**Правительство Российской Федерации**

Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Кафедра компьютерной безопасности

**ОТЧЁТ**

**К ЛАБОРАТОРНОЙ РАБОТЕ №10**

**по дисциплине**

**«Языки программирования»**

Работу выполнил

студент группы СКБ-193 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А.Сурков

подпись, дата

Работу проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.А.Булгаков

подпись, дата

Москва 2020

**Постановка задачи**

Разработать консольное приложение с использованием библиотеки Qt. При запуске формируется вектор, содержащий имена файлов текущего каталога и их размеры. Из этого вектора формируется множество имен файлов таким образом, чтобы их размеры составляли неубывающую последовательность. Данное множество выводится на консоль по следующим правилам: ограничителями являются фигурные скобки; элементы разделяются запятой.

При каждом событии таймера из множества удаляется элемент, среднеквадратическое отклонение размера которого от 3/4 среднего размера файлов, оставшихся в множестве, минимально, после чего обновленное множество выводится на консоль.

Всякий раз после вывода множества на новой строке печатается приглашение ко вводу команды (символ '>').

Организовать работу приложения таким образом, чтобы по таймеру выводилась информация о текущем состоянии множества, а при поступлении данных на стандартный ввод обрабатывалась команда пользователя. Предусмотреть следующие команды:

1)**stop**– остановка таймера;  
2)**start** – запуск таймера;  
3)**restart** – перезапуск приложения;  
4)**timeout** <**целое**> – задает новый интервал для таймера;  
5)**exit** – завершение приложения.

**1. Алгоритм решения задачи**

Данная программа написана с помощью фреймворка Qt 4.8.7 и состоит из файла, в котором реализован класс Klass, с его свойствами, методами, сигналами и слотами (Klass.h), которые, в свою очередь, решают задачи лабораторной работы. Также, программа содержит главный файл (main.cpp), в котором подключается класс Klass и запускается цикл событий. UML- диаграмма классов представлена на рисунке 1.

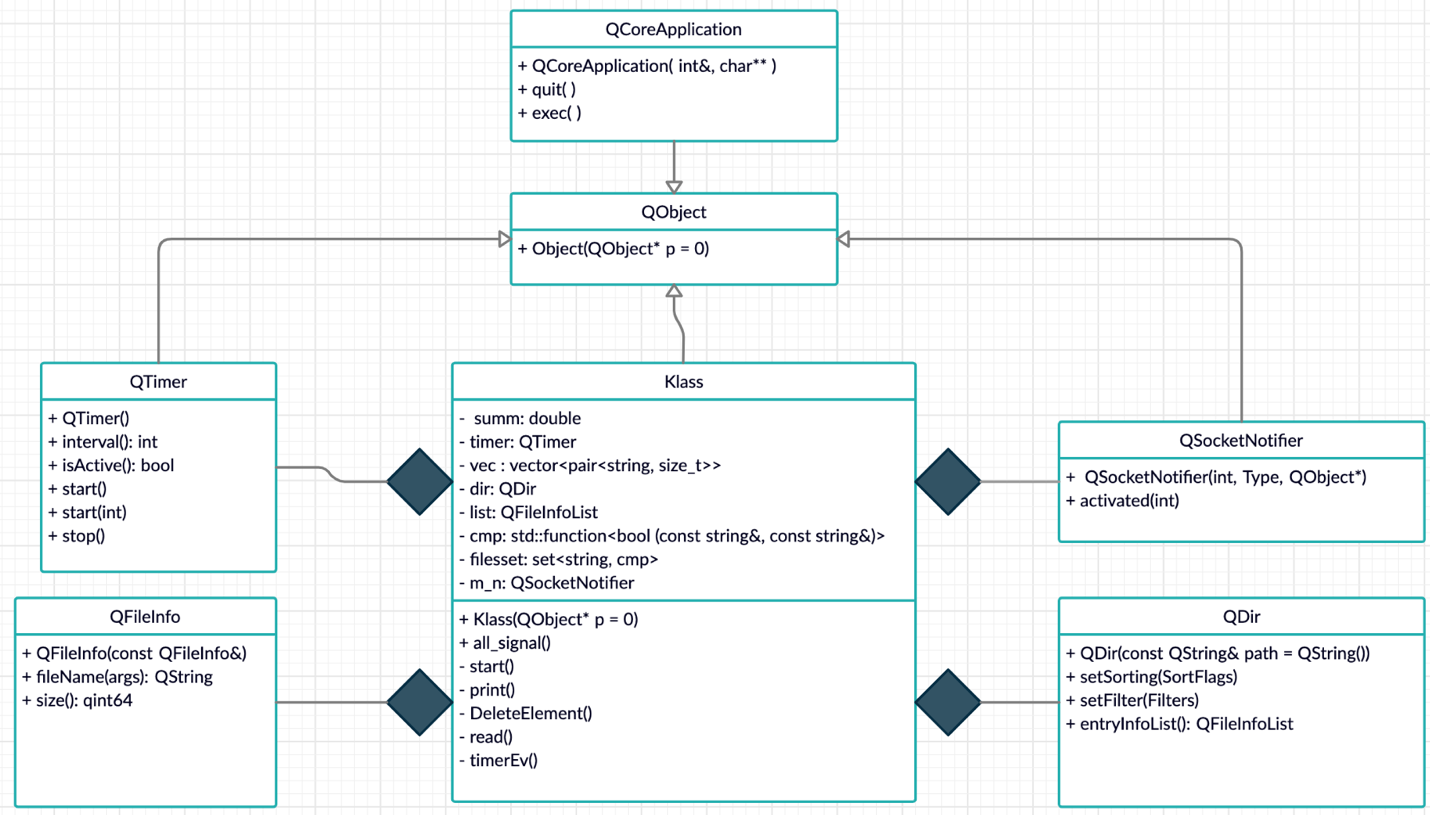


Рисунок 1 – UML диаграмма классов

**2. Выполнение задания**

**2.1. Файл Klass.h**

В данном файле определен и реализован класс Klass.  
Данный класс открыто наследуется от **QObject**, содержащегося в Qt. При запуске программы формируется вектор **(vector<pair<string, size\_t>> vec)**, содержащий имена и размеры файлов в текущем каталоге. Данные о каталоге получается с помощью объекта класса **QDir (QDir dir)**. На основе этих данных формируется множество **(set<string, cmp> filesset)** ,состоящее из имён файлов текущего каталога,таким образом, чтобы их размеры составляли неубывающую последовательность. Чтобы в множестве элементы находились в том же порядке что и в векторе был создан компаратор cmp. Все это реализовано в функции **start()**. Также в классе содержится таймер **(QTimer timer)**, который через заданный интервал времени посылает сигнал **timeout().** Этот сигнал обрабатывается слотом **timerEv()**, который удаляет из множества элемент, среднеквадратическое отклонение размера которого от 3/4 среднего размера файлов, оставшихся в множестве, минимально, затем выводит множество уже с удаленным файлом на экран. Удаление и вывод производятся в функциях **DeleteElement()** и **print()**. Также в классе содержится объект класса **QSocketNotifier (QSocketNotifier m\_n)**, который каждый раз, когда пользователь вводит команду, выпускается сигнал **activated(int)**. Этот сигнал обрабатывается слотом **read()**, который считывает эти данные и, в зависимости от команды введенной пользователем, выполняет действия такие как: ставит таймер на паузу при помощи функции **stop()** , возобновляет таймер с помощью функции **start()** , задаёт новый интервал у таймера при помощи функции **start(int)** , начинает работу приложения заново при помощи функции **start()** из класса **Klass**. При поступлении команды **exit** генерируется сигнал **all\_signal()**.

**2.2. Файл main.cpp**

В данном файле подключается заголовочный файл класса Klass и создается обьект этого класса. Также создаётся экземпляр класса **QCoreApplication**, который предоставляет слот **quit()** для обработки сигнала **all\_signals()**. После запускается цикл событий.

**3. Процедура получения исполняемых файлов**

В программе есть файл 2.pro, который создает qmake, а сам qmake создает уже makefile.

**4. Тестирование программы**

Результат работы программы представлен в листинге 1.

Листинг 1

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, .DS\_Store, Makefile, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, .DS\_Store, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, moc\_Klass.o, 2 } > { main.cpp, 2.pro, 2 }

>

{ main.cpp, 2 }

> { 2 }

>

{ }

>exit

Как видим программа сработала верно и функция **exit()** тоже.

Протестируем другие функции реализованный в данном консольном приложении **stop(), start(), timeout (int)** .

Результат работы программы представлен в листинге 2.

Листинг 2

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, .DS\_Store, Makefile, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, .DS\_Store, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, main.o, moc\_Klass.o, 2 }

> stop

> timeout 4000

> stop

> start

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, main.o, moc\_Klass.o, 2 }

> restart

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, .DS\_Store, Makefile, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, .DS\_Store, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, Klass.h, main.o, moc\_Klass.o, 2 }

>

{ main.cpp, 2.pro, .Klass.h.un~, .main.cpp.un~, moc\_Klass.cpp, main.o, moc\_Klass.o, 2 }

> exit

**Приложение А**

A1. Исходный код файла **main.cpp**

#include <QCoreApplication>

#include "Klass.h"

#include <QTimer>

using namespace std;

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

{

QCoreApplication a(argc, argv);

Klass klass;

QObject::connect(&klass, SIGNAL(all\_signal()), &a, SLOT(quit()));

return a.exec();

}

A2. Исходный код файла **Klass.h**

#ifndef KLASS\_H

#define KLASS\_H

#include <iostream>

#include <QDir>

#include <QTimerEvent>

#include <QSocketNotifier>

#include <vector>

#include <set>

#include <QTimer>

#include <cmath>

#include <functional>

using namespace std;

class Klass: public QObject

{

Q\_OBJECT

private:

QDir dir;

vector <pair<string, size\_t>> vec;

std::function<bool (const string&, const string&)> cmp ;

set <string, decltype(cmp)> filesset;

QTimer timer;

QSocketNotifier m\_n;

void print(){

bool flag = true;

std::cout << std::endl << "{ ";

for (const string& file: filesset) {

if (!flag) {

cout << ", ";

} else {

flag = false;

}

cout << file;

}

cout << " }";

cout << endl <<"> " << flush;

}

void DeleteElement(){

double rms;

double summ = 0;

bool flag = false;

double min;

int delete\_element = 0;

for(auto element: vec){

summ += element.second;

}

summ = summ \* 0.75 / vec.size();

if (vec.size() == 0 || filesset.size() == 0){

return;

}

if (filesset.size() == 1){

filesset.clear();

return;

}

for (size\_t i = 0; i < vec.size(); i++){

int size\_file = vec[i].second;

rms = sqrt((size\_file - summ)\*(size\_file - summ));

if (flag == false){

min = rms;

flag = true;

}

if (min > rms){

min = rms;

delete\_element = i;

}

}

filesset.erase(filesset.find(vec[delete\_element].first));

vec.erase(vec.begin() + delete\_element);

}

public:

explicit Klass(QObject \*p = nullptr)

: QObject(p), m\_n(fileno(stdin), QSocketNotifier::Read, this)

{

cmp = [&](const string& s1, const string& s2)->bool{

int q1size = 0;

int q2size = 0;

for (size\_t i = 0; i < vec.size(); i++){

string name = vec[i].first;

if (s1 == name){

q1size = vec[i].second;

}

if (s2 == name){

q2size = vec[i].second;

}

}

return q1size < q2size;

};

dir.setFilter(QDir::Files | QDir::Hidden | QDir::NoSymLinks);

dir.setSorting(QDir::Size | QDir::Reversed);

QObject::connect(&timer, SIGNAL(timeout()), this, SLOT(timerEv()));

start();

QObject::connect(&m\_n, SIGNAL(activated(int)), this, SLOT(read()));

}

private slots:

void timerEv(){

DeleteElement();

print();

}

void start(){

QFileInfoList list = dir.entryInfoList();

vec.clear();

filesset.clear();

vec.resize(list.size());

filesset = set<string, decltype(cmp)> (cmp);

for (int i = 0; i < list.size(); i++){

QFileInfo fileInfo = list.at(i);

vec[i].first = fileInfo.fileName().toStdString();

vec[i].second = fileInfo.size();

filesset.insert(vec[i].first);

}

print();

if (timer.interval() == 0) {

timer.start(3000);

} else {

timer.start();

}

}

void read(){

string cmd;

getline(cin, cmd);

if (cmd == "stop") {

timer.stop();

}

else if (cmd == "start") {

if (!timer.isActive()) {

timer.start();

}

} else if (cmd == "restart") {

start();

} else if (cmd == "exit") {

emit all\_signal();

} else if (cmd.length() >=9 && cmd.substr(0, 7) == "timeout") {

string str = cmd.substr(8);

if (isdigit(str[0])) {

int num = stoi(str);

timer.start(num);

}

}

if (cmd != "restart") {

cout << "> " << std::flush;

}

}

signals:

void all\_signal();

};

#endif // KLASS\_H