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 _____ Корытов П.В.
Преподаватель _____ Спицин А.В.

Санкт-Петербург
2019

СОДЕРЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

1.1. Цель работы

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.2. Формулировка задания

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.3. Индивидуальное задание

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

2. ХОД РАБОТЫ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3. ВЫВОДЫ

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

ПРИЛОЖЕНИЕ А

Исходный код 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”,
                    e.toString());
17              return true;
18          }
19          return false;
20      }
21      void setMessageBoxWidget(QWidget *value) {
22          messageBoxWidget = value;
23      }
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  }
```

ПРИЛОЖЕНИЕ Б

Исходный код mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "graphwidget.h"
#include "hashMap.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = nullptr);

    ~MainWindow();

private slots:
    void on_addFigureButton_clicked();
    void on_itemAdded(QGraphicsItem* item);
    void addItem(QGraphicsItem* item, QString hash, QString className);
    void on_itemDelete(QString hash);
    void on_itemEdit(QString hash);
    void on_itemEdited(QGraphicsItem* item);

    void on_actionSave_triggered();

    void on_actionClear_triggered();

    void on_actionOpen_triggered();

    void on_actionNew_triggered();

private:
    HashMap<QString, QGraphicsItem*> hashMap;
    Ui::MainWindow *ui;
    GraphWidget* widget;
};

#endif // MAINWINDOW_H
```

ПРИЛОЖЕНИЕ В

Исходный код mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "adddialog.h"
#include "figures/shape.h"
#include "figures/pentagram.h"
#include "figures/atansegment.h"
#include "figures/pentagramtext.h"
#include "figures/text.h"

#include <QFileDialog>
#include <QJsonDocument>
#include <QTableWidgetItem>
#include <QMessageBox>


uint hashFunc(QString string) {
    return qHash(string) % SIZE;
}

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent), hashMap(hashFunc), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    widget = new GraphWidget(this);
    setCentralWidget(widget);
}

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

void MainWindow::on_addFigureButton_clicked()
{
    auto dialog = new AddDialog(ui->classComboBox->currentText(), nullptr,
        this);
    connect(dialog, &AddDialog::itemChanged, this, &MainWindow::on_itemAdded);
    dialog->setAttribute(Qt::WA_DeleteOnClose, true);
    dialog->show();
}

void MainWindow::on_itemAdded(QGraphicsItem *item)
{
    auto hash = ui->hashCodeEdit->text();
    addItem(item, hash, ui->classComboBox->currentText());
}

void MainWindow::addItem(QGraphicsItem *item, QString hash, QString className)
{
    hashMap.create(hash, item);

    Shape* shape = dynamic_cast<Shape*>(item);
    shape->setHashKey(hash);
    widget->scene()->addItem(shape);
    shape->setPos(widget->centerPos());
    shape->update();

    auto newRow = ui->tableWidget->rowCount();
    ui->tableWidget->insertRow(newRow);
    ui->tableWidget->setItem(newRow, 0, new QTableWidgetItem(hash));
    ui->tableWidget->setItem(newRow, 1, new QTableWidgetItem(className));
    ui->tableWidget->setItem(newRow, 2, new
        QTableWidgetItem(shape->toString()));

    QWidget* controlWidget = new QWidget();
    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, [=](){
            this->on_itemDelete(hash); });
75      connect(editButton, &QPushButton::clicked, this, [=](){
            this->on_itemEdit(hash); });
76      ui->hashCodeEdit->clear();
77      widget->scene()->update();
78  }
79
80  void MainWindow::on_itemDelete(QString hash)
81  {
82      QGraphicsItem* item = hashMap.at(hash);
83
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
94  void MainWindow::on_itemEdit(QString hash)
95  {
96      QGraphicsItem* item = hashMap.at(hash);
97
98      auto dialog = new AddDialog(ui->classComboBox->currentText(), item, this);
99      connect(dialog, &AddDialog::itemChanged, this, &MainWindow::on_itemEdited);
100     dialog->setAttribute(Qt::WA_DeleteOnClose, true);
101     dialog->show();
102 }
103
104 void MainWindow::on_itemEdited(QGraphicsItem *item)
105 {
106     item->update();
107     widget->scene()->update();
108 }
109
110 void MainWindow::on_actionSave_triggered()
111 {
112     QJsonObject object;
113     for (auto it = hashMap.begin(); it <= hashMap.end(); it++){
114         Shape* shape = dynamic_cast<Shape*>(it.value());
115         object[it.key()] = shape->toJSON();
116     }
117     QJsonDocument doc(object);
118
119     QString fileName = QFileDialog::getSaveFileName(this, "Сохранить",
            "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", "",
             "JSON (*.json)");
136      QFile file(fileName);
137      file.open(QIODevice::ReadOnly | QIODevice::Text);
138      QJsonDocument doc = QJsonDocument::fromJson(file.readAll());
139      QJsonObject object = doc.object();
140
141      on_actionClear_triggered();
142
143      for (auto key: object.keys()) {
144          Shape* shape = nullptr;
145          auto doc = object[key].toObject();
146          auto params = doc["params"].toObject();
147          auto scenePos = doc["scenePos"].toObject();
148          if (doc["className"] == "Pentagram") {
149              shape = new Pentagram(params["size"].toDouble());
150          } else if (doc["className"] == "PentagramText") {
151              shape = new PentagramText(params["string"].toString(),
                     params["size"].toDouble());
152          } else if (doc["className"] == "Text") {
153              shape = new Text(params["string"].toString(),
                     params["width"].toDouble());
154          } else if (doc["className"] == "AtanSegment") {
155              shape = new AtanSegment(params["precision"].toInt(),
156                      params["width"].toDouble(),
157                      params["height"].toDouble());
158          }
159          if (shape != nullptr) {
160              addItem(shape, key, doc["className"].toString());
161              shape->setPos(scenePos["x"].toDouble(), scenePos["y"].toDouble());
162          }
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  }
```

ПРИЛОЖЕНИЕ Г

Исходный код hashMap.h

```
#pragma once

#include ”exception.h”
#include <algorithm>
#include <iostream>
#include <functional>
#include <string>

#define SIZE 2000000

template<typename Key>
unsigned int defaultHash(Key key) {
    return static_cast<unsigned int>(key) % SIZE;
}

template<typename Key>
bool defaultCompare(const Key& v1, const Key& v2) {
    return  v1 == v2;
}

template <typename Key, typename Value>
class HashMap;

template <typename Key, typename Value>
class HashNode {
public:
    explicit HashNode(const Key& key, const Value& value): key(key),
       value(value) {}

    HashNode (HashNode& node): key(node.key), value(node.value) {}

    friend void swap(HashNode& a, HashNode& b){
        std::swap(a.key, b.key);
        std::swap(a.value, b.value);
    }

    HashNode& operator=(const HashNode& other) {
        swap(*this, other);
        return *this;
    }

    HashNode& operator=(HashNode&& other) noexcept {
        swap(*this, other);
    }

    [[nodiscard]] const Value& getValue() {
        return value;
    }

    void setValue(Value newValue) {
        value = newValue;
    }

    [[nodiscard]] const Key& getKey() {
        return key;
    }

    [[nodiscard]] HashNode *getNext() const {
        return next;
    }

    void setNext(HashNode *newNext) {
        next = newNext;
    }

private:
    Key key;
    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),
                    compareFunc(hashMap.compareFunc) {
77          if (node == nullptr) {
78              getNext();
79          } else {
80              currentValue = hashMap.hashFunc(node->getKey());
81          }
82      }
83
84      HashMapIterator(HashMapIterator& it)
85              : HashMapIterator(it.hashMap, it.currentNode)
86      {
87          compareFunc = it.compareFunc;
88      }
89
90      HashMapIterator&operator++() {
91          getNext();
92          return *this;
93      }
94
95      HashMapIterator operator++(int) {
96          auto ret = HashMapIterator(*this);
97          getNext();
98          return ret;
99      }
100
101     const Key& key() {
102         return currentNode->getKey();
103     }
104
105     const Value& value() {
106         return currentNode->getValue();
107     }
108
109     bool end(){
110         return currentValue >= SIZE - 1;
111     }
112
113     friend bool operator==(const HashMapIterator<Key, Value>& it1,
114                            const HashMapIterator<Key, Value>& it2) {
115         return it1.compareFunc(it1.currentNode->getKey(),
                    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(),
                    it2.currentNode->getKey());
121     }
122
123     friend bool operator<(const HashMapIterator<Key, Value>& it1,
124                           const HashMapIterator<Key, Value>& it2) {
125         return it1.currentValue < it2.currentValue;
126     }
127
128     friend bool operator<=(const HashMapIterator<Key, Value>& it1,
129                            const HashMapIterator<Key, Value>& it2) {
130         if (it1.currentNode == nullptr && it2.currentNode == nullptr) {
131             return false;
132         }
133         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,
                      entry->getValue());
221           }
222       }
223
224       void update(const Key& key, const Value& value) {
225           auto [prev, entry, hashValue] = getEntry(key);
226
227           if (entry == nullptr) {
228               entry = new HashNode<Key, Value>(key, value);
229               if (prev == nullptr) {
230                   table[hashValue] = entry;
231               } else {
232                   prev->setNext(entry);
233               }
234           } else {
235               entry->setValue(value);
236           }
237       }
238
239       void remove(const Key& key) {
240           auto [prev, entry, hashValue] = getEntry(key);
241
242           if (entry == nullptr) {
243               throw ElementNotFoundException<Key>(key);
244           } else {
245               if (prev == nullptr) {
246                   table[hashValue] = entry->getNext();
247               } else {
248                   prev->setNext(entry->getNext());
249               }
250               delete entry;
251           }
252       }
253
254       HashMapIterator<Key, Value> begin(){
255           return HashMapIterator<Key, Value>(*this);
256       }
257
258       HashMapIterator<Key, Value> end(){
259           unsigned int lastHash;
260           for (lastHash = SIZE - 1; table[lastHash] == nullptr && lastHash > 0;
                  lastHash--);
261
262           if (lastHash > 0) {
263               HashNode<Key, Value>* node = table[lastHash];
264               while (node->getNext() != nullptr) {
265                   node = node->getNext();
266               }
267               return HashMapIterator<Key, Value>(*this, node);
268           }
269           return HashMapIterator<Key, Value>(*this);
270       }
271
272   private:
273       std::tuple<HashNode<Key, Value>*, HashNode<Key, Value>*,
274               unsigned int>getEntry(const Key& key) {
275           unsigned int hashValue = hashFunc(key);
276
277           HashNode<Key, Value> *entry = table[hashValue];
278           HashNode<Key, Value>* prev = nullptr;
```

```
279             while (entry != nullptr && !compareFunc(entry->getKey(), key)) {
280                 prev = entry;
281                 entry = entry->getNext();
282             }
283             return std::make_tuple(prev, entry, hashValue);
284         }
285 
286         HashNode<Key, Value> **table;
287         std::function<unsigned int(Key)> hashFunc;
288         std::function<int(Key, Key)> compareFunc;
289     };
```

ПРИЛОЖЕНИЕ Е

Исходный код 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().toStdString();
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),
        value(element) {}
38
39 protected:
40     QString toString() const override {
41         QString string;
42         QTextStream outStream(&string);
43         outStream << "ElementAlreadyExistsException␣{Key:␣" << key << ",␣
            Value:" << value << "}";
44         return string;
45     }
46
47 private:
48     Key key;
49     Value value;
50 };
```

ПРИЛОЖЕНИЕ Ж

Исходный код figures/shape.h

```
#ifndef SHAPE_H
#define SHAPE_H

#include ”graphwidget.h”
#include ”point.h”
#include <QGraphicsItem>
#include <QJsonObject>

class Shape : virtual public QGraphicsItem
{
public:
    explicit Shape();
    virtual ~Shape() override = default;

    // QGraphicsItem interface
    virtual QRectF boundingRect() const override = 0;
    virtual QPainterPath shape() const override = 0;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
        *option, QWidget *widget) override;

    friend std::ostream &operator<<(std::ostream &os, const Shape &shape);

    [[nodiscard]] virtual QString toString();
    [[nodiscard]] virtual QJsonObject toJSON();
    [[nodiscard]] QString getHashKey() const;
    void setHashKey(const QString &value);

    [[nodiscard]] bool getDrawHashKey() const;
    void setDrawHashKey(bool value);

    void setColor(const QColor &value);
    QColor getColor() const;

protected:
    virtual QColor primaryColor(const QStyleOptionGraphicsItem* option);
    virtual void print(std::ostream &o) const = 0;

    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override;

    static QPolygonF getPolygon(QVector<Point> points);
    static void mergeJsons(QJsonObject& doc, const QJsonObject& second);

private:
    QColor color = Qt::yellow;
    QString hashKey;
    bool drawHashKey = true;
};

inline std::ostream &operator<<(std::ostream &os, const Shape &shape) {
    shape.print(os);
    return os;
}

#endif // SHAPE_H
```

ПРИЛОЖЕНИЕ 3

Исходный код figures/shape.cpp

```
#include "shape.h"

#include <QStyleOptionGraphicsItem>
#include <QJsonObject>
#include <QJsonDocument>

Shape::Shape()
{
    setFlag(ItemIsMovable);
    setFlag(ItemSendsGeometryChanges);
    setCacheMode(DeviceCoordinateCache);
    setZValue(-1);
}

void Shape::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
    QWidget *widget)
{
    if (drawHashKey) {
        auto width = painter->fontMetrics().horizontalAdvance(hashKey);
        auto height = painter->fontMetrics().height();
        auto size = QSizeF(-width, -height);
        auto rect = QRectF(boundingRect().bottomRight(), size);
        rect = rect.normalized();

        painter->drawText(rect, Qt::AlignRight | Qt::AlignBottom, hashKey);
    }
}

QString Shape::toString()
{
    QJsonDocument doc(toJSON());
    return QString(doc.toJson(QJsonDocument::Compact));
}

QJsonObject Shape::toJSON()
{
    return QJsonObject {
        {"scenePos", QJsonObject {
                {"x", scenePos().x()},
                {"y", scenePos().y()},
            }
        }
    };
}

QColor Shape::primaryColor(const QStyleOptionGraphicsItem *option)
{
    QColor col = QColor(color);
    if (option->state & QStyle::State_Sunken) {
        col = col.darker(120);
    }
    return col;
}

void Shape::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
    update();
    QGraphicsItem::mousePressEvent(event);
}

void Shape::mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
{
    update();
    QGraphicsItem::mouseReleaseEvent(event);
}

QPolygonF Shape::getPolygon(QVector<Point> points)
```

```
 67  {
 68      auto polygon = QPolygonF();
 69      for (auto point: points){
 70          polygon << point;
 71      }
 72      return polygon;
 73  }
 74
 75  void Shape::mergeJsons(QJsonObject &doc, const QJsonObject &second)
 76  {
 77      for (auto key: second.keys()) {
 78          doc[key] = second[key];
 79      }
 80  }
 81
 82  QColor Shape::getColor() const
 83  {
 84      return color;
 85  }
 86
 87  void Shape::setColor(const QColor &value)
 88  {
 89      color = value;
 90  }
 91
 92  bool Shape::getDrawHashKey() const
 93  {
 94      return drawHashKey;
 95  }
 96
 97  void Shape::setDrawHashKey(bool value)
 98  {
 99      drawHashKey = value;
100  }
101
102  QString Shape::getHashKey() const
103  {
104      return hashKey;
105  }
106
107  void Shape::setHashKey(const QString &value)
108  {
109      hashKey = value;
110      drawHashKey = true;
111  }
```

ПРИЛОЖЕНИЕ И

Исходный код 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
36     uint precision;
37     double width;
38     double height;
39
40     double atanStart = -1;
41     double atanStop = 1;
42     double lineWidth = 1;
43 };
44
45 #endif // ATANSEGMENT_H
```

ПРИЛОЖЕНИЕ К

Исходный код figures/atansegment.cpp

```
#include "atansegment.h"
#include <QJsonObject>
#include <QtMath>
#include <QStyleOptionGraphicsItem>
#include <cmath>


AtanSegment::AtanSegment(uint precision, double width, double height)
    :precision(precision), width(width), height(height)
{

}

QRectF AtanSegment::boundingRect() const
{
    qreal adjust = 2;
    return QRectF(-width / 2 - adjust, -height / 2 - adjust,
                  width + adjust, height + adjust);
}

QPainterPath AtanSegment::shape() const
{
    QPainterPath path;
    path.addRect(boundingRect());
    return path;
}

void AtanSegment::paint(QPainter *painter, const QStyleOptionGraphicsItem
    *option, QWidget *widget)
{
    Shape::paint(painter, option, widget);
    auto points = getPath();
    QBrush brush(primaryColor(option));
    QPen pen(brush, lineWidth);
    painter->setPen(pen);
    painter->setBrush(brush);
    for (int i = 0; i < points.length() - 1; i++) {
        painter->drawLine(points[i], points[i+1]);
    }
    // painter->drawRect(boundingRect());
}

QJsonObject AtanSegment::toJSON()
{
    QJsonObject object {
        {"className", "AtanSegment"},
        {"params", QJsonObject {
                {"precision", (int)precision},
                {"width", width},
                {"height", height}
            }
        }
    };
    mergeJsons(object, Shape::toJSON());
    return object;
}

void AtanSegment::print(std::ostream &o) const
{
    o << "AtanSegment"; // TODO
}

QVector<Point> AtanSegment::getPath() const
{
    auto path = QVector<Point>();
    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 }
```

ПРИЛОЖЕНИЕ Л

Исходный код 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      Pentagram(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
```

ПРИЛОЖЕНИЕ М

Исходный код figures/pentagram.cpp

```
 1  #include "pentagram.h"
 2  #include <cmath>
 3  #include <QJsonObject>
 4  #include <QPainter>
 5  #include <QStyleOption>
 6  #include <QPolygonF>
 7
 8  Pentagram::Pentagram(double size)
 9      :size(size)
10  {
11
12  }
13
14  QRectF Pentagram::boundingRect() const
15  {
16      qreal adjust = 2;
17      return QRectF(-size - adjust, - size - adjust,
18                    size * 2 + adjust, size * 2 + adjust);
19  }
20
21  QPainterPath Pentagram::shape() const
22  {
23      QPainterPath path;
24      path.addPolygon(getPolygon(getPoints()));
25      return path;
26  }
27
28  void Pentagram::paint(QPainter *painter, const QStyleOptionGraphicsItem
        *option, QWidget *widget)
29  {
30      Shape::paint(painter, option, widget);
31      painter->setPen(QPen(Qt::black));
32      painter->setBrush(primaryColor(option));
33      auto path = getPath();
34      auto polygon = getPolygon(path);
35      painter->drawPolygon(polygon, Qt::WindingFill);
36  }
37
38  void Pentagram::print(std::ostream &o) const
39  {
40      o << "Pentagram";
41  }
42
43  double Pentagram::getSize() const
44  {
45      return size;
46  }
47
48  void Pentagram::setSize(double value)
49  {
50      size = value;
51  }
52
53  QJsonObject Pentagram::toJSON()
54  {
55      QJsonObject object {
56          {"className", "Pentagram"},
57          {"params", QJsonObject {
58                  {"size", size}
59              }
60          }
61      };
62      mergeJsons(object, Shape::toJSON());
63      return object;
64  }
65
66  QVector<Point> Pentagram::getPoints() const
```

```
67  {
68      auto points = QVector<Point>();
69      for (int i = 0; i < 5; i++) {
70          auto p = Point {size, 0};
71          p.setPhi(i * M_PI * 2 / 5);
72          points.push_back(p);
73      }
74      return points;
75  }
76
77  QVector<Point> Pentagram::getPath() const
78  {
79      auto points = getPoints();
80      return QVector<Point> {
81          points[0],
82          points[2],
83          points[4],
84          points[1],
85          points[3],
86          points[0]
87      };
88  }
```

ПРИЛОЖЕНИЕ Н

Исходный код 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
```

ПРИЛОЖЕНИЕ О

Исходный код figures/text.cpp

```
#include "text.h"
#include <QStyleOptionGraphicsItem>
#include <QJsonObject>

Text::Text(QString string, double width)
    :string(string), width(width)
{
    height = width / string.length() * 8;
}

QRectF Text::boundingRect() const
{
    qreal adjust = 2;
    return QRectF(-adjust - width / 2, -adjust - height / 2,
                  width + adjust, height + adjust);
}

QPainterPath Text::shape() const
{
    QPainterPath path;
    path.addRect(boundingRect());
    return path;
}

void Text::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
    QWidget *widget)
{
    Shape::paint(painter, option, widget);
    auto factor = width / painter->fontMetrics().horizontalAdvance(string);
    auto font = painter->font();
    if (string.length() == 1) {
        factor *= 0.5;
    }
    font.setPointSizeF(font.pointSizeF() * factor);
    height = painter->fontMetrics().height() * font.pointSizeF() / 5;
    QRectF rect(-width / 2, -height / 4, width, height / 2);

    painter->setFont(font);
    painter->setPen(QPen(textColor(option)));
    painter->drawText(rect, Qt::AlignHCenter | Qt::AlignVCenter, string);
    // painter->drawRect(rect);
}

void Text::print(std::ostream &o) const
{
    o << "Text"; // TODO
}

double Text::getHeight() const
{
    return height;
}

void Text::setHeight(double value)
{
    height = value;
}

QJsonObject Text::toJSON()
{
    QJsonObject object {
        {"className", "Text"},
        {"params", QJsonObject {
                {"string", string},
                {"width", width}
            }
        }
```

```
67      };
68      mergeJsons(object, Shape::toJSON());
69      return object;
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  }
```

ПРИЛОЖЕНИЕ П

Исходный код 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
```

ПРИЛОЖЕНИЕ Р

Исходный код figures/pentagramtext.cpp

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

ПРИЛОЖЕНИЕ С

Исходный код graphwidget.h

```
#ifndef GRAPHWIDGET_H
#define GRAPHWIDGET_H

#include <QGraphicsView>

class GraphWidget : public QGraphicsView
{
    Q_OBJECT
public:
    GraphWidget(QWidget* parent = nullptr);
    [[nodiscard]] QPointF centerPos();
    void setSceneSize();

public slots:
    void zoomIn();
    void zoomOut();

protected:
    void keyPressEvent(QKeyEvent *event) override;
    void wheelEvent(QWheelEvent *event) override;
    void drawBackground(QPainter *painter, const QRectF &rect) override;
    void scaleView(qreal scaleFactor);

    // QWidget interface
    void resizeEvent(QResizeEvent *event) override;
};

#endif // GRAPHWIDGET_H
```

ПРИЛОЖЕНИЕ Т

Исходный код graphwidget.cpp

```
 1  #include "graphwidget.h"
 2  #include <QWheelEvent>
 3
 4  #include <iostream>
 5  #include <math.h>
 6  #include "figures/pentagram.h"
 7  #include "figures/atansegment.h"
 8  #include "figures/text.h"
 9  #include "figures/pentagramtext.h"
10
11  GraphWidget::GraphWidget(QWidget* parent)
12      :QGraphicsView (parent)
13  {
14      QGraphicsScene *scene = new QGraphicsScene(this);
15      scene->setItemIndexMethod(QGraphicsScene::NoIndex);
16      scene->setSceneRect(-200, -200, 400, 400);
17      setScene(scene);
18      setCacheMode(CacheBackground);
19      setViewportUpdateMode(BoundingRectViewportUpdate);
20      setRenderHint(QPainter::Antialiasing);
21      setTransformationAnchor(AnchorUnderMouse);
22      scale(qreal(0.8), qreal(0.8));
23      setMinimumSize(400, 400);
24
25      setSceneSize();
26  }
27
28  QPointF GraphWidget::centerPos()
29  {
30      auto size = this->size();
31      auto widgetCenter = QPoint(size.width() / 2, size.height() / 2);
32      return this->mapToScene(widgetCenter);
33  }
34
35  void GraphWidget::setSceneSize()
36  {
37      auto size = this->size();
38      scene()->setSceneRect(-size.width() / 2, - size.height() / 2,
            size.width(), size.height());
39  }
40
41  void GraphWidget::keyPressEvent(QKeyEvent *event)
42  {
43      std::cout << "Key pressed" << std::endl;
44  }
45
46  void GraphWidget::wheelEvent(QWheelEvent *event)
47  {
48      scaleView(pow((double)2, -event->delta() / 240.0));
49  }
50
51  void GraphWidget::scaleView(qreal scaleFactor)
52  {
53      qreal factor = transform().scale(scaleFactor,
            scaleFactor).mapRect(QRectF(0, 0, 1, 1)).width();
54      if (factor < 0.07 || factor > 100)
55          return;
56
57      scale(scaleFactor, scaleFactor);
58  }
59
60  void GraphWidget::resizeEvent(QResizeEvent *event)
61  {
62      setSceneSize();
63      QGraphicsView::resizeEvent(event);
64  }
65
```

```
 66  void GraphWidget::zoomIn()
 67  {
 68      scaleView(qreal(1.2));
 69  }
 70
 71  void GraphWidget::zoomOut()
 72  {
 73      scaleView(1 / qreal(1.2));
 74  }
 75
 76
 77  void GraphWidget::drawBackground(QPainter *painter, const QRectF &rect)
 78  {
 79      Q_UNUSED(rect)
 80
 81      // Shadow
 82      QRectF sceneRect = this->sceneRect();
 83      QRectF rightShadow(sceneRect.right(), sceneRect.top() + 5, 5,
             sceneRect.height());
 84      QRectF bottomShadow(sceneRect.left() + 5, sceneRect.bottom(),
             sceneRect.width(), 5);
 85      if (rightShadow.intersects(rect) || rightShadow.contains(rect))
 86          painter->fillRect(rightShadow, Qt::darkGray);
 87      if (bottomShadow.intersects(rect) || bottomShadow.contains(rect))
 88          painter->fillRect(bottomShadow, Qt::darkGray);
 89
 90      // Fill
 91      QLinearGradient gradient(sceneRect.topLeft(), sceneRect.bottomRight());
 92      gradient.setColorAt(0, Qt::white);
 93      gradient.setColorAt(1, Qt::lightGray);
 94      painter->fillRect(rect.intersected(sceneRect), gradient);
 95      painter->setBrush(Qt::NoBrush);
 96      painter->drawRect(sceneRect);
 97
 98      // Text
 99      QRectF textRect(sceneRect.left() + 4, sceneRect.top() + 4,
100                      sceneRect.width() - 4, sceneRect.height() - 4);
101      QString message(tr("Корытов␣Павел,␣6304"));
102
103      QFont font = painter->font();
104      font.setBold(true);
105      font.setPointSize(14);
106      painter->setFont(font);
107      painter->setPen(Qt::black);
108      painter->drawText(textRect, message);
109  }
```

ПРИЛОЖЕНИЕ У

Исходный код 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,
            QWidget *parent = nullptr);
17      ~AddDialog();
18
19  signals:
20      void itemChanged(QGraphicsItem* item);
21
22  private slots:
23      void on_selectColorButton_clicked();
24
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
```

ПРИЛОЖЕНИЕ Ф

Исходный код adddialog.cpp

```
#include <QColorDialog>
#include <QSpinBox>
#include <QSplitter>
#include <QLineEdit>
#include <tuple>

#include "adddialog.h"
#include "ui_adddialog.h"
#include "figures/atansegment.h"
#include "figures/pentagram.h"
#include "figures/pentagramtext.h"
#include "figures/text.h"

AddDialog::AddDialog(QString className, QGraphicsItem *item, QWidget *parent)
    :QDialog(parent), ui(new Ui::AddDialog), item(item),  className(className)
{
    ui->setupUi(this);
    addControls();
}

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

void AddDialog::on_selectColorButton_clicked()
{
    color = QColorDialog::getColor(color, this, "Выберите цвет");
}

void AddDialog::addControls()
{
    if (className == "Pentagram") {
        addPentagramControls();
    } else if (className == "AtanSegment") {
        addAtanSegmentControls();
    } else if (className == "Text"){
        addTextControls();
    } else if (className == "PentagramText") {
        addPentagramControls();
        addTextControls();
    }
    if (item != nullptr) {
        Shape* shape = dynamic_cast<Shape*>(item);
        color = shape->getColor();
        ui->showHashCodeCheckBox->setChecked(shape->getDrawHashKey());
    }
    auto splitter = new QSplitter(Qt::Vertical);
    ui->controlsLayout->addWidget(splitter);
}

void AddDialog::addPentagramControls()
{
    auto spinBox = new QDoubleSpinBox();
    spinBox->setRange(1, 1000);
    spinBox->setObjectName("sizeSpinBox");
    ui->controlsLayout->addWidget(new QLabel("Размер"));
    ui->controlsLayout->addWidget(spinBox);
    if (item == nullptr){
        spinBox->setValue(100);
    } else {
        Pentagram* p = dynamic_cast<Pentagram*>(item);
        spinBox->setValue(p->getSize());
    }
}
```

```
 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 heigthSpinBox = new QDoubleSpinBox();
 78      widthSpinBox->setRange(1, 1000);
 79      heigthSpinBox->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(heigthSpinBox);
 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          heigthSpinBox->setValue(100);
 92      } else {
 93          AtanSegment* a = dynamic_cast<AtanSegment*>(item);
 94          spinBox->setValue(a->getPrecision());
 95          widthSpinBox->setValue(a->getWidth());
 96          heigthSpinBox->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 =
                  ui->controlBox->findChild<QDoubleSpinBox*>("sizeSpinBox")->value();
140           Pentagram* p = dynamic_cast<Pentagram*>(item);
141           p->setSize(size);
142       }
143       if (className == "AtanSegment") {
144           auto precision =
                  ui->controlBox->findChild<QSpinBox*>("precisionSpinBox")->value();
145           auto width =
                  ui->controlBox->findChild<QDoubleSpinBox*>("widthSpinBox")->value();
146           auto height =
                  ui->controlBox->findChild<QDoubleSpinBox*>("heightSpinBox")->value();
147           AtanSegment* a = dynamic_cast<AtanSegment*>(item);
148           a->setPrecision(precision);
149           a->setWidth(width);
150           a->setHeight(height);
151       }
152       if (className == "Text" || className == "PentagramText") {
153           auto width =
                  ui->controlBox->findChild<QDoubleSpinBox*>("textWidthSpinBox")->value();
154           auto text = ui->controlBox->findChild<QLineEdit*>("lineEdit")->text();
155           Text* t = dynamic_cast<Text*>(item);
156           t->setString(text);
157           t->setWidth(width);
158       }
159       Shape* s = dynamic_cast<Shape*>(item);
160       s->setColor(color);
161       s->setDrawHashKey(ui->showHashCodeCheckBox->isChecked());
162   }
163
164   void AddDialog::on_buttonBox_accepted()
165   {
166       if (item == nullptr) {
167           item = makeItem();
168       }
169       setItemValues();
170       emit itemChanged(item);
171       close();
172   }
```

ПРИЛОЖЕНИЕ X

Исходный код 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
```

ПРИЛОЖЕНИЕ Ц

Исходный код 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  }
```