|  |  |
| --- | --- |
| 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 домашнему заданию №** | 3 |

**Название:**

Программирование на C++ с использованием классов

**Дисциплина:** Объектно-ориентированное программирование

Вариант 16.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-21Б |  |  | Т.Е.Старжевский |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | О.А.Веселовская |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2022

Часть 3.1. Композиция

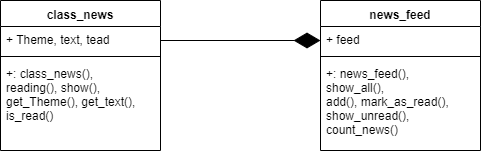
Разработать и реализовать диаграмму классов для описанных объектов предметной области, используя механизм композиции. Протестировать все методы каждого класса. Все поля классов должны быть скрытыми (private) или защищенными (protected). Методы не должны содержать операций ввода/вывода, за исключением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

Объект – новость. Поля: тема, текст сообщения, признак «прочитано». Методы: процедура инициализации объекта, процедура вывода информации об объекте на экран, процедура «прочтения» новости и функции, возвращающие по запросу значения полей.

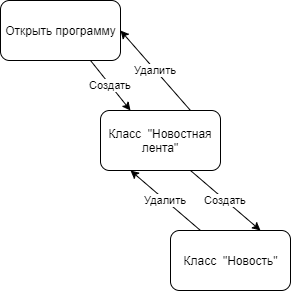
Объект – новостная лента. Состоит из массива новостей. Методы объекта должны позволять: инициализировать объект, выводить на экран всю ленту, добавлять новость в ленту, отмечать как прочитанную новость с заданным номером, выводить на экран только непрочитанные новости с их одновременным «прочтением», узнавать количество новостей в ленте.

В отчете привести диаграмму разработанных классов и объектную декомпозицию.

**Диаграмма разработанных классов:**

****

**Объектная декомпозиция:**

****

**Код программы:**

#pragma once

#include <iostream>

#include <vector>

#define n 10

using namespace std;

struct new\_time

{

string Theme, text;

};

class class\_news

{

private:

string Theme, text;

bool read;

public:

class\_news(string iTheme = "None", string itext = "None", bool iread = false)

:Theme (iTheme), text (itext), read (iread) {}

void reading()

{

if (read)

cout << "\nAlready was written\n";

else

read = true;

}

void show()

{

cout << "===== " << Theme << " =====\n" << text << "\n";

reading();

}

string get\_Theme ()

{

return Theme;

}

string get\_text()

{

return text;

}

bool is\_read()

{

return read;

}

};

class news\_feed

{

public:

vector<class\_news> feed;

news\_feed() = default;

news\_feed(vector <new\_time> inews\_feed)

{

for (int i = 0; i < inews\_feed.size(); i++)

{

feed.push\_back(class\_news(inews\_feed[i].Theme, inews\_feed[i].text));

}

}

void show\_all()

{

int i = 0;

while (i < feed.size())

{

feed[i++].show();

cout << endl;

}

}

void add(new\_time inews\_time)

{

feed.push\_back(class\_news(inews\_time.Theme, inews\_time.text));

}

void mark\_as\_read(int num)

{

feed[num].reading();

}

void show\_unread()

{

int i = 0;

while (i < feed.size())

{

if (!feed[i].is\_read())

{

feed[i].show();

cout << endl;

}

i++;

}

}

int count\_news()

{

return feed.size();

}

};

#include <iostream>

#include "classes.h"

#include <fstream>

#include <string>

#include <conio.h>

using namespace std;

int main()

{

news\_feed News\_feed; //Class

new\_time Inew\_feed; //struct //C:\Users\yarik\Desktop\Standard\BMSTU\2 sem\OOP\DZ\DZ3\DZ3

fstream file; string name = "C:\\Users\\yarik\\Desktop\\Standard\\BMSTU\\2 sem\\OOP\\DZ\\DZ3\\DZ3.part1\\Times.txt";

file.open(name, fstream::in);

if (!file.is\_open())

cout << "File doesn`t exist. Name: " << name;

else

{

while (!file.eof())

{

getline(file, Inew\_feed.Theme);

getline(file, Inew\_feed.text);

News\_feed.add(Inew\_feed);

}

cout << "Enter to read the feed\n";

if (\_getch() == '\r')

{

News\_feed.show\_all();

cout << "-> It is end of feed\n";

}

Inew\_feed = { "Visual Studio","Today visual studio have been updated\nAll changes you can see in our documents" };

News\_feed.add(Inew\_feed);

//Unread

cout << "--> Enter to see unread messages\n";

if (\_getch() == '\r')

{

cout << "#UNREAD messages:" << endl;

News\_feed.show\_unread();

}

}

system("Pause");

return 0;

}

**Тестирование программы:**

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат |
| Файл с новостями | Вывод информации об объектах |  |

При заданных значениях программа работает корректно.

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

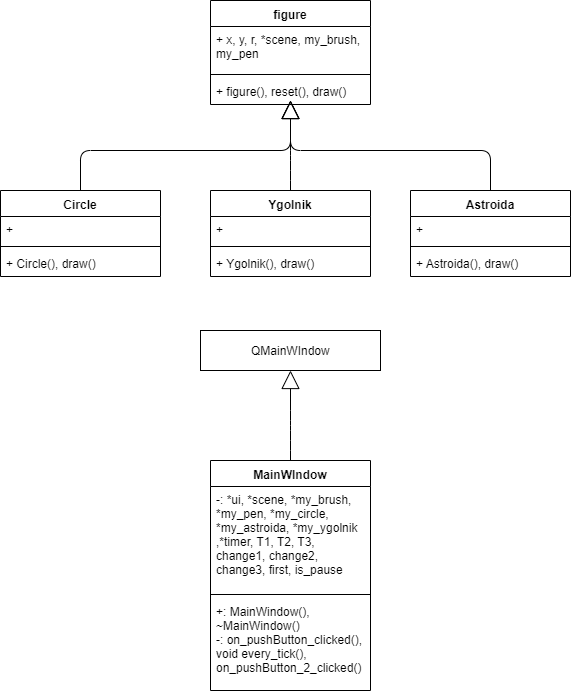
Разработать программу, содержащую описание трех графических объектов:

окружность, правильный шестиугольник, астроида.

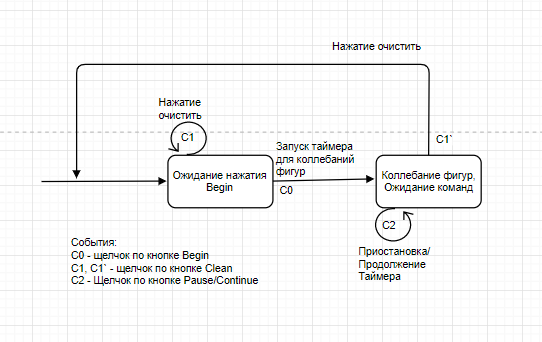
Реализуя механизм полиморфизма, привести объекты в одновременное колебательное движение вдоль горизонтальной оси с различными периодами колебаний.

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

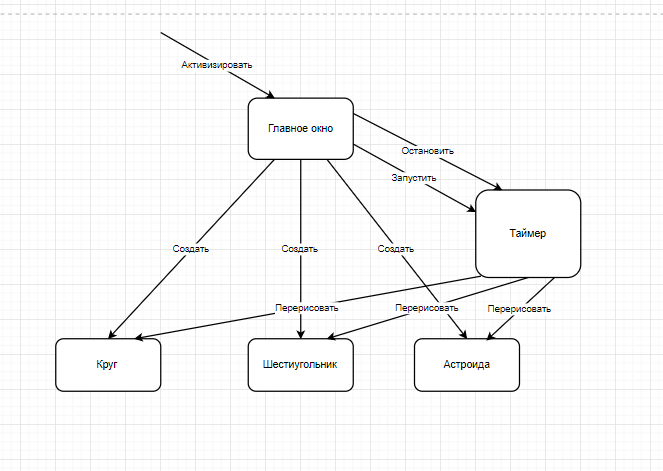
**Иерархия классов:**

****

**Графическое состояние интерфейса:**



**Объектная декомпозиция:**



**Код программы:**

#ifndef CLASSES\_H

#define **CLASSES\_H**

#include <QGraphicsItem>

#include <QPolygon>

#include <QtMath>

#include "qgraphicsscene.h"

#include "qbrush.h"

#include "qpen.h"

class **figure**

{

public:

qreal **x**,**y**,**r**;

QGraphicsScene \***scene**;

QBrush **my\_brush**;

QPen **my\_pen**;

**figure**(qreal **ix** = 0, qreal **iy** = 0, qreal **ir** = 0,QGraphicsScene \***iscene** = nullptr,

QBrush **imy\_brush** = QBrush(), QPen **imy\_pen** = QPen());

void **reset**(qreal **ix** = 0, qreal **iy** = 0, qreal **ir** = 0);

virtual void ***draw***(QPen **imy\_pen** = QPen(), QBrush **imy\_brush** = QBrush()) = 0;

};

class **Circle**: public figure

{

public:

**Circle**(qreal **ix** = 0, qreal **iy** = 0, qreal **ir** = 0, QGraphicsScene \***iscene** = nullptr,

QBrush **imy\_brush** = QBrush(), QPen **imy\_pen** = QPen());

void ***draw*** (QPen **imy\_pen** = QPen(), QBrush **imy\_brush** = QBrush()) override;

};

class **Ygolnik**: public figure

{

public:

**Ygolnik** (qreal **ix** = 0, qreal **iy** = 0, qreal **ir** = 0, QGraphicsScene \***iscene** = nullptr,

QBrush **imy\_brush** = QBrush(), QPen **imy\_pen** = QPen());

void ***draw*** (QPen **imy\_pen** = QPen(), QBrush **imy\_brush** = QBrush()) override;

};

class **Astroida**:public figure

{

public:

**Astroida** (qreal **ix** = 0, qreal **iy** = 0, qreal **ir** = 0, QGraphicsScene \***iscene** = nullptr,

QBrush **imy\_brush** = QBrush(), QPen **imy\_pen** = QPen());

void ***draw*** (QPen **imy\_pen** = QPen(), QBrush **imy\_brush** = QBrush()) override;

};

#endif // CLASSES\_H

#include "classes.h"

figure::**figure**(qreal **ix**, qreal **iy**, qreal **ir**,QGraphicsScene \***iscene**, QBrush **imy\_brush**, QPen **imy\_pen**):

x(ix), y(iy), r(ir), scene(*iscene*), my\_brush(imy\_brush),my\_pen(imy\_pen)

{

}

void figure::**reset**(qreal **ix**, qreal **iy**, qreal **ir**)

{

x= ix;

y = iy;

r = ir;

}

Circle::**Circle**(qreal **ix**, qreal **iy**, qreal **ir**,QGraphicsScene \***iscene**, QBrush **imy\_brush**, QPen **imy\_pen**):figure(ix,iy,ir,*iscene*, imy\_brush, imy\_pen)

{

}

void Circle::***draw***(QPen **imy\_pen**, QBrush **imy\_brush**)

{

my\_pen = imy\_pen;

my\_brush = imy\_brush;

this->scene->addEllipse(x, y, 2\*r, 2\*r, my\_pen, my\_brush);

}

Ygolnik::**Ygolnik**(qreal **ix**, qreal **iy**, qreal **ir**, QGraphicsScene \***iscene**, QBrush **imy\_brush**, QPen **imy\_pen**):figure(ix,iy,ir,*iscene*, imy\_brush, imy\_pen)

{

}

void Ygolnik::***draw***(QPen **imy\_pen**, QBrush **imy\_brush**)

{

my\_pen = imy\_pen;

my\_brush = imy\_brush;

QPolygon **polygon**;

polygon << QPoint(x+r,y) << QPoint (x+r/2, y - r \* sqrt(3)/2) << QPoint (x - r/2, y - r \* sqrt(3)/2)

<< QPoint(x - r, y) << QPoint (x - r/2, y + r \* sqrt(3)/2) << QPoint (x+r/2, y + r \* sqrt(3)/2);

this->scene->addPolygon(polygon, my\_pen, my\_brush);

}

Astroida::**Astroida**(qreal **ix**, qreal **iy**, qreal **ir**, QGraphicsScene \***iscene**, QBrush **imy\_brush**, QPen **imy\_pen**):figure(ix,iy,ir,*iscene*, imy\_brush, imy\_pen)

{

}

void Astroida::***draw***(QPen **imy\_pen**, QBrush **imy\_brush**)

{

my\_pen = imy\_pen;

my\_brush = imy\_brush;

QPolygon **polygon**;

for (int **i** = 0; i < 361; ++i)

{

polygon << QPoint (x + r \* pow(sin(qDegreesToRadians(i)),3), y + r \* pow(cos(qDegreesToRadians(i)),3));

}

this->scene->addPolygon(polygon, my\_pen, my\_brush);

}

#ifndef MAINWINDOW\_H

#define **MAINWINDOW\_H**

#include <QGraphicsItem>

#include <QMainWindow>

#include "classes.h"

#include <time.h> //не нужно

#include <QTimer>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \***parent** = nullptr);

~**MainWindow**();

private slots:

void **on\_pushButton\_clicked**();

private:

Ui::MainWindow \***ui**;

QGraphicsScene \***scene**;

QBrush \***my\_brush**;

QPen \***my\_pen**;

Circle \***my\_circle**;

Ygolnik \***my\_ygolnik**;

Astroida \***my\_astroida**;

QTimer \***timer**;

qreal **T1**, **T2**, **T3**;

bool **change1**, **change2**, **change3**, **first**, **is\_pause**;

private slots:

void **every\_tick**();

void **on\_pushButton\_2\_clicked**();

};

#endif // MAINWINDOW\_H

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \***parent**)

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

scene = new QGraphicsScene(this);

ui->graphicsView->setScene(*scene*);

my\_pen = new QPen(Qt::blue);

my\_pen->setWidth(5);

my\_brush = new QBrush(Qt::red);

my\_circle = new Circle(0, 0, 0, *scene*, \*my\_brush, \*my\_pen);

my\_ygolnik = new Ygolnik(0, 0, 0, *scene*, \*my\_brush, \*my\_pen);

my\_astroida = new Astroida (0, 0, 0, *scene*, \*my\_brush, \*my\_pen);

srand(time(NULL)); // Не нужно

first = true;

T1 = T2 = T3 = 0;

change1 = change2 = change3 = is\_pause = false;

timer = new QTimer(this);

connect(timer, SIGNAL(timeout()), this, SLOT(every\_tick()));

}

MainWindow::~**MainWindow**()

{

delete ui;

}

void MainWindow::**on\_pushButton\_clicked**()

{

if (first)

{

timer->start(10); //Каждую 1 мс вызываем timeout

ui->pushButton->setText("Pause");

first = false;

} else {

if (is\_pause)

{

ui->pushButton->setText("Pause");

is\_pause = false;

timer->start(10); //Каждую 1 мс вызываем timeout

} else

{

ui->pushButton->setText("Continue");

is\_pause = true;

timer->stop();

}

}

}

void MainWindow::**every\_tick**()

{

scene->clear();

if (change1)

{

T1 -= 1;

if (T1 < -120) change1 = false;

} else {

T1 += 1;

if (T1 > 120 ) change1 = true;

}

if (change2)

{

T2 -= 2;

if (T2 < -120) change2 = false;

} else {

T2 += 2;

if (T2 > 120 ) change2 = true;

}

if (change3)

{

T3 -= 3;

if (T3 < -120) change3 = false;

} else {

T3 += 3;

if (T3 > 120 ) change3 = true;

}

my\_pen->setColor(Qt::white);

my\_pen->setWidth(0);

my\_brush->setColor(Qt::yellow);

my\_circle->reset(-150, T1 - 70, 70);

my\_circle->*draw*(\*my\_pen, \*my\_brush);

my\_brush->setColor(QColor (212, 175, 55));

my\_ygolnik->reset(185, T2, 70);

my\_ygolnik->*draw*(\*my\_pen, \*my\_brush);

my\_brush->setColor(QColor(212, 50, 205));

my\_astroida->reset (450, T3, 70);

my\_astroida->*draw* (\*my\_pen, \*my\_brush);

}

void MainWindow::**on\_pushButton\_2\_clicked**()

{

ui->graphicsView->scene()->clear();

timer->stop();

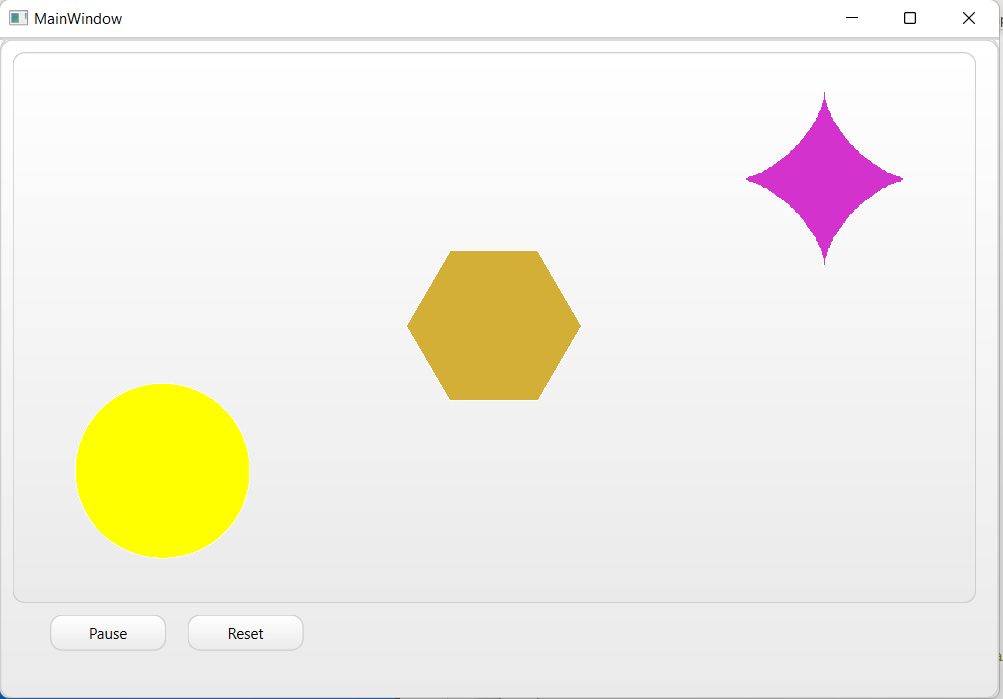
first = true;

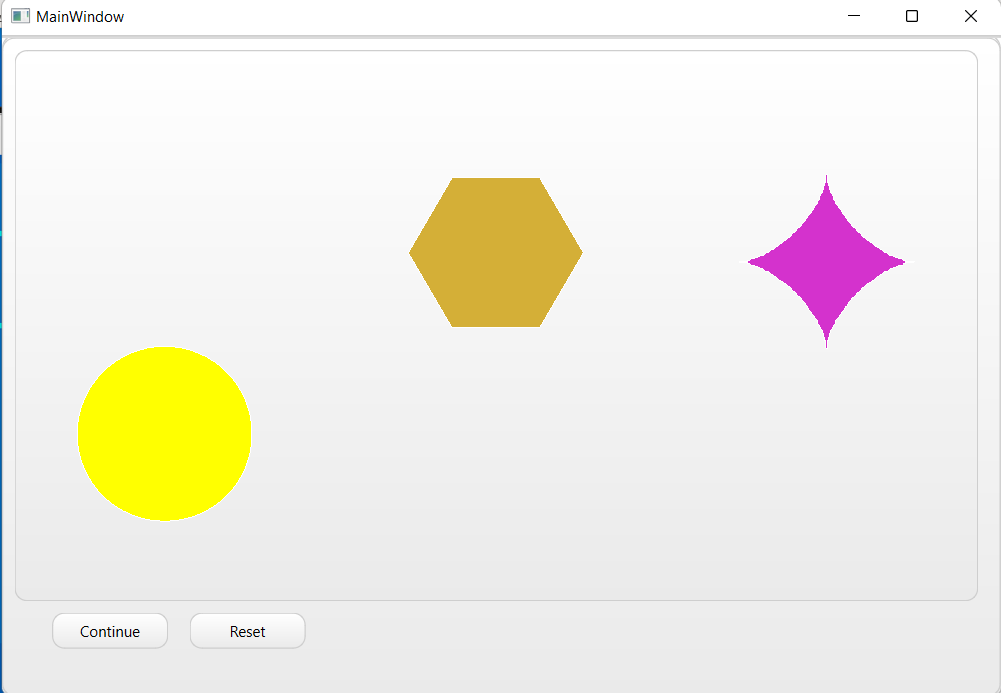
T1 = T2 = T3 = 0;

change1 = change2 = change3 = is\_pause = false;

ui->pushButton->setText("Begin");

}





**Вывод:** Научился программировать на C++ с использованием классов.