# Пояснительная записка к программе «Частотный словарь» (FrequencyDictionary.App.exe)

## Задание

Составить частотный словарь для заданного текстового файла.

Программа должна быть реализована в виде консольного приложения. Первым параметром задается имя входного файла. Вторым параметром задается имя файла с результатом. Кодировкой исходного и результирующего файлов считается Windows-1251.

Слова разделяются пробелами и символами перевода строки. Слова сравниваются без учета регистра. Формат результирующего файла – «{0},{1}\n» где {0} – слово, а {1} – количество его вхождений в исходный файл.

Результирующий файл должен быть отсортирован по убыванию встречаемости.

Приложение должно обладать следующими качествами:

1) Производительностью, разумной экономностью ресурсов CPU и Memory.

2) Скалируемостью (от scalability) на многоядерных/многопроцессорных системах.

3) Использовать классы стандартной библиотеки.Net 3.5SP1 или 4.0.

4) Ожидать неправильного ввода от пользователя.

5) Иметь модульную декомпозицию и содержать юнит тесты.

6) Обладать архитектурой удобной для дальнейшего расширения функциональности.

7) Иметь описание архитектуры в любом удобном виде.

## Описание архитектуры

Основная идея построенной архитектуры – это возможность реиспользования в других проектах, облегчение тестирования, возможность расширения.

Решение (FrequencyDictionary) предоставляет из себя 4 проекта:

1. FrequencyDictionary.App – само консольное приложение. Не содержит логики для анализа, выполняет функцию запуска приложения в консоли, конфигурацию аналитической библиотеки, запуск модуля анализа, вывод информации об ошибке для пользователя в консоль. Может быть потенциально заменено на Windows Forms, WPF или Web приложение без необходимости правки модуля анализа.
2. TextMetrics.Core – библиотека анализа текстов. Содержит всю логику по организации анализа, самого конкретного анализа (частотного словаря), вывода результата. Позволяет расширять функциональность независимо от способа использования. Будет рассмотрена подробнее ниже.
3. TextMetrics.Core.Test – библиотека предоставляющая совокупность тестов для TextMetrics.Core. Содержит как отдельные юнит тесты для классов, использующихся в библиотеке анализа, так и интеграционный тест с анализом большого текста (примерно 0,5 Гб, 80 млн. слов) составленного из случайных слов, использующих русский алфавит.
4. TextMetrics.Model – проект, в котором находятся диаграммы описывающие архитектуру библиотеки анализа.

### Архитектура библиотеки анализа текстов (TextMetrics.Core)



Рисунок 1 Диаграмма компонентов библиотеки анализа

В сущности, саму библиотеку можно использовать в общем виде как некоторый модуль позволяющий осуществлять анализ текста с целью построения каких либо метрик описывающих сам текст. В данном частном случае реализован только количественный анализ используемых слов в тексте (частотный словарь).

Анализ текста разделяется на 4 фазы осуществляемые следующими модулями или сервисами:

1. Собственно парсер текста (Text Parser на диаграмме), который представляют собой абстракцию источника текста, и может предоставлять текст по некоторым логически разбитым участкам (например, строкам в текстовом файле). Реализует интерфейс ITextParser. Может внутри себя быть реализован асинхронно, важно только возвращать некоторую последовательность (очередь) фрагментов. Может существовать реализация читающая текст через HTTP, может быть читающая MSWord файл или XML. В данной программе реализация FileTextParser читает текстовый файл и разбивает его на строки разделенные символом конца строки.
2. Парсер для лексического анализа (Token Parser на диаграмме). Представляет из себя стратегию по распознавание и выделение лексем из входной последовательности символов и выдачу их в виде последовательности (очереди) токенов (термины – из <http://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7>). Реализует интерфейс ITokensParserStrategy. В данном конкретном случае в реализации AlphaTokensParserStrategy текст разбивается на слова, остальные сущности, такие как числа, например, опускаются. Может быть напротив реализован класс, который разбивает текст только на числа, опуская слова.
3. Агрегатор токенов (Token Aggregator на диаграмме). Осуществляет саму функцию анализа текста. Также предоставляет возможность по получению результата анализа в виде последовательности структур ключ-значение. Реализует интерфейс ITokensAggregator. В интерфейсе также специфицируется, возможно ли использование агрегатора из различных потоков одновременно, так как если он использует внутренние структуры для хранения промежуточных результатов анализа, они должны быть потокобезопасны.

Токены, полученные из Token Parser могут быть подвергнуты разнообразному анализу. В данном конкретном случае в реализации TokensFrequencyAggregator анализируется количество слов в тексте. Другие возможные реализации – например, посчитать количество палиндромов в тексте, посчитать количество слов начинающихся на букву «а» и т.д

1. Хранитель результатов анализа (Aggregation data Storage на диаграмме). Осуществляет функцию по сохранению, внешнему представлению результатов анализа текста, которые сделал Token Aggregator на предыдущем шаге. Реализует интерфейс IAggregeationStorage. В данном конкретном случае в реализации TextFileAggregationStorage сохраняет данные в текстовом файле в формате «{0},{1}\n» где {0} – ключ, а {1} – значение. Возможны и другие реализации, например, может быть реализован класс, который будет отправлять данные анализа по электронной почте.

Дополнительные модули:

1. Конфигурация (Configuration на диаграмме). Определяет конфигурацию анализатора, предоставляет методы для создания конкретных экземпляров сущностей реализующих 4 представленных выше модуля. Реализует интерфейс ITextMetricsConfiguration. Именно здесь может быть использована какая либо библиотека, осуществляющая Depency Injection и/или систему подключаемых модулей (Plugins)
2. Алгоритм подсчета метрик (Text metrics calculator на диаграмме). Определяет сам алгоритм подсчета метрики, объединяет отдельные модули сервисы с целью выполнения расчёта. Может определять последовательность, кратность, необходимость распараллеливания функций осуществляемых 4 модулями описанными выше. Использует ITextMetricsConfiguration для определения, какими именно реализациями модулей можно воспользоваться. Не задается интерфейсом в силу предоставления максимальной гибкости, и скорее всего единственности конкретной реализации. В программе используется класс TextMetricsCalculator, который может быть запущен в режиме параллельного лексического анализа, или в режиме последовательного – строка-за-строкой. В программе используется параллельный режим работы.

Примечания.

- В TokensFrequencyAggregator используется хэш таблица (Dictionary in C#) для накопления результатов анализа. В ней ключом является слово, значение - количество таких слов в тексте. Данная структура позволяет добавлять новое слово/увеличивать счетчик слова за время O(1). Количество слов в русском языке в общеупотребительных формах не превышает  40 тысяч (по данным "Частотного словаря русского языка" под ред. Л. Н. Засориной. М., 1977), <http://www.nkj.ru/archive/articles/389/>. Проведенный анализ текста «Война и мир» показал количество слов со словоформами ~53000, в библии на английском языке ~13000. Так что указанные данные говорят о том, что размер таблицы вряд ли превысит лимит в 100000 слов, а значит и ячеек, что даст нам не более (20 байт на ячейку (см в <http://www.dotnetperls.com/dictionary-memory>) + 20 байт на слово (возьмем 20 байт за средний объем памяти для одного слова учитывая ссылку на него и двухбайтные символы)) \* 100000 = 4 MB занимаемой памяти. Что в современных реалиях весьма малое потребление и, при условии бесконечности текста, стремящееся к O(1) по памяти.

- FileTextParser в процессе конструирования проводит проверку кодировки файла на совместимость с указанной в задании 1251, и проверят файл на то, является ли он текстовым или бинарным.

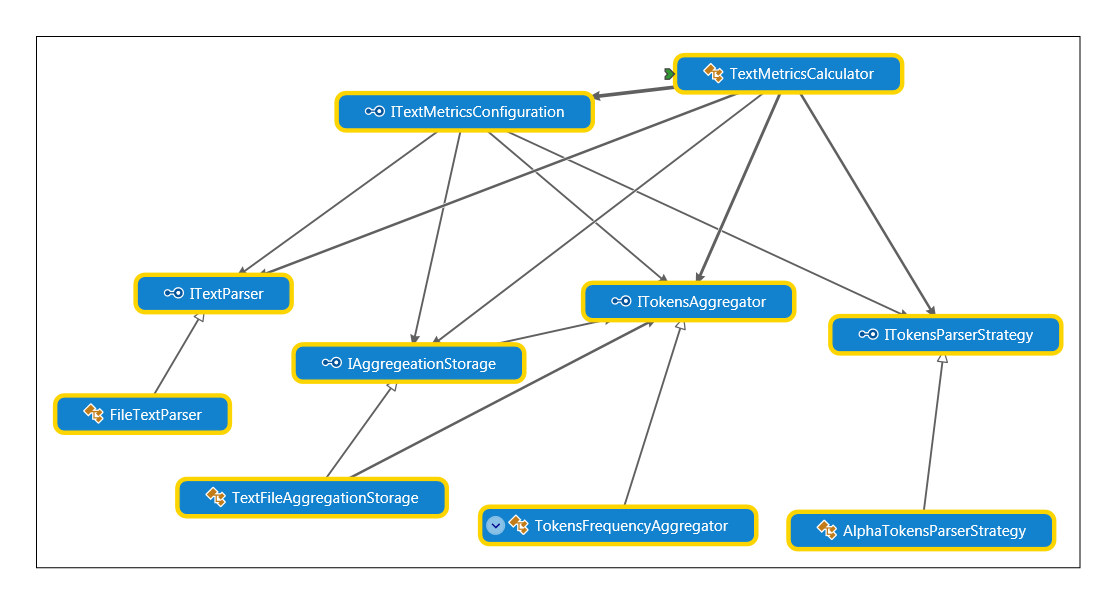


Рисунок 2 Диаграмма сущностей библиотеки анализа



Рисунок 3 Диаграмма классов библиотеки анализа



Рисунок 4 Диаграмма алгоритма реализуемого в программе