|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

**Отчет**

|  |  |
| --- | --- |
| **По домашней работе №** | 3 |

**Название:**

Программирование на С++ с использованием классов

**Дисциплина:** Объектно-ориентированное программирование

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22Б |  |  | И.А. Люляев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  |  |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

Часть 3.1. Композиция

Разработать и реализовать диаграмму классов для описанных объектов предметной области, используя механизм композиции. Протестировать все методы каждого класса. Все поля классов должны быть скрытыми (private) или защищенными (protected). Методы не должны содержать операций ввода/вывода, за исключением процедуры, единственной задачей которой является вывод информации об объекте на экран.

Объект – каюта. Поля: номер, класс обслуживания, количество мест в каюте. Методы: процедура инициализации, процедура вывода информации о каюте на экран и функции, возвращающие значения полей по запросу.

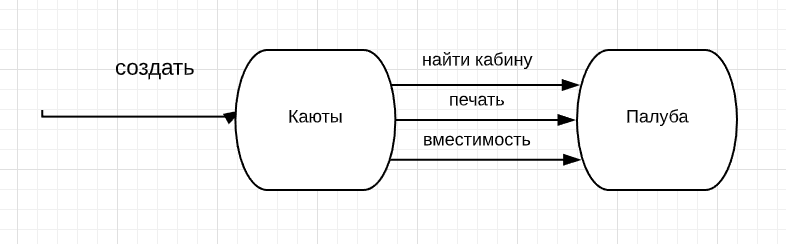
Объект – пассажирская палуба. Состоит из нескольких кают. Методы объекта должны позволять: инициализировать объект, выводить на экран содержимое объекта, определять количество пассажирских мест на палубе, находить номер каюты по параметрам (класс, количество человек).

В отчете привести диаграмму разработанных классов и объектную декомпозицию.

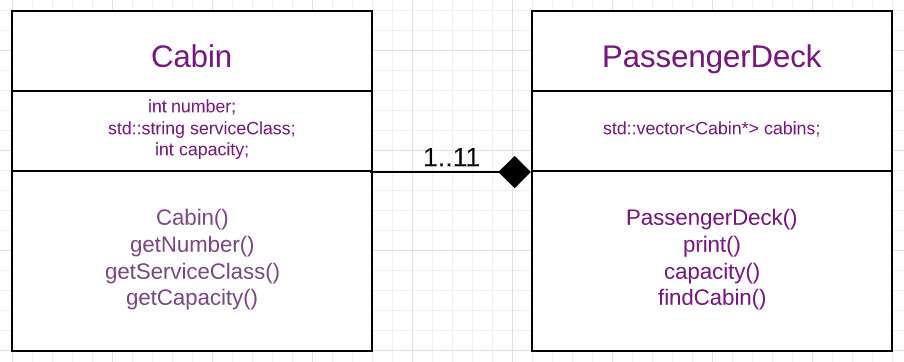
**Код программы:**

#include **<iostream>**#include **<vector>****class** Cabin {  
**public**:  
 Cabin(**int** number, **const** std::string &serviceClass, **int** capacity)  
 :number(number),  
 serviceClass(serviceClass),  
 capacity(capacity) {  
  
 }  
  
 Cabin() = **default**;  
  
 **void** print() {  
 std::cout << **"Number: "** << number << std::endl  
 << **"Service class: "** << serviceClass << std::endl  
 << **"Capacity: "** << capacity << std::endl;  
 }  
  
 **int** getNumber() **const** {  
 **return** number;  
 }  
  
 **const** std::string &getServiceClass() **const** {  
 **return** serviceClass;  
 }  
  
 **int** getCapacity() **const** {  
 **return** capacity;  
 }  
  
**private**:  
 **int** number;  
 std::string serviceClass;  
 **int** capacity;  
};  
  
**class** PassengerDeck {  
**public**:  
 PassengerDeck(**int** cabinsNumber, Cabin cabins[]) {  
 **for** (**int** i = 0; i < cabinsNumber; i++) {  
 **this**->cabins.push\_back(cabins[i]);  
 }  
 }  
  
 **void** print() {  
 **for** (**int** i = 0; i < cabins.size(); i++) {  
 cabins[i].print();  
 std::cout << std::endl;  
 }  
 }  
  
 **int** capacity() {  
 **int** deckCapacity = 0;  
 **for** (**int** i = 0; i < cabins.size(); i++) {  
 deckCapacity += cabins[i].getCapacity();  
 }  
 **return** deckCapacity;  
 }  
  
 **int** findCabin(**const** std::string &serviceClass, **int** capacity) {  
 **for** (**int** i = 0; i < cabins.size(); i++) {  
 **if** ((serviceClass == cabins[i].getServiceClass()) && (capacity == cabins[i].getCapacity())) {  
 **return** cabins[i].getNumber();  
 }  
 }  
 **return** -1;  
 }  
  
**private**:  
 std::vector<Cabin> cabins;  
};  
  
#define **NUMBER\_OF\_CABINS\_FIRSTCLASS** 2  
#define **NUMBER\_OF\_CABINS\_SECONDCLASS** 8  
#define **NUMBER\_OF\_ALL\_CABINS** (**NUMBER\_OF\_CABINS\_FIRSTCLASS** + **NUMBER\_OF\_CABINS\_SECONDCLASS**)  
  
  
**int** main() {  
  
 Cabin cabins[**NUMBER\_OF\_ALL\_CABINS** + 1];  
 **for** (**int** i = 0; i < **NUMBER\_OF\_CABINS\_FIRSTCLASS**; i++) {  
 Cabin cabin(i, **"first"**, 2);  
 cabins[i] = cabin;  
 }  
  
 **for** (**int** i = **NUMBER\_OF\_CABINS\_FIRSTCLASS**; i < **NUMBER\_OF\_ALL\_CABINS**; i++) {  
 Cabin cabin(i, **"second"**, 4);  
 cabins[i] = cabin;  
 }  
 Cabin cabin1(**NUMBER\_OF\_ALL\_CABINS**, **"second"**, 7);  
 cabins[**NUMBER\_OF\_ALL\_CABINS**] = cabin1;  
  
 PassengerDeck\* passengerDeck = **new** PassengerDeck(**NUMBER\_OF\_ALL\_CABINS**+1, cabins);  
 passengerDeck->print();  
 std::cout << **"Passeneger deck capacity: "** << passengerDeck->capacity() << std::endl;  
 std::cout << **"Cabin search result: "**<< passengerDeck->findCabin(**"second"**, 7);  
  
 **return** 0;  
}

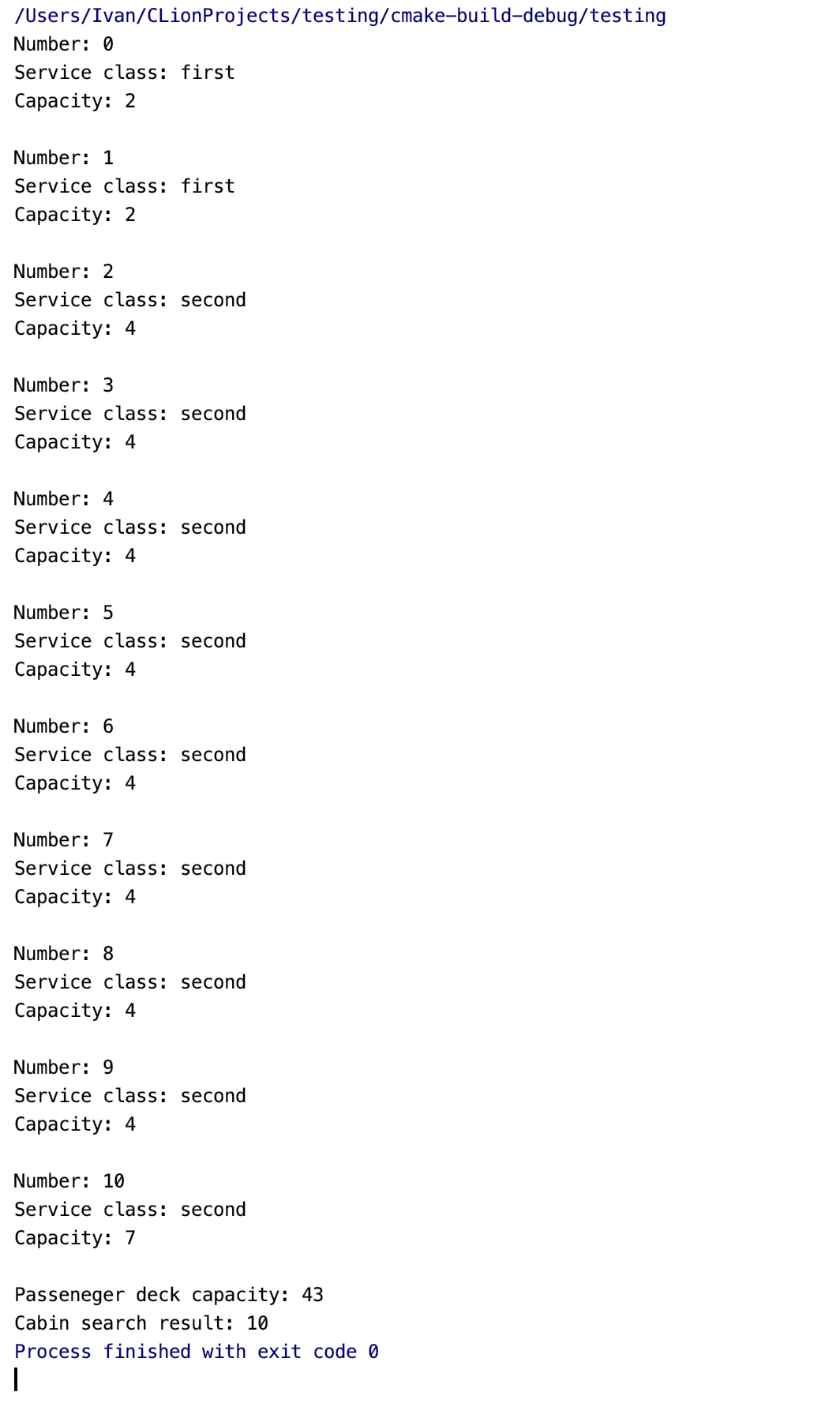
**Диаграмма объектов:**

****

**Диаграмма классов:**

****

**Результаты тестирования программы:**

****

**Вывод:** по результатам выполнения первой части домашнего задания я создал и реализовал классы, реализующие механизм композиции.

***Часть 3.2. Qt. Полиморфное наследование***

Разработать программу, содержащую описание трех графических объектов:

изображения букв: С, Ф, О.

Реализуя механизм полиморфизма, выполнять многократное масштабирование объектов в автоматическом режиме с различными коэффициентами.

В отчете привести диаграмму используемых классов Qt и разработанных классов, граф состояний пользовательского интерфейса и объектную декомпозицию.

**Код программы:**

**main.cpp**

#include "window.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

Window w;

w.show();

return a.exec();

}

**Figure.h**

**#**ifndef FIGURE\_H

#define FIGURE\_H

#include <QPainter>

class Figure

{

public:

Figure(QPainter\* painter, int r);

virtual void draw();

QPainter\* painter;

protected:

int x;

int y;

int r;

};

class Symbol\_S : public Figure {

public:

Symbol\_S(QPainter\* painter, int r);

void draw();

};

class Symbol\_F : public Figure {

public:

Symbol\_F(QPainter\* painter, int r);

void draw();

};

class Symbol\_O : public Figure {

public:

Symbol\_O(QPainter\* painter, int r);

void draw();

};

#endif // FIGURE\_H

**Figure.cpp**

#include "figure.h"

Figure::Figure(QPainter\* painter, int r)

{

this->painter = painter;

this->r = r;

}

void Figure::draw() {

this->painter->setPen(Qt::blue);

this->painter->setFont(QFont("Arial", r));

}

Symbol\_S::Symbol\_S(QPainter\* painter, int r) : Figure(painter, r) {

}

Symbol\_F::Symbol\_F(QPainter\* painter, int r) : Figure(painter, r) {}

Symbol\_O::Symbol\_O(QPainter\* painter, int r) : Figure(painter, r) {}

void Symbol\_S::draw(){

Figure::draw();

this->painter->drawText(200, 300, "C");}

void Symbol\_F::draw(){

Figure::draw();

this->painter->drawText(200, 300, "Ф");}

void Symbol\_O::draw(){

Figure::draw();

this->painter->drawText(200, 300, "О");}

**drawscreen.h**

**#**ifndef DRAWSCREEN\_H

#define DRAWSCREEN\_H

#include <QWidget>

#include <QString>

#include <QPainter>

#include <QDebug>

#include "figure.h"

class DrawScreen : public QWidget{

Q\_OBJECT

public:

explicit DrawScreen(QWidget \*parent = nullptr);

void setK(float k);

void setSymbol(QString symbol);

void scale();

Figure\* test;

float r; // font width

float k; // coefficient

QString symbol;

protected:

void paintEvent(QPaintEvent \*event); // override();

signals:

};

#endif // DRAWSCREEN\_H

**drawscreen.cpp**

**#**include "drawscreen.h"

DrawScreen::DrawScreen(QWidget \*parent) : QWidget(parent)

{

setFixedSize(QSize(500, 500));

r = 5;

k = 1.1;

symbol = "O";

}

void DrawScreen::setK(float k) {

this->k = k;

}

void DrawScreen::setSymbol(QString symbol) {

this->symbol = symbol;

}

void DrawScreen::scale()

{

r = r\*k;

if (r < 200) return;

else r = r/k;

}

void DrawScreen::paintEvent(QPaintEvent \*event) {

scale();

QPainter\* painter = new QPainter(this);

if (symbol == "С") test = new Symbol\_S(painter, r);

if (symbol == "Ф") test = new Symbol\_F(painter, r);

if (symbol == "O") test = new Symbol\_O(painter, r);

test->draw();

painter->end();

qDebug() << "update..";

}

**window.h**

**#**ifndef WINDOW\_H

#define WINDOW\_H

#include <QWidget>

#include <QPushButton>

#include <QLineEdit>

#include <QTextEdit>

#include <QTimer>

#include "drawscreen.h"

QT\_BEGIN\_NAMESPACE

namespace Ui { class Window; }

QT\_END\_NAMESPACE

class Window : public QWidget{

Q\_OBJECT

public:

Window(QWidget \*parent = nullptr);

~Window();

DrawScreen\* drawScreen;

QLineEdit\* symbol;

QLineEdit\* coefficient;

QTimer\* timer;

protected:

QPushButton\* btn;

public slots:

void changeScale();

void startScaling();

void symbolSet();

void kSet();

};

#endif // WINDOW\_H

**window.cpp**

#include "window.h"

#include "ui\_window.h"

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QString>

Window::Window(QWidget \*parent)

{

this->setWindowTitle("Symbols scaling");

drawScreen = new DrawScreen(this);

btn = new QPushButton("Start");

symbol = new QLineEdit("O");

coefficient = new QLineEdit("1.1");

QVBoxLayout\* controlLayout = new QVBoxLayout();

controlLayout->addWidget(btn);

controlLayout->addWidget(symbol);

controlLayout->addWidget(coefficient);

QHBoxLayout\* mainLayout = new QHBoxLayout(this);

mainLayout->addWidget(drawScreen);

mainLayout->addLayout(controlLayout);

connect(btn, SIGNAL(clicked()), this, SLOT(startScaling()));

connect(symbol, SIGNAL(editingFinished()), this, SLOT(symbolSet()));

connect(coefficient, SIGNAL(editingFinished()), this, SLOT(kSet()));

timer = new QTimer(this);

connect(timer, SIGNAL(timeout()), this, SLOT(changeScale()));}

Window::~Window(){}

void Window::changeScale() {

drawScreen->scale();

drawScreen->update();}

void Window::startScaling() {

drawScreen->r = 5;

timer->start(500);}

void Window::symbolSet() {

QString s = symbol->text();

drawScreen->setSymbol(s);}

void Window::kSet() {

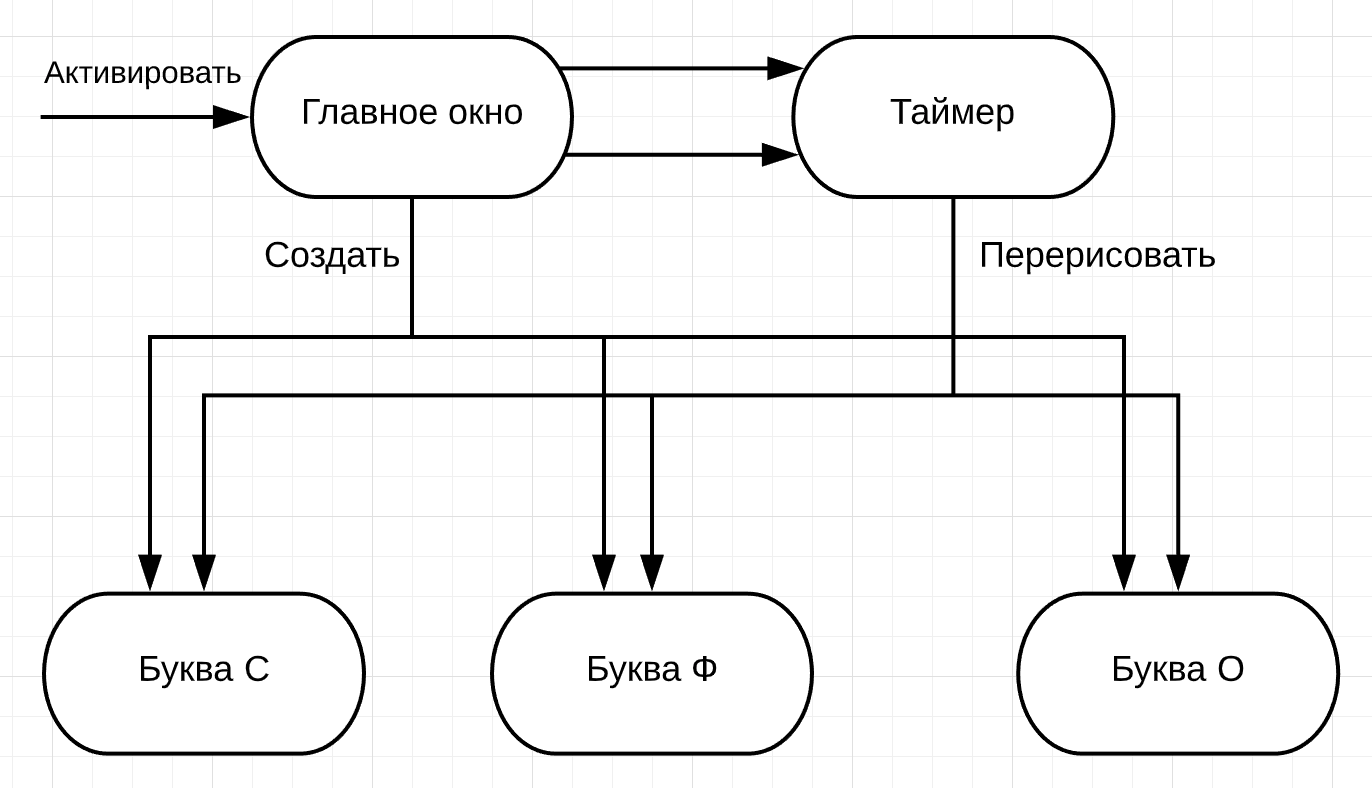
QString s = coefficient->text();

bool isFloat = false;

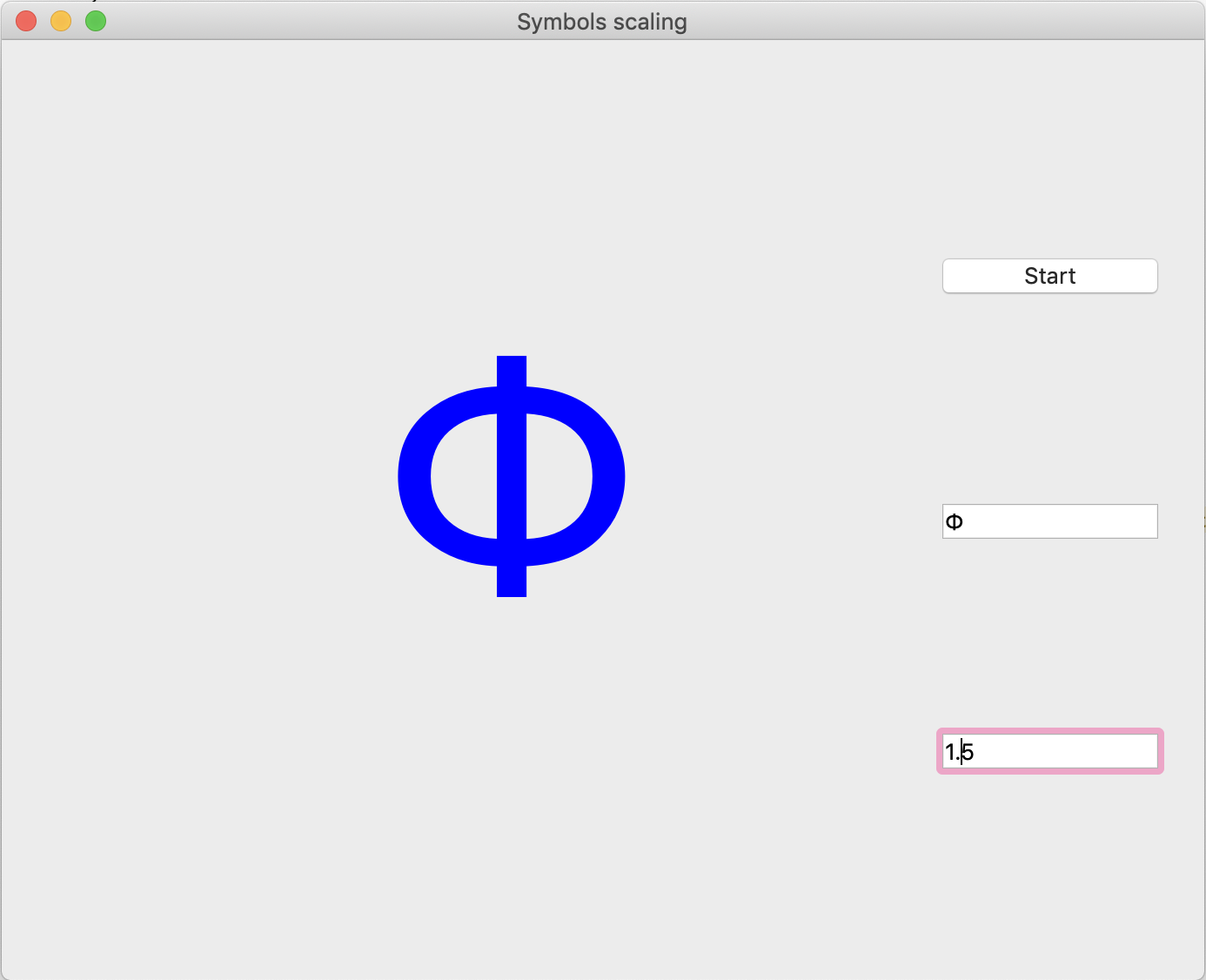
float k = s.toFloat(&isFloat);

if (isFloat) drawScreen->setK(k);}

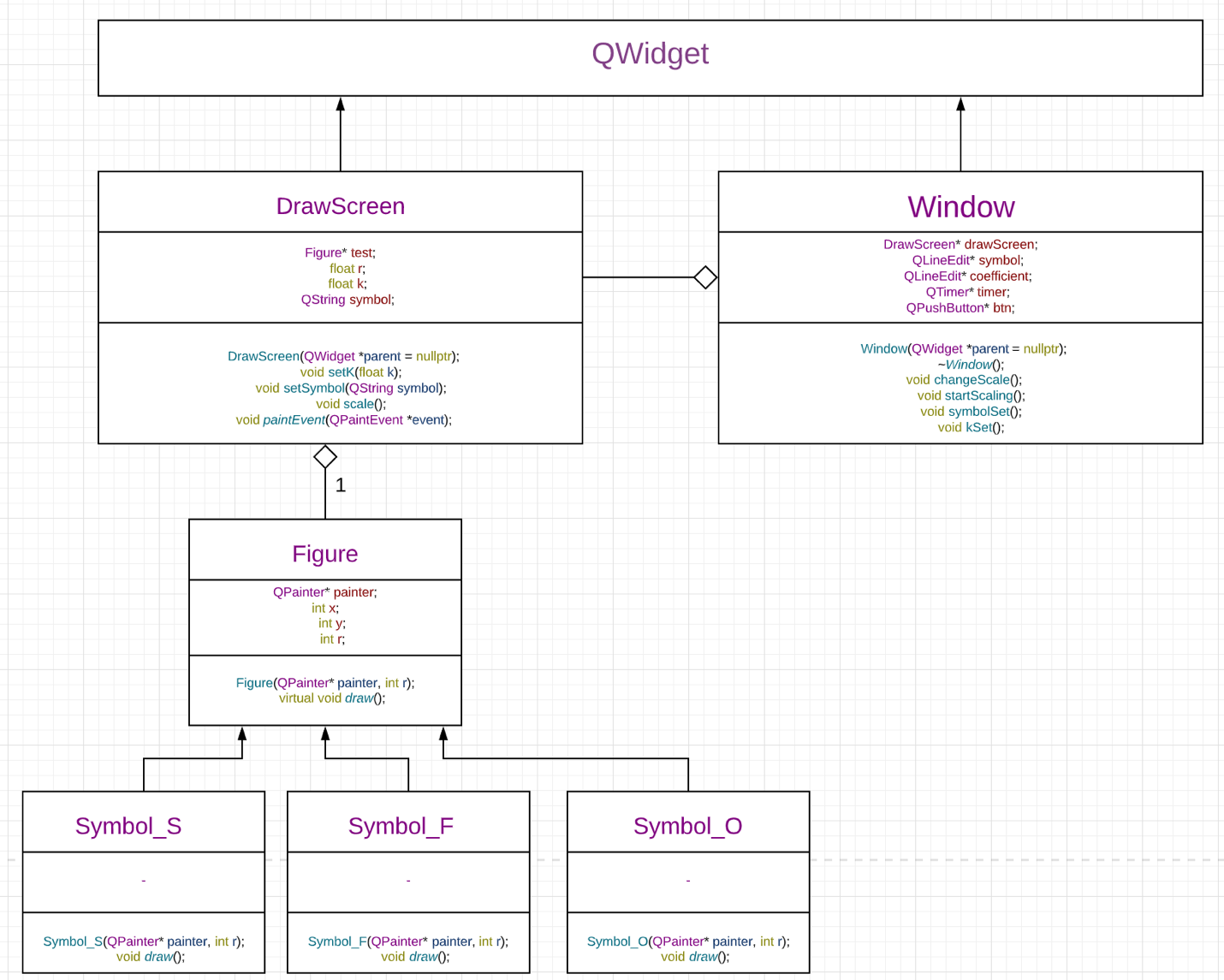
**Диаграмма объектов:**

****

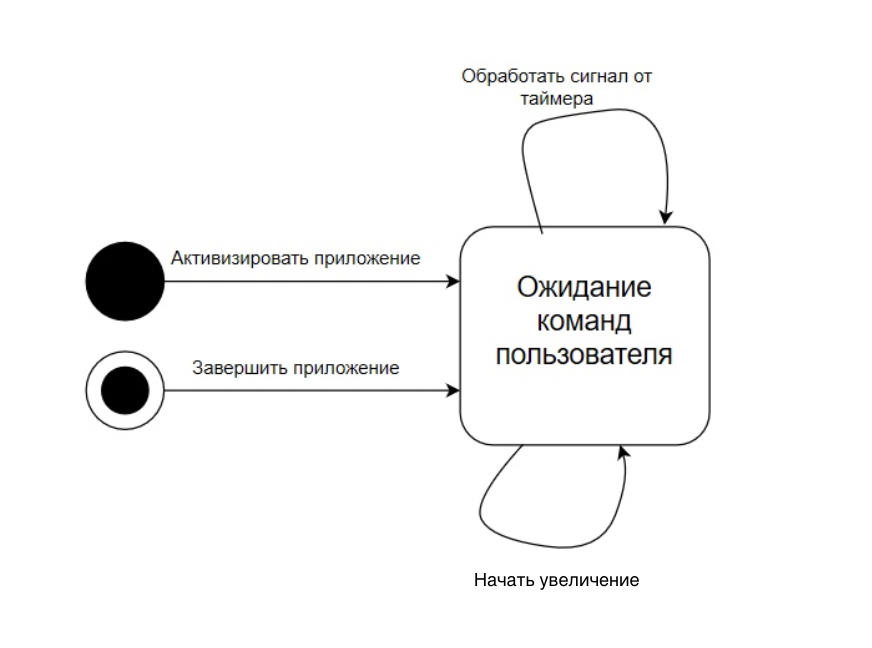
**Внешний вид формы:**

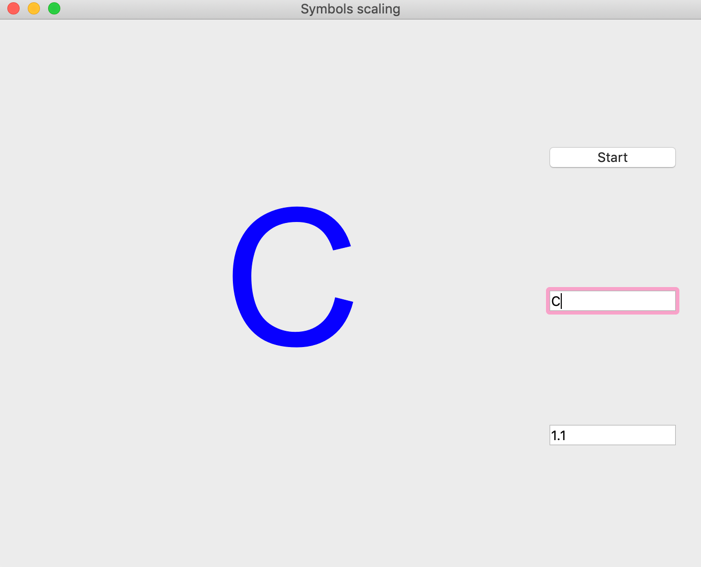
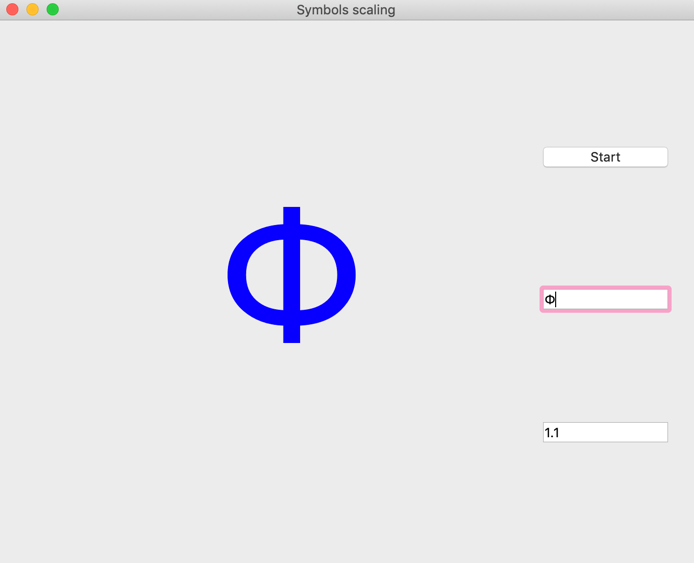
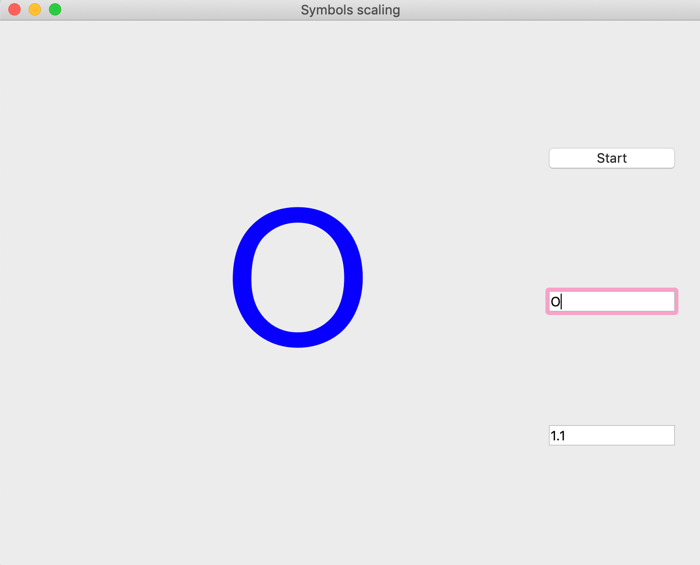
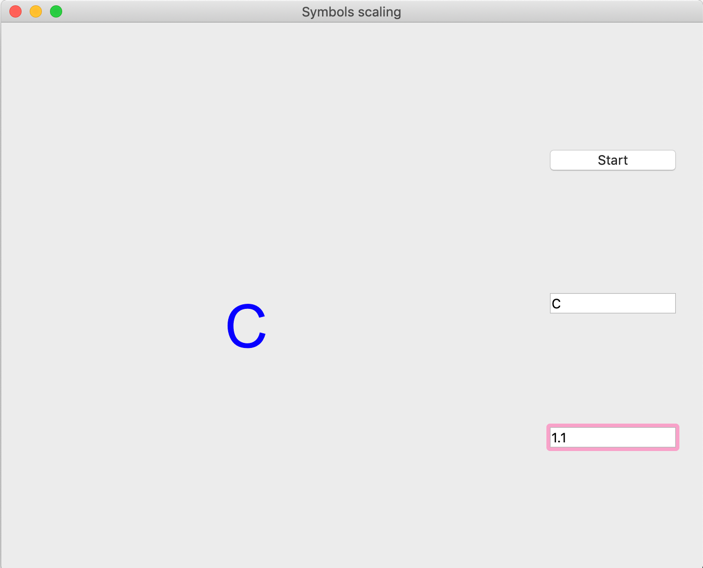
****

**Диаграмма классов:**

****

**Граф состояний интерфейса:**

****

****

**Вывод:** По результату выполнения второй части домашнего задания, я реализовал классы, с использованием полиморфного наследования – классы-потомки переопределяли метод базового класса. При заданных исходных параметрах программа успешно справляется со своей работой.