Столяров Владимир
Студент Б14-507
Алгоритм Рабина-Карпа
Лабораторная работа 4

ОТЧЕТ

Цель лабораторной работы №4: изучение возможностей диаграммы классов, а также приобретение навыков создания диаграмм классов и применения этих диаграмм для создания кода на языке C++ (шаблона приложения).

Диаграмма классов UML позволяет обозначать отношения между классами и их экземплярами. Диаграмма классов является ключевым элементом в объектно-ориентированном моделировании. На диаграмме классы представлены в рамках, содержащих три компонента:

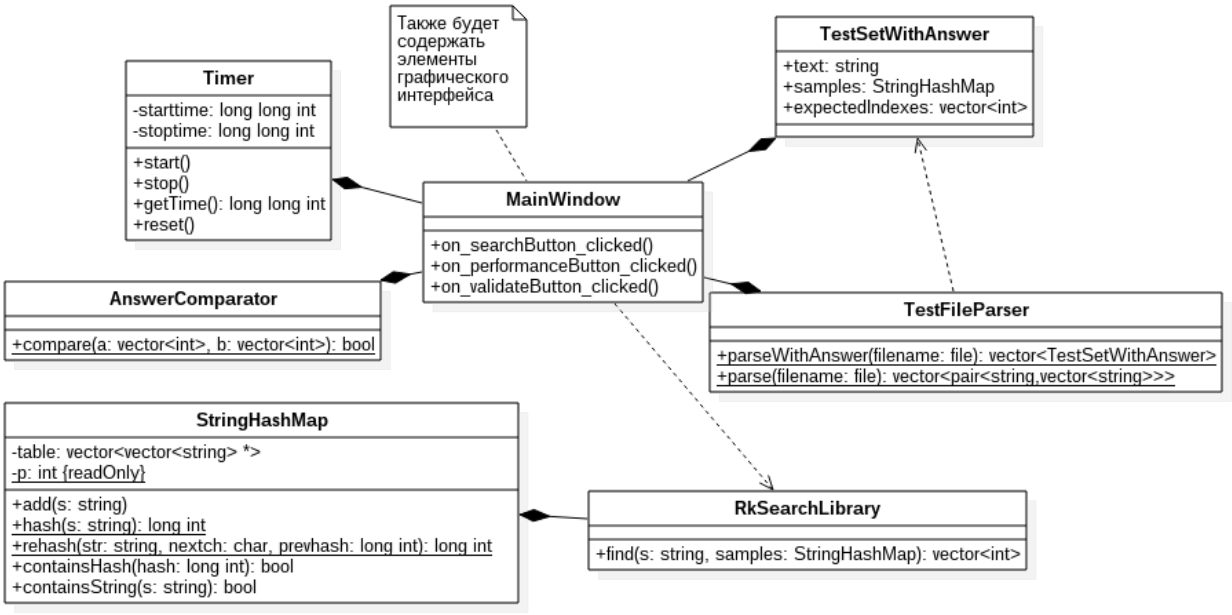
В верхней части написано имя класса. Имя класса выравнивается по центру и пишется полужирным шрифтом. Имена классов начинаются с заглавной буквы. Если класс абстрактный — то его имя пишется полужирным курсивом.

Посередине располагаются поля (атрибуты) класса. Они выравнены по левому краю и начинаются с маленькой буквы. Статические поля выделены подчёркиванием. Поля «только для чтения» помечены как «{readOnly}».

Нижняя часть содержит методы класса. Они также выравнены по левому краю и пишутся с маленькой буквы.

Между классами существуют несколько видов взаимодействий, обозначенных ниже:

Ниже представлена диаграмма классов проекта, разрабатываемая в рамках серии данных лабораторных работ.



Классы RkSearchLibrary и StringHashMap реализованы в разделяемой библиотеке (для возможности работы с действующим лицом «внешняя система»).

Класс RkSearchLibrary содержит основной функционал, необходимый для выполнения алгоритма поиска: метод find.

Класс StringHashMap — реализация хеш-таблицы для заранее заданного количества строк. Использует кольцевую хеш-функцию для быстроты пересчёта хеша (ключ к производительности алгоритма).

Класс MainWindow описывает главное окно графического приложения, в котором реализуются варианты использования: проверка производительности (с помощью класса Timer, отсчитывающего время выполнения алгоритма в наносекундах и класса TestFileParser содержит метод, разбирающего файл, содержащего наборы тестовых данных), проверка корректности (класс TestFileParser содержит метод, разбирающий файл, содержащий тестовые наборы и эталонные ответы, представляемые классом TestSetWithAnswer, класс AnswerComparator содержит метод, сравнивающий полученный ответ с эталоном).

Замечание: в данной работе были опущены конструкторы и деструкторы классов. По диаграмме классов удалось успешно сгенерировать код-шаблон приложения (сигнатуры некоторых методов были незначительно изменены для повышения производительности, например, использована передача объектов по константным ссылкам). Генерация производилась с помощью расширения «C++ code generation and reverse engineering» для StarUML. Код генерируется сразу готовым для документации с использованием doxygen (расставляются специализированные комментарии).

Сгенерированный код (только заголовочные файлы):

```
#ifndef _STRINGHASHMAP_H
#define _STRINGHASHMAP_H

class StringHashMap {
public:

    /**
```

```
     * @param s
     */
    void add(string s);

    /**
     * @param s
     */
    static long int hash(string s);

    /**
     * @param prevch
     * @param nextch
     * @param prevhash
     */
    static long int rehash(str string, char nextch, long int prevhash);

    /**
     * @param hash
     */
    bool containsHash(long int hash);

    /**
     * @param s
     */
    bool containsString(string s);
private:
    vector<vector<string>*> table;
    static const int p;
};

#endif //_STRINGHASHMAP_H

#ifndef _RKSEARCHLIBRARY_H
#define _RKSEARCHLIBRARY_H

class RkSearchLibrary {
public:

    /**
     * @param s
     * @param samples
     */
    vector<int> find(string s, StringHashMap samples);
};

#endif //_RKSEARCHLIBRARY_H
```

```
#ifndef _ANSWERCOMPARATOR_H
#define _ANSWERCOMPARATOR_H

class AnswerComparator {
public:

    /**
     * @param a
     * @param b
     */
    static bool compare(vector<int> a, vector<int> b);
};

#endif //_ANSWERCOMPARATOR_H

#ifndef _MAINWINDOW_H
#define _MAINWINDOW_H

class MainWindow {
public:

    void on_searchButton_clicked();

    void on_performanceButton_clicked();

    void on_validateButton_clicked();
};

#endif //_MAINWINDOW_H

#ifndef _TESTFILEPARSER_H
#define _TESTFILEPARSER_H

class TestFileParser {
public:

    /**
     * @param filename
     */
    static vector<TestSetWithAnswer> parseWithAnswer(file filename);

    /**
     * @param filename
     */
    static vector<pair<string,vector<string>>> parse(file filename);
};
```

```
#endif //_TESTFILEPARSER_H

#ifndef _TESTSETWITHANSWER_H
#define _TESTSETWITHANSWER_H

class TestSetWithAnswer {
public:
   string text;
   StringHashMap samples;
   vector<int> expectedIndexes;
};

#endif //_TESTSETWITHANSWER_H

#ifndef _TIMER_H
#define _TIMER_H

class Timer {
public:

   void start();

   void stop();

   long long int getTime();

   void reset();
private:
   long long int starttime;
   long long int stoptime;
};

#endif //_TIMER_H
```

Выводы:

- диаграмма классов достаточно подробно описывает структуру будущей программы.
- мы убедились в том, что диаграммы классов позволяют достаточно полно описать атрибуты и методы будущих классов.
- диаграмма классов позволяет сгенерировать код – шаблон приложения, что облегчает его дальнейшую разработку.
- код - шаблон всё-же необходимо в дальнейшем редактировать.