string", string},
64             {"width", width}
65         }}
66     };
67     mergeJsons(object, Shape::toJSON());
68     return object;
69 }
70
71
72 double Text::getWidth() const
73 {
74     return width;
75 }
76
77 void Text::setWidth(double value)
78 {
79     width = value;
80 }
81
82 QString Text::getString() const
83 {
84     return string;
85 }
86
87 void Text::setString(const QString &value)
88 {
89     string = value;
90 }
91
92 QColor Text::textColor(const QStyleOptionGraphicsItem *option)
93 {
94     if (option->state & QStyle::State_Sunken) {
95         return Qt::gray;
96     }
97     return Qt::black;
98 }

```

Листинг 11. Код ../src/figures/pentagramtext.h

```

1  #ifndef PENTAGRAMTEXT_H
2  #define PENTAGRAMTEXT_H
3
4  #include "figures/pentagram.h"
5  #include "figures/text.h"
6
7  class PentagramText: virtual public Pentagram, virtual public Text
8  {
9  public:
10     PentagramText(QString string="", double size = 100);
11
12     // QGraphicsItem interface
13 public:
14     virtual QRectF boundingRect() const override;
15     virtual QPainterPath shape() const override;
16     virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
        *option, QWidget *widget) override;
17
18     virtual QJsonObject toJSON() override;
19 protected:
20     virtual void print(std::ostream &o) const override;
21 };
22

```

23 **#endif** // PENTAGRAMTEXT_H

Листинг 12. Код ../src/figures/pentagramtext.cpp

```
1  #include "pentagramtext.h"
2  #include <QJsonObject>
3
4  PentagonText::PentagramText(QString string, double size)
5      :Pentagram(size), Text(string, size / 2)
6  {
7
8  }
9
10 QRectF PentagonText::boundingRect() const
11 {
12     return Pentagon::boundingRect();
13 }
14
15 QPainterPath PentagonText::shape() const
16 {
17     return Pentagon::shape();
18 }
19
20 void PentagonText::paint(QPainter *painter, const QStyleOptionGraphicsItem
    *option, QWidget *widget)
21 {
22     Pentagon::paint(painter, option, widget);
23     auto drawHashKey = getDrawHashKey();
24     setDrawHashKey(false);
25     Text::paint(painter, option, widget);
26     setDrawHashKey(drawHashKey);
27 }
28
29 QJsonObject PentagonText::toJSON()
30 {
31     QJsonObject params {};
32     mergeJsons(params, Pentagon::toJSON()["params"].toObject());
33     mergeJsons(params, Text::toJSON()["params"].toObject());
34     QJsonObject object {
35         {"className", "PentagramText"},
36         {"params", params}
37     };
38     mergeJsons(object, Shape::toJSON());
39     return object;
40 }
41
42 void PentagonText::print(std::ostream &o) const
43 {
44     o << "PentagramText"; // TODO
45 }
```

ПРИЛОЖЕНИЕ Б

Код контейнера

Листинг 13. Код ../src/hashMap.h

```
1  #pragma once
2
3  #include "exception.h"
4  #include <algorithm>
5  #include <iostream>
6  #include <functional>
7  #include <string>
8
9  #define SIZE 2000000
10
11 template<typename Key>
12 unsigned int defaultHash(Key key) {
13     return static_cast<unsigned int>(key) % SIZE;
14 }
15
16 template<typename Key>
17 bool defaultCompare(const Key& v1, const Key& v2) {
18     return v1 == v2;
19 }
20
21 template <typename Key, typename Value>
22 class HashMap;
23
24 template <typename Key, typename Value>
25 class HashNode {
26 public:
27     explicit HashNode(const Key& key, const Value& value): key(key),
28         value(value) {}
29
30     HashNode (HashNode& node): key(node.key), value(node.value) {}
31
32     friend void swap(HashNode& a, HashNode& b){
33         std::swap(a.key, b.key);
34         std::swap(a.value, b.value);
35     }
36
37     HashNode& operator=(const HashNode& other) {
38         swap(*this, other);
39         return *this;
40     }
41
42     HashNode& operator=(HashNode&& other) noexcept {
43         swap(*this, other);
44     }
45
46     [[nodiscard]] const Value& getValue() {
47         return value;
48     }
49
50     void setValue(Value newValue) {
51         value = newValue;
52     }
53
54     [[nodiscard]] const Key& getKey() {
55         return key;
56     }
57
58     [[nodiscard]] HashNode *getNext() const {
59         return next;
60     }
61
62     void setNext(HashNode *newNext) {
63         next = newNext;
64     }
```

```

65 private:
66     Key key;
67     Value value;
68     HashNode* next = nullptr;
69 };
70
71 template <typename Key, typename Value>
72 class HashMapIterator {
73 public:
74     explicit HashMapIterator(const HashMap<Key, Value>& hashMap,
75                             HashNode<Key, Value>* node = nullptr)
76         : hashMap(hashMap), currentNode(node),
77           compareFunc(hashMap.compareFunc) {
78         if (node == nullptr) {
79             getNext();
80         } else {
81             currentValue = hashMap.hashFunc(node->getKey());
82         }
83     }
84     HashMapIterator(HashMapIterator& it)
85         : HashMapIterator(it.hashMap, it.currentNode)
86     {
87         compareFunc = it.compareFunc;
88     }
89     HashMapIterator& operator++() {
90         getNext();
91         return *this;
92     }
93
94     HashMapIterator operator++(int) {
95         auto ret = HashMapIterator(*this);
96         getNext();
97         return ret;
98     }
99
100     const Key& key() {
101         return currentNode->getKey();
102     }
103
104     const Value& value() {
105         return currentNode->getValue();
106     }
107
108     bool end() {
109         return currentValue >= SIZE - 1;
110     }
111
112     friend bool operator==(const HashMapIterator<Key, Value>& it1,
113                           const HashMapIterator<Key, Value>& it2) {
114         return it1.compareFunc(it1.currentNode->getKey(),
115                                it2.currentNode->getKey());
116     }
117
118     friend bool operator!=(const HashMapIterator<Key, Value>& it1,
119                           const HashMapIterator<Key, Value>& it2) {
120         return !it1.compareFunc(it1.currentNode->getKey(),
121                                 it2.currentNode->getKey());
122     }
123
124     friend bool operator<(const HashMapIterator<Key, Value>& it1,
125                          const HashMapIterator<Key, Value>& it2) {
126         return it1.currentValue < it2.currentValue;
127     }
128
129     friend bool operator<=(const HashMapIterator<Key, Value>& it1,
130                           const HashMapIterator<Key, Value>& it2) {
131         if (it1.currentNode == nullptr && it2.currentNode == nullptr) {
132             return false;
133         }
134         return it1.currentValue <= it2.currentValue;

```

```

134     }
135
136 private:
137     void getNext() {
138         if (currentNode != nullptr) {
139             currentNode = currentNode->getNext();
140         }
141         while (currentNode == nullptr && (currentValue < SIZE - 1)) {
142             currentValue++;
143             currentNode = hashMap.table[currentValue];
144         }
145         if (currentNode == nullptr && currentValue >= SIZE - 1) {
146             currentValue++;
147         }
148     }
149
150     std::function<int(Key, Key)> compareFunc;
151     const HashMap<Key, Value>& hashMap;
152     HashNode<Key, Value>* currentNode = nullptr;
153     unsigned int currentValue = 0;
154 };
155
156
157 template<typename Key, typename Value>
158 class HashMap {
159     template <typename K, typename V>
160     friend class HashMapIterator;
161 public:
162     explicit HashMap(
163         const std::function<unsigned int(Key)>& hash = defaultHash<Key>,
164         const std::function<int(Key, Key)>& compare = defaultCompare<Key>
165         : hashFunc(hash), compareFunc(compare) {
166         table = new HashNode<Key, Value>* [SIZE]();
167     }
168
169     ~HashMap() {
170         clear();
171         delete[] table;
172     }
173
174     void clear() {
175         for (unsigned int i = 0; i < SIZE; i++) {
176             HashNode<Key, Value> *entry = table[i];
177             while (entry != nullptr) {
178                 HashNode<Key, Value> *prev = entry;
179                 entry = entry->getNext();
180                 delete prev;
181             }
182         }
183         delete[] table;
184         table = new HashNode<Key, Value>* [SIZE]();
185     }
186
187     const Value& at(const Key& key) {
188         unsigned int hashValue = hashFunc(key);
189         HashNode<Key, Value>* node = table[hashValue];
190
191         while (node != nullptr) {
192             if (compareFunc(node->getKey(), key)) {
193                 return node->getValue();
194             }
195             node = node->getNext();
196         }
197         throw ElementNotFoundException<Key>(key);
198     }
199
200     bool get(const Key& key, Value& value) {
201         try {
202             value = at(key);
203             return true;
204         } catch (ElementNotFoundException<Key>& ex) {
205             return false;

```

```

206     }
207 }
208
209 void create(const Key& key, const Value& value) {
210     auto [prev, entry, hashValue] = getEntry(key);
211
212     if (entry == nullptr) {
213         entry = new HashNode<Key, Value>(key, value);
214         if (prev == nullptr) {
215             table[hashValue] = entry;
216         } else {
217             prev->setNext(entry);
218         }
219     } else {
220         throw ElementAlreadyExistsException<Key, Value>(key,
221             entry->getValue());
222     }
223 }
224
225 void update(const Key& key, const Value& value) {
226     auto [prev, entry, hashValue] = getEntry(key);
227
228     if (entry == nullptr) {
229         entry = new HashNode<Key, Value>(key, value);
230         if (prev == nullptr) {
231             table[hashValue] = entry;
232         } else {
233             prev->setNext(entry);
234         }
235     } else {
236         entry->setValue(value);
237     }
238 }
239
240 void remove(const Key& key) {
241     auto [prev, entry, hashValue] = getEntry(key);
242
243     if (entry == nullptr) {
244         throw ElementNotFoundException<Key>(key);
245     } else {
246         if (prev == nullptr) {
247             table[hashValue] = entry->getNext();
248         } else {
249             prev->setNext(entry->getNext());
250         }
251         delete entry;
252     }
253 }
254
255 HashMapIterator<Key, Value> begin(){
256     return HashMapIterator<Key, Value>(*this);
257 }
258
259 HashMapIterator<Key, Value> end(){
260     unsigned int lastHash;
261     for (lastHash = SIZE - 1; table[lastHash] == nullptr && lastHash > 0;
262         lastHash--);
263
264     if (lastHash > 0) {
265         HashNode<Key, Value>* node = table[lastHash];
266         while (node->getNext() != nullptr) {
267             node = node->getNext();
268         }
269         return HashMapIterator<Key, Value>(*this, node);
270     }
271     return HashMapIterator<Key, Value>(*this);
272 }
273
274 private:
275     std::tuple<HashNode<Key, Value>*, HashNode<Key, Value>*,
276         unsigned int> getEntry(const Key& key) {
277         unsigned int hashValue = hashFunc(key);

```

```

276     HashNode<Key, Value> *entry = table[hashValue];
277     HashNode<Key, Value>* prev = nullptr;
278     while (entry != nullptr && !compareFunc(entry->getKey(), key)) {
279         prev = entry;
280         entry = entry->getNext();
281     }
282     return std::make_tuple(prev, entry, hashValue);
283 }
284
285 HashNode<Key, Value> **table;
286 std::function<unsigned int(Key)> hashFunc;
287 std::function<int(Key, Key)> compareFunc;
288 };

```

Листинг 14. Код ../src/exception.h

```

1  # pragma once
2
3  #include <QTextStream>
4
5  class Exception {
6  public:
7      friend std::ostream &operator<<(std::ostream &os, const Exception &ex);
8      virtual ~Exception() = default;
9      virtual QString toString() const = 0;
10 };
11
12 inline std::ostream &operator<<(std::ostream &os, const Exception &ex) {
13     os << ex.toString().toString();
14     return os;
15 }
16
17 template <typename Key>
18 class ElementNotFoundException: public Exception {
19 public:
20     explicit ElementNotFoundException(Key key) : key(key) {}
21
22 protected:
23     QString toString() const override {
24         QString string;
25         QTextStream outStream(&string);
26         outStream << "ElementNotFoundException{Key:_" << key << "}";
27         return string;
28     }
29
30 private:
31     Key key;
32 };
33
34 template <typename Key, typename Value>
35 class ElementAlreadyExistsException: public Exception {
36 public:
37     ElementAlreadyExistsException(Key key, const Value &element) : key(key),
38         value(element) {}
39
40 protected:
41     QString toString() const override {
42         QString string;
43         QTextStream outStream(&string);
44         outStream << "ElementAlreadyExistsException{Key:_" << key << ",_
45             Value:" << value << "}";
46         return string;
47     }
48
49 private:
50     Key key;
51     Value value;
52 };

```


ПРИЛОЖЕНИЕ В

Код классов графического интерфейса

Листинг 15. Код ../src/main.cpp

```
1  #include "mainwindow.h"
2  #include "exception.h"
3  #include <QApplication>
4  #include <QMessageBox>
5  #include <QTextStream>
6  #include <QString>
7
8  class MyApplication: virtual public QApplication {
9      // QCoreApplication interface
10 public:
11     using QApplication::QApplication;
12     bool notify(QObject *receiver, QEvent *event) override {
13         try {
14             return QApplication::notify(receiver, event);
15         } catch (Exception& e) {
16             QMessageBox::critical(messageBoxWidget, "Исключение",
17                                   e.toString());
18             return true;
19         }
20         return false;
21     }
22     void setMessageBoxWidget(QWidget *value) {
23         messageBoxWidget = value;
24     }
25 private:
26     QWidget* messageBoxWidget;
27 };
28
29 int main(int argc, char *argv[])
30 {
31     MyApplication a(argc, argv);
32     MainWindow* w = new MainWindow();
33     w->setAttribute(Qt::WA_DeleteOnClose, true);
34     w->show();
35     a.setMessageBoxWidget(w);
36     return a.exec();
37 }
```

Листинг 16. Код ../src/mainwindow.h

```
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QMainWindow>
5  #include "graphwidget.h"
6  #include "hashMap.h"
7
8  namespace Ui {
9      class MainWindow;
10 }
11
12 class MainWindow : public QMainWindow
13 {
14     Q_OBJECT
15 public:
16     explicit MainWindow(QWidget *parent = nullptr);
17     ~MainWindow();
18
19 private slots:
20     void on_addFigureButton_clicked();
21 }
```

```

22     void on_itemAdded(QGraphicsItem* item);
23     void addItem(QGraphicsItem* item, QString hash, QString className);
24     void on_itemDelete(QString hash);
25     void on_itemEdit(QString hash);
26     void on_itemEdited(QGraphicsItem* item);
27
28     void on_actionSave_triggered();
29
30     void on_actionClear_triggered();
31
32     void on_actionOpen_triggered();
33
34     void on_actionNew_triggered();
35
36 private:
37     QMap<QString, QGraphicsItem*> hashMap;
38     Ui::MainWindow *ui;
39     QGraphicsWidget* widget;
40 };
41
42 #endif // MAINWINDOW_H

```

Листинг 17. Код ../src/mainwindow.cpp

```

1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3  #include "adddialog.h"
4  #include "figures/shape.h"
5  #include "figures/pentagram.h"
6  #include "figures/atansegment.h"
7  #include "figures/pentagramtext.h"
8  #include "figures/text.h"
9
10 #include <QFileDialog>
11 #include <QJsonDocument>
12 #include <QTableWidgetItem>
13 #include <QMessageBox>
14
15
16 uint hashFunc(QString string) {
17     return qHash(string) % SIZE;
18 }
19
20 MainWindow::MainWindow(QWidget *parent) :
21     QMainWindow(parent), hashMap(hashFunc), ui(new Ui::MainWindow)
22 {
23     ui->setupUi(this);
24     widget = new QGraphicsWidget(this);
25     setCentralWidget(widget);
26 }
27
28 MainWindow::~MainWindow()
29 {
30     delete ui;
31 }
32
33 void MainWindow::on_addFigureButton_clicked()
34 {
35     auto dialog = new AddDialog(ui->classComboBox->currentText(), nullptr,
36                                 this);
37     connect(dialog, &AddDialog::itemChanged, this, &MainWindow::on_itemAdded);
38     dialog->setAttribute(Qt::WA_DeleteOnClose, true);
39     dialog->show();
40 }
41
42 void MainWindow::on_itemAdded(QGraphicsItem *item)
43 {
44     auto hash = ui->hashCodeEdit->text();
45     addItem(item, hash, ui->classComboBox->currentText());
46 }

```

```

47 void MainWindow::addItem(QGraphicsItem *item, QString hash, QString className)
48 {
49     hashMap.create(hash, item);
50
51     Shape* shape = dynamic_cast<Shape*>(item);
52     shape->setHashKey(hash);
53     widget->scene()->addItem(shape);
54     shape->setPos(widget->centerPos());
55     shape->update();
56
57     auto newRow = ui->tableWidget->rowCount();
58     ui->tableWidget->insertRow(newRow);
59     ui->tableWidget->setItem(newRow, 0, new QTableWidgetItem(hash));
60     ui->tableWidget->setItem(newRow, 1, new QTableWidgetItem(className));
61     ui->tableWidget->setItem(newRow, 2, new
        QTableWidgetItem(shape->toString()));
62
63     QWidget* controlWidget = new QWidget();
64     QHBoxLayout* layout = new QHBoxLayout();
65     QPushButton* deleteButton = new QPushButton("Del");
66     QPushButton* editButton = new QPushButton("Edit");
67     layout->addWidget(deleteButton);
68     layout->addWidget(editButton);
69     layout->setAlignment(Qt::AlignCenter);
70     layout->setContentsMargins(0, 0, 0, 0);
71     controlWidget->setLayout(layout);
72     ui->tableWidget->setCellWidget(newRow, 3, controlWidget);
73
74     connect(deleteButton, &QPushButton::clicked, this, [=]() {
75         this->on_itemDelete(hash); });
76     connect(editButton, &QPushButton::clicked, this, [=]() {
77         this->on_itemEdit(hash); });
78     ui->hashCodeEdit->clear();
79     widget->scene()->update();
80 }
81 void MainWindow::on_itemDelete(QString hash)
82 {
83     QGraphicsItem* item = hashMap.at(hash);
84     widget->scene()->removeItem(item);
85     hashMap.remove(hash);
86     for (int i = 0; i < ui->tableWidget->rowCount(); i++) {
87         if (ui->tableWidget->item(i, 0)->text() == hash) {
88             ui->tableWidget->removeRow(i);
89             break;
90         }
91     }
92 }
93 void MainWindow::on_itemEdit(QString hash)
94 {
95     QGraphicsItem* item = hashMap.at(hash);
96
97     auto dialog = new AddDialog(ui->classComboBox->currentText(), item, this);
98     connect(dialog, &AddDialog::itemChanged, this, &MainWindow::on_itemEdited);
99     dialog->setAttribute(Qt::WA_DeleteOnClose, true);
100     dialog->show();
101 }
102 void MainWindow::on_itemEdited(QGraphicsItem *item)
103 {
104     item->update();
105     widget->scene()->update();
106 }
107 void MainWindow::on_actionSave_triggered()
108 {
109     QJsonObject object;
110     for (auto it = hashMap.begin(); it <= hashMap.end(); it++){
111         Shape* shape = dynamic_cast<Shape*>(it.value());

```

```

115         object[it.key()] = shape->toJSON();
116     }
117     QJsonDocument doc(object);
118     QString fileName = QFileDialog::getSaveFileName(this, "Сохранить",
119         "data.json", "JSON_(*.json)");
120     QFile file(fileName);
121     file.open(QIODevice::WriteOnly);
122     file.write(doc.toJson());
123     file.close();
124 }
125
126 void MainWindow::on_actionClear_triggered()
127 {
128     ui->tableWidget->setRowCount(0);
129     widget->scene()->clear();
130     hashMap.clear();
131 }
132
133 void MainWindow::on_actionOpen_triggered()
134 {
135     QString fileName = QFileDialog::getOpenFileName(this, "Открыть_файл", "",
136         "JSON_(*.json)");
137     QFile file(fileName);
138     file.open(QIODevice::ReadOnly | QIODevice::Text);
139     QJsonDocument doc = QJsonDocument::fromJson(file.readAll());
140     QJsonObject object = doc.object();
141     on_actionClear_triggered();
142     for (auto key: object.keys()) {
143         Shape* shape = nullptr;
144         auto doc = object[key].toObject();
145         auto params = doc["params"].toObject();
146         auto scenePos = doc["scenePos"].toObject();
147         if (doc["className"] == "Pentagram") {
148             shape = new Pentagram(params["size"].toDouble());
149         } else if (doc["className"] == "PentagramText") {
150             shape = new PentagramText(params["string"].toString(),
151                 params["size"].toDouble());
152         } else if (doc["className"] == "Text") {
153             shape = new Text(params["string"].toString(),
154                 params["width"].toDouble());
155         } else if (doc["className"] == "AtanSegment") {
156             shape = new AtanSegment(params["precision"].toInt(),
157                 params["width"].toDouble(),
158                 params["height"].toDouble());
159         }
160         if (shape != nullptr) {
161             addItem(shape, key, doc["className"].toString());
162             shape->setPos(scenePos["x"].toDouble(), scenePos["y"].toDouble());
163         }
164     }
165
166 void MainWindow::on_actionNew_triggered()
167 {
168     MainWindow* w = new MainWindow();
169     w->setAttribute(Qt::WA_DeleteOnClose, true);
170     w->show();
171 }

```

Листинг 18. Код ../src/adddialog.h

```

1 #ifndef ADDDIALOG_H
2 #define ADDDIALOG_H
3
4 #include <QDialog>
5 #include <QGraphicsItem>
6

```

```

7  namespace Ui {
8  class AddDialog;
9  }
10
11 class AddDialog : public QDialog
12 {
13     Q_OBJECT
14
15 public:
16     explicit AddDialog(QString className, QGraphicsItem *item = nullptr,
17         QWidget *parent = nullptr);
18     ~AddDialog();
19
20 signals:
21     void itemChanged(QGraphicsItem* item);
22
23 private slots:
24     void on_selectColorButton_clicked();
25     void on_buttonBox_accepted();
26
27 private:
28     void addControls();
29     void addPentagramControls();
30     void addAtanSegmentControls();
31     void addTextControls();
32
33     [[nodiscard]] QGraphicsItem *makeItem();
34     void setItemValues();
35
36     Ui::AddDialog *ui;
37     QColor color = Qt::yellow;
38     QGraphicsItem* item;
39     QString className;
40 };
41
42 #endif // ADDDIALOG_H

```

Листинг 19. Код ../src/adddialog.cpp

```

1  #include <QColorDialog>
2  #include <QSpinBox>
3  #include <QSplitter>
4  #include <QLineEdit>
5  #include <tuple>
6
7  #include "adddialog.h"
8  #include "ui_adddialog.h"
9  #include "figures/atansegment.h"
10 #include "figures/pentagram.h"
11 #include "figures/pentagramtext.h"
12 #include "figures/text.h"
13
14 AddDialog::AddDialog(QString className, QGraphicsItem *item, QWidget *parent)
15     :QDialog(parent), ui(new Ui::AddDialog), item(item), className(className)
16 {
17     ui->setupUi(this);
18     addControls();
19 }
20
21 AddDialog::~AddDialog()
22 {
23     delete ui;
24 }
25
26 void AddDialog::on_selectColorButton_clicked()
27 {
28     color = QColorDialog::getColor(color, this, "Выберите_цвет");
29 }
30
31 void AddDialog::addControls()

```

```

32 {
33     if (className == "Pentagram") {
34         addPentagramControls();
35     } else if (className == "AtanSegment") {
36         addAtanSegmentControls();
37     } else if (className == "Text"){
38         addTextControls();
39     } else if (className == "PentagramText") {
40         addPentagramControls();
41         addTextControls();
42     }
43     if (item != nullptr) {
44         Shape* shape = dynamic_cast<Shape*>(item);
45         color = shape->getColor();
46         ui->showHashCodeCheckBox->setChecked(shape->getDrawHashKey());
47     }
48     auto splitter = new QSplitter(Qt::Vertical);
49     ui->controlsLayout->addWidget(splitter);
50 }
51
52 void AddDialog::addPentagramControls()
53 {
54     auto spinBox = new QDoubleSpinBox();
55     spinBox->setRange(1, 1000);
56     spinBox->setObjectName("sizeSpinBox");
57     ui->controlsLayout->addWidget(new QLabel("Размер"));
58     ui->controlsLayout->addWidget(spinBox);
59     if (item == nullptr){
60         spinBox->setValue(100);
61     } else {
62         Pentagon* p = dynamic_cast<Pentagon*>(item);
63         spinBox->setValue(p->getSize());
64     }
65 }
66
67 void AddDialog::addAtanSegmentControls()
68 {
69     auto spinBox = new QSpinBox();
70     spinBox->setRange(1, 1000);
71     spinBox->setObjectName("precisionSpinBox");
72
73     auto widthSpinBox = new QDoubleSpinBox();
74     widthSpinBox->setRange(1, 1000);
75     widthSpinBox->setObjectName("widthSpinBox");
76
77     auto heighthSpinBox = new QDoubleSpinBox();
78     heighthSpinBox->setRange(1, 1000);
79     heighthSpinBox->setObjectName("heightSpinBox");
80
81     ui->controlsLayout->addWidget(new QLabel("Точность"));
82     ui->controlsLayout->addWidget(spinBox);
83     ui->controlsLayout->addWidget(new QLabel("Высота"));
84     ui->controlsLayout->addWidget(heighthSpinBox);
85     ui->controlsLayout->addWidget(new QLabel("Ширина"));
86     ui->controlsLayout->addWidget(widthSpinBox);
87
88     if (item == nullptr) {
89         spinBox->setValue(100);
90         widthSpinBox->setValue(100);
91         heighthSpinBox->setValue(100);
92     } else {
93         AtanSegment* a = dynamic_cast<AtanSegment*>(item);
94         spinBox->setValue(a->getPrecision());
95         widthSpinBox->setValue(a->getWidth());
96         heighthSpinBox->setValue(a->getHeight());
97     }
98 }
99 }
100
101 void AddDialog::addTextControls()
102 {

```

```

103     auto textEdit = new QLineEdit();
104     textEdit->setObjectName("lineEdit");
105
106     auto textWidth = new QDoubleSpinBox();
107     textWidth->setObjectName("textWidthSpinBox");
108     textWidth->setRange(1, 1000);
109
110     ui->controlsLayout->addWidget(new QLabel("Текст"));
111     ui->controlsLayout->addWidget(textEdit);
112     ui->controlsLayout->addWidget(new QLabel("Ширина_текста"));
113     ui->controlsLayout->addWidget(textWidth);
114     if (item == nullptr) {
115         textWidth->setValue(40);
116     } else {
117         Text* t = dynamic_cast<Text*>(item);
118         textWidth->setValue(t->getWidth());
119         textEdit->setText(t->getString());
120     }
121 }
122
123 QGraphicsItem *AddDialog::makeItem()
124 {
125     if (className == "Pentagram"){
126         return new Pentagram();
127     } else if (className == "AtanSegment") {
128         return new AtanSegment();
129     } else if (className == "Text") {
130         return new Text();
131     } else {
132         return new PentagramText();
133     }
134 }
135
136 void AddDialog::setItemValues()
137 {
138     if (className == "Pentagram" || className == "PentagramText"){
139         auto size =
140             ui->controlBox->findChild<QDoubleSpinBox*>("sizeSpinBox")->value();
141         Pentagram* p = dynamic_cast<Pentagram*>(item);
142         p->setSize(size);
143     }
144     if (className == "AtanSegment") {
145         auto precision =
146             ui->controlBox->findChild<QSpinBox*>("precisionSpinBox")->value();
147         auto width =
148             ui->controlBox->findChild<QDoubleSpinBox*>("widthSpinBox")->value();
149         auto height =
150             ui->controlBox->findChild<QDoubleSpinBox*>("heightSpinBox")->value();
151         AtanSegment* a = dynamic_cast<AtanSegment*>(item);
152         a->setPrecision(precision);
153         a->setWidth(width);
154         a->setHeight(height);
155     }
156     if (className == "Text" || className == "PentagramText") {
157         auto width =
158             ui->controlBox->findChild<QDoubleSpinBox*>("textWidthSpinBox")->value();
159         auto text = ui->controlBox->findChild<QLineEdit*>("lineEdit")->text();
160         Text* t = dynamic_cast<Text*>(item);
161         t->setString(text);
162         t->setWidth(width);
163     }
164     Shape* s = dynamic_cast<Shape*>(item);
165     s->setColor(color);
166     s->setDrawHashKey(ui->showHashCodeCheckBox->isChecked());
167 }
168
169 void AddDialog::on_buttonBox_accepted()
170 {
171     if (item == nullptr) {
172         item = makeItem();
173     }
174 }

```

```
169     setItemValues();
170     emit itemChanged(item);
171     close();
172 }
```


ПРИЛОЖЕНИЕ Г

Пример сериализации

Листинг 20. Код ./example.json

```
1  {
2    "123": {
3      "className": "Pentagram",
4      "params": {
5        "size": 67
6      },
7      "scenePos": {
8        "x": -121.25,
9        "y": -93.75
10     }
11   },
12   "345": {
13     "className": "AtanSegment",
14     "params": {
15       "height": 203,
16       "precision": 100,
17       "width": 100
18     },
19     "scenePos": {
20       "x": -116.25,
21       "y": 85
22     }
23   },
24   "Hello,hello": {
25     "className": "Text",
26     "params": {
27       "string": "Hello,hello",
28       "width": 176
29     },
30     "scenePos": {
31       "x": -4.972010294170364,
32       "y": 25.306874964876506
33     }
34   },
35   "ff": {
36     "className": "PentagramText",
37     "params": {
38       "size": 100,
39       "string": "Text",
40       "width": 40
41     },
42     "scenePos": {
43       "x": 80.82771161861068,
44       "y": -89.80856846512297
45     }
46   }
47 }
```