Системное и прикладное программное обеспечение.

Отчёт по первой части третьей лабораторной работы. Файловый обозреватель.

## 1. Постановка задачи.

Отдельным консольным приложением реализуем основную функцию – вычисление размера, в зависимости от стратегии.

Общий принцип работы следующий: пользователь указывает директорию, для которой требуется вычислить размер содержимого. Функция, в зависимости от выбранной стратегии, проводит вычисление, где результатом является набор соответствующих данных, вида:

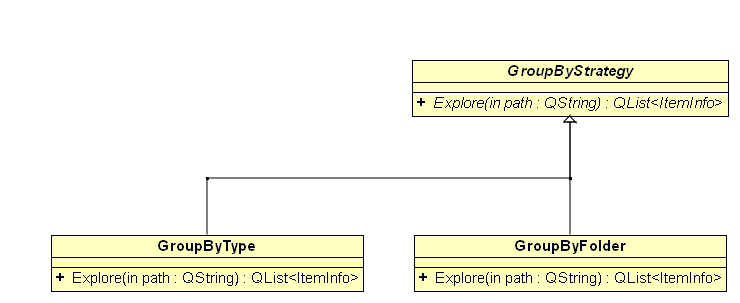
список файлов и папок (только верхнего уровня) / список типов файлов, содержащихся в директории, а также занимаемый ими объём в процентах (точность два знака после запятой; если точности не хватает, а размер элемента не равен нулю, требуется это показать - например, в виде "< 0.01%").

Основная цель – проработка всевозможных условий. С целью доказательства того факта, что всевозможные условия обработаны, требуется подготовить тесты (примеры), на которых будете демонстрировать состоятельность вашей функции.

## 2. Предлагаемое решение.

Рассмотрим паттерн стратегия, его смысл заключается в том что когда в программе одна задача решается разными алгоритмами, алгоритмы можно вынести в отдельные классы. Тогда задача вызывает алгоритм, которым она в данный момент хочет воспользоваться.

В данной задаче в виде стратегии я представил алгоритм группировки файлов. У неё получились две реализации: группировка по папкам, и по типам.



В общем классе стратегии объявлен один метод, который как раз и отвечает за группировку файлов по некоторому правилу. На входе он принимает директорию, которую требуется сгруппировать. Возвращает массив элементов директории, элементами могут являться папки или типы файлов, в зависимости от типа группировки.

Перед рассмотрением конкретных реализаций стоит остановиться подробнее на элементах директории. В данном случае элемент реализован в виде класса ItemInfo. Его задача заключается в том, что имея в качестве исходных данных имя, размер элемента и общий размер папки, вернуть по запросу имя без изменений, отформатированный размер элемента в подходящих единицах измерения, и отформатированное процентное соотношение данного элемента ко всему объёму папки. Для достижения данных возможностей используется встроенный класс QLocal, он позволяет преобразовать объём из байтов в удобный вид. При вычислении процентного соотношения реализованы всякого рода проверки, например избегается ситуация деления на ноль при группировки папки без файлов вообще, а так же при очень маленьком объёме файла по отношению к объёму всей папки, об этом сигнализирует значок, появляющийся слева от процентов “< 0.01 %”. Точность выводимых процентов – два знака после запятой. Стоит отметить что объёмы хранятся в виде 64 битных переменных, таким образом отсутствует ограничение в 4 гигабайта по объёму папки. Используется знаковый тип, ввиду того что именно он возвращается из функции получения объёма файла, так что приведение его к беззнаковому может только привести к проблемам, всё равно присутствует операция деления этих чисел для нахождения процентного соотношения, таким образом рано или поздно они вновь будут приведены к знаковому виду.

Вернёмся к стратегиям группировки. Мы остановились на стратегии группировки по папкам. Стоит сказать что в данной реализации файлы текущей папки интерпретируются как одна общая папка «(Current Directory)», так как и было показано в примере из лекции.

Так как данная стратегия предполагает только один метод Explore, его и рассмотрим. В данной реализации он принимает на вход путь директории которую требуется сгруппировать, и возвращает массив информации о папках директории. Первым делом получается информация о данной директории, и если она не является папкой, либо вообще не существует, выбрасывается исключение. Потом создаются объекты в которых хранится информация, например возвращаемый массив и накопитель общего размера папки. Затем данная директория открывается как папка. Обработка файлов начинается с прохода по всем файлам текущей папки. После того как текущая папка посчитана она заносится в массив. Переходим к подсчёту размера подпапок. Для этого перебираются все подпапки верхнего уровня, в процессе, для каждой подпапки создаётся итератор директории. С помощью этого итератора перебираются все файлы во всех подпапках данной подпапки. Единственное что интересует нас в этих файлах это их размер, который просто идёт в зачёт размера подпапки. После того как вся подпапка просканирована, она идёт в зачёт общего размера папки и добавляется в массив элементов. После всех этих процедур выясняется общий размер папки, таким образом нужно каждому из элементов сообщить общий размер папки, чтобы в дальнейшем они могли правильно вычислить процентное соотношение.

Вторая реализация отвечает за группировку файлов по типам. В результате получается массив элементов, в которых в качестве имени содержится тип файла, а размер говорит об общем размере всех файлов данного типа в данной директории. В начале так же происходит проверка валидности директории. Так как теперь нет возможности последовательно пройтись по какому-либо списку, возникает необходимость использования словаря. Таким образом сначала будет сформирован словарь, использующий в качестве ключа тип файла, а в качестве значения размер всех файлов данного типа в текущей директории. Для прохода по всем файлам на этот раз будет использоваться всего один цикл, по мере его прохождения новые типы файлов заносятся в словарь, и в зачёт размера типа данных идёт размер файлов, также он учитывается в общем размере всей директории. После прохождения всех файлов директории, необходимо переписать посчитанные значения из словаря в массив элементов. Так как при использовании цикла на основе диапазона словарь передаёт только значения, без ключей, возникает необходимость в использовании итератора. На этот раз при добавлении в массив элементов, общий размер директории известен.

Осталось разобрать файл с примерами. Для повышения удобства были созданы три вспомогательные функции.

Первая отвечает за вывод массива элементов в определённом формате. Вывод осуществляется в переданный текстовый поток с форматированием по полям. Выводится три столбика: имя элемента, его размер и процентное соотношение к общему размеру директории.

Следующая функция отвечает за вывод примера. То есть в неё передаётся стратегия и директория. В данной функции находится отлов ошибок, возникших при группировке директории. Он находится именно здесь, для того чтобы при возникновении исключения при группировке текущей директории, это не помешало следующим примерам.

Также есть функция PrintExample() в которой собраны все имеющиеся примеры. В неё соответственно передаётся текстовый поток и стратегия группировки.

В функции main создаётся текстовый поток, выводящий информацию в stdout, по аналогии с std::cout, но с поддержкой QString. Далее создаются стратегии группировки и вызывается функция вывода примеров по данным стратегиям.

## 3. Коды программ.

item\_info.h – информация об элементе строки

group\_by\_strategy.h – стратегия группировки

group\_by\_folder.h – группировка по папкам

group\_by\_type.h – группировка по типу

## 4. Инструкция пользователя.

Начнём со структуры данных, хранящей информацию ItemInfo. Данный класс с точки зрения пользователя нужен только для того, чтобы получать из него необходимую информацию. А именно: имя элемента, отформатированные размер и процентное соотношение. Соответственное методы, для их получения:

name() - возвращает ссылку на строку с именем

size() - возвращает отформатированный размер файла с единицей измерения

percentage() - возвращает процентное соотношение к общему размеру директории

Форматирование находится в данном классе для повышения удобства отображения информации, во второй части, посвящённой интерфейсу, это будет реализовано через MVC модель.

Для группировки файлов в некоторой директории служат два класса: GroupByFolder и GroupByType, для который базовый класс это GroupByStrategy. Первый группирует файлы по папкам, второй по типу файла. Для того чтобы сгруппировать файлы нужно создать экземпляр данного класса и вызвать его метод Explore(), который принимает на входе путь до желаемой директории.

## 5. Тестирование.

Всего получилось 7 вариантов развития событий при сканировании директорий, для каждого заготовлена специальная папка:

test/1/ - пустая папка

test/2/ - не пустая, без подпапок

test/3/ - не пустая, с подпапкой 50/50

test/4/ - пустая, с подпапкой

test/5/ - не пустая, уровень вложенности > 1

test/6/ - не пустая, с пустой подпапкой

test/7/ - не пустая, с подпапкой, имеет файл нулевого размера и очень маленький файл в подпапке

test/X/ - не существующая директория

Для каждого примера исключения обрабатываются отдельно, таким образом при возникновении исключения в первом примере, остальные всё равно отрабатываются.

Если запустить программу, то видим что все вышестоящие примеры отрабатывают как задумано. Среди них можно отметить последний пример X, он генерирует исключение о том, что такого пути нет, но примеры, стоящие после него, продолжают работу.