package com.javarush.task.task31.task3101;  
  
/\*   
Проход по дереву файлов  
1. На вход метода main подаются два параметра.  
Первый — path — путь к директории, второй — resultFileAbsolutePath — имя файла, который будет содержать результат.  
2. Для каждого файла в директории path и в ее всех вложенных поддиректориях выполнить следующее:  
2.1. Если у файла длина в байтах больше 50, то удалить его (используй метод FileUtils.deleteFile).  
2.2. Если у файла длина в байтах НЕ больше 50, то для всех таких файлов:  
2.2.1. Отсортировать их по имени файла в возрастающем порядке, путь не учитывать при сортировке.  
2.2.2. Переименовать resultFileAbsolutePath в ‘allFilesContent.txt‘ (используй метод FileUtils.renameFile).  
2.2.3. В allFilesContent.txt последовательно записать содержимое всех файлов из п. 2.2.1. Тела файлов разделять «n«.  
Все файлы имеют расширение txt.  
\*/  
public class Solution {  
 public static void main(String[] args) {  
  
 }  
  
 public static void deleteFile(File file) {  
 if (!file.delete()) System.*out*.println("Can not delete file with name " + file.getName());  
 }  
}

**package** com.javarush.task.task31.task3102;  
  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.\*;  
**import** java.util.stream.Collectors;  
  
*/\*   
Находим все файлы  
Реализовать логику метода getFileTree, который должен в директории root найти список всех файлов включая вложенные.  
Используй очередь, рекурсию не используй.  
Верни список всех путей к найденным файлам, путь к директориям возвращать не надо.  
Путь должен быть абсолютный.  
  
  
Требования:  
1. Метод getFileTree должен принимать аргументом String root, по которому нужно найти все вложенные файлы.  
2. Метод getFileTree должен возвращать список строк.  
3. Нужно реализовать метод getFileTree: найти все файлы по указанному пути и добавить их в список.  
4. Метод getFileTree должен быть вызван только 1 раз (рекурсию не использовать).  
\*/***public class** Solution {  
  
  
 **public static** List<String> getFileTree(String root) **throws** IOException {  
  
 }  
  
 **public static void** main(String[] args) {  
  
 }  
}

**package** com.javarush.task.task31.task3111;  
  
**import** java.io.IOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.List;  
  
*/\*   
Продвинутый поиск файлов  
Давай реализуем настраиваемый поиск файлов в директории.  
Просмотри интерфейс java.nio.file.FileVisitor и его базовую реализацию java.nio.file.SimpleFileVisitor.  
Во время поиска по дереву файлов с помощью метода Files.walkFileTree(Path start, FileVisitor<? super Path> visitor)  
мы используем объект FileVisitor для выполнения необходимых операций над найденными объектами и.  
  
Наш класс для поиска будет называться SearchFileVisitor и расширять SimpleFileVisitor.  
  
Поиск можно будет выполнять по таким критериям:  
— выражение, встречающееся в названии файла (String);  
— выражение, встречающееся в содержимом файла (String);  
— максимальный и минимальный размер файла (int).  
Можно задавать либо один, либо сразу несколько критериев для поиска.  
  
Я в main написал код, который использует готовый SearchFileVisitor для поиска файлов, тебе осталась совсем легкая задача — выполнить его реализацию.  
Подсказка 1: методы get… , set… — это геттеры и сеттеры полей. Основная логика класса по поиску выполняется в методе visitFile(Path file, BasicFileAttributes attrs).  
Подсказка 2: для работы с файлами используй только классы из пакета java.nio.  
  
  
Требования:  
1. В классе SearchFileVisitor нужно создать поля partOfName, partOfContent, minSize, maxSize и сеттеры для них.  
2. В классе SearchFileVisitor нужно создать поле foundFiles и геттер для него. Поле должно быть сразу инициализировано.  
3. Если в SearchFileVisitor задан критерий поиска partOfName, метод visitFile должен добавить файл в foundFiles, если в названии содержится строка partOfName.  
4. Если в SearchFileVisitor задан критерий поиска partOfContent, метод visitFile должен добавить файл в foundFiles, если в содержимом встречается строка partOfContent.  
5. Если в SearchFileVisitor задан критерий поиска maxSize, метод visitFile должен добавить файл в foundFiles, если размер файла меньше maxSize.  
6. Если в SearchFileVisitor задан критерий поиска minSize, метод visitFile должен добавить файл в foundFiles, если размер файла больше maxSize.  
7. Метод visitFile должен быть реализован так, чтобы учитывать сразу несколько критериев поиска.  
\*/***public class** Solution {  
  
 **public static void** main(String[] args) **throws** IOException {  
 SearchFileVisitor searchFileVisitor = **new** SearchFileVisitor();  
  
 searchFileVisitor.setPartOfName(**"amigo"**);  
 searchFileVisitor.setPartOfContent(**"programmer"**);  
 searchFileVisitor.setMinSize(500);  
 searchFileVisitor.setMaxSize(10000);  
  
 Files.*walkFileTree*(Paths.*get*(**"D:/SecretFolder"**), searchFileVisitor);  
  
 List<Path> foundFiles = searchFileVisitor.getFoundFiles();  
 **for** (Path file : foundFiles) {  
 System.***out***.println(file);  
 }  
 }  
  
}

**package** com.javarush.task.task31.task3111;  
  
**import** java.io.IOException;  
**import** java.nio.file.FileVisitResult;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.SimpleFileVisitor;  
**import** java.nio.file.attribute.BasicFileAttributes;  
  
**public class** SearchFileVisitor **extends** SimpleFileVisitor<Path> {  
  
 @Override  
 **public** FileVisitResult visitFile(Path file, BasicFileAttributes attrs) **throws** IOException {  
 **byte**[] content = Files.*readAllBytes*(file); *// размер файла: content.length* **return super**.visitFile(file, attrs);  
 }  
}

**package** com.javarush.task.task31.task3103;  
  
**import** java.io.IOException;  
**import** java.nio.charset.Charset;  
**import** java.nio.file.Files;  
**import** java.nio.file.Paths;  
**import** java.util.List;  
  
*/\*   
Своя реализация  
Реализуй логику методов:  
1. readBytes — должен возвращать все байты файла fileName.  
2. readLines — должен возвращать все строки файла fileName. Используй кодировку по умолчанию.  
3. writeBytes — должен записывать массив bytes в файл fileName.  
4. copy — должен копировать файл resourceFileName на место destinationFileName.  
  
ГЛАВНОЕ УСЛОВИЕ:  
Никаких других импортов!  
  
  
Требования:  
1. Импорты в классе Solution менять нельзя.  
2. Метод readBytes должен возвращать все байты файла fileName.  
3. Метод readLines должен возвращать все строки файла fileName с кодировкой по умолчанию.  
4. Метод writeBytes должен записывать массив bytes в файл fileName.  
5. Метод copy должен копировать файл resourceFileName на место destinationFileName.  
\*/***public class** Solution {  
 **public static byte**[] readBytes(String fileName) **throws** IOException {  
 **return null**;  
 }  
  
 **public static** List<String> readLines(String fileName) **throws** IOException {  
 **return null**;  
 }  
  
 **public static void** writeBytes(String fileName, **byte**[] bytes) **throws** IOException {  
 }  
  
 **public static void** copy(String resourceFileName, String destinationFileName) **throws** IOException {  
 }  
}

**package** com.javarush.task.task31.task3104;  
  
**import** java.io.File;  
**import** java.io.IOException;  
**import** java.nio.file.\*;  
**import** java.nio.file.attribute.BasicFileAttributes;  
**import** java.util.ArrayList;  
**import** java.util.EnumSet;  
**import** java.util.List;  
  
*/\*   
Поиск скрытых файлов  
В классе Solution переопредели логику двух методов:  
— visitFile кроме своей логики должен добавлять в archived все пути к zip и rar файлам  
— visitFileFailed должен добавлять в failed все пути к недоступным файлам и возвращать SKIP\_SUBTREE  
  
Пример вывода:  
D:/mydir/BCD.zip  
  
Метод main не участвует в тестировании  
  
  
Требования:  
1. В классе Solution нужно переопределить метод visitFile.  
2. Метод visitFile, кроме своей логики, должен добавлять в поле archived все пути к zip и rar файлам.  
3. В классе Solution нужно переопределить метод visitFileFailed.  
4. Метод visitFileFailed должен добавлять в поле failed все пути к недоступным файлам и возвращать SKIP\_SUBTREE.  
\*/***public class** Solution **extends** SimpleFileVisitor<Path> {  
 **public static void** main(String[] args) **throws** IOException {  
 EnumSet<FileVisitOption> options = EnumSet.*of*(FileVisitOption.***FOLLOW\_LINKS***);  
 **final** Solution solution = **new** Solution();  
 Files.*walkFileTree*(Paths.*get*(**"D:/"**), options, 20, solution);  
  
 List<String> result = solution.getArchived();  
 System.***out***.println(**"All archived files:"**);  
 **for** (String path : result) {  
 System.***out***.println(**"\t"** + path);  
 }  
  
 List<String> failed = solution.getFailed();  
 System.***out***.println(**"All failed files:"**);  
 **for** (String path : failed) {  
 System.***out***.println(**"\t"** + path);  
 }  
 }  
  
 **private** List<String> **archived** = **new** ArrayList<>();  
 **private** List<String> **failed** = **new** ArrayList<>();  
  
 **public** List<String> getArchived() {  
 **return archived**;  
 }  
  
 **public** List<String> getFailed() {  
 **return failed**;  
 }  
}

**package** com.javarush.task.task31.task3112;  
  
**import** java.io.IOException;  
**import** java.net.URL;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
  
*/\*   
Загрузчик файлов  
Реализуй метод downloadFile(String urlString, Path downloadDirectory), на вход которого подается ссылка для скачивания файла и папка, куда нужно закачать файл.  
Все ссылки имеют вид:  
https://yastatic.net/morda-logo/i/citylogos/yandex19-logo-ru.png  
http://toogle.com/files/rules.txt  
https://pacemook.com/photos/image1.jpg  
  
Метод должен создать объект URL и загрузить содержимое файла на локальный диск.  
Выкачивай сначала во временную директорию, чтобы в случае неуспешной загрузки в твоей директории не оставались недокачанные файлы.  
Затем перемести файл в пользовательскую директорию. Имя для файла возьми из ссылки.  
Используй только классы и методы из пакета java.nio.  
  
  
Требования:  
1. Метод downloadFile должен создавать объект URL для переданной ссылки.  
2. Метод downloadFile должен создать временный файл с помощью метода Files.createTempFile.  
3. Метод downloadFile должен скачать файл по ссылке во временный файл, используя метод Files.copy.  
4. Метод downloadFile должен переместить файл из временной директории в пользовательскую, используя метод Files.move.  
5. Имя сохраненного файла должно быть таким же, как в URL-ссылке.  
\*/***public class** Solution {  
  
 **public static void** main(String[] args) **throws** IOException {  
 Path passwords = *downloadFile*(**"https://www.amigo.com/ship/secretPassword.txt"**, Paths.*get*(**"D:/MyDownloads"**));  
  
 **for** (String line : Files.*readAllLines*(passwords)) {  
 System.***out***.println(line);  
 }  
 }  
  
 **public static** Path downloadFile(String urlString, Path downloadDirectory) **throws** IOException {  
 *// implement this method* }  
}

**package** com.javarush.task.task31.task3113;  
  
**import** java.io.IOException;  
  
*/\*   
Что внутри папки?  
Напиши программу, которая будет считать подробную информацию о папке и выводить ее на консоль.  
  
Первым делом считай путь к папке с консоли.  
Если введенный путь не является директорией — выведи «[полный путь] — не папка» и заверши работу.  
Затем посчитай и выведи следующую информацию:  
  
Всего папок — [количество папок в директории]  
Всего файлов — [количество файлов в директории и поддиректориях]  
Общий размер — [общее количество байт, которое хранится в директории]  
  
Используй только классы и методы из пакета java.nio.  
  
Квадратные скобки [ ] выводить на экран не нужно.  
  
  
Требования:  
1. Метод main должен считывать путь к папке с консоли.  
2. Если введенный путь не является директорией - нужно вывести "[полный путь] - не папка" и завершить работу.  
3. На консоль должна быть выведена следующая информация: "Всего папок - [количество папок в директории]".  
4. На консоль должна быть выведена следующая информация: "Всего файлов - [количество файлов в директории и поддиректориях]".  
5. На консоль должна быть выведена следующая информация: "Общий размер - [общее количество байт, которое хранится в директории]".  
\*/***public class** Solution {  
  
 **public static void** main(String[] args) **throws** IOException {  
 }  
}

**Построй дерево(1)**

Амиго, похоже ты уже достаточно окреп. Самое время проверить свои навыки в большой задаче!  
Сегодня реализуем свое дерево немного нестандартным способом(на базе AbstractList).  
Вводную информацию можешь получить используя свой любимый поисковик и текст ниже.

Элементы дерева должны следовать так как показано на картинке:  
*http://info.javarush.ru/uploads/images/00/04/89/2014/03/21/ee9a9b.jpg*

Для начала сделаем наше дерево потомком класса AbstractList с параметром типа String, а также  
реализуем интерфейсы Cloneable и Serializable.

Реализацию методов get(int index) и size() пока оставь стандартной.

Требования:  
1. Класс CustomTree должен поддерживать интерфейс Cloneable.  
2. Класс CustomTree должен поддерживать интерфейс Serializable.  
3. Класс CustomTree должен быть потомком класса AbstractList.

**Построй дерево(2)**

Несмотря на то что наше дерево является потомком класса AbstractList, это не список в привычном понимании.  
В частности нам недоступны принимающие в качестве параметра индекс элемента.  
Такие методы необходимо переопределить и бросить новое исключение типа UnsupportedOperationException.

Вот их список:  
public String get(int index)  
public String set(int index, String element)  
public void add(int index, String element)  
public String remove(int index)  
public List<String> subList(int fromIndex, int toIndex)  
protected void removeRange(int fromIndex, int toIndex)  
public boolean addAll(int index, Collection<? extends String> c)

Требования:  
1. При попытке вызвать метод get(int index) должно возникать исключение типа UnsupportedOperationException.  
2. При попытке вызвать метод set(int index, String element) должно возникать исключение типа UnsupportedOperationException.  
3. При попытке вызвать метод add(int index, String element) должно возникать исключение типа UnsupportedOperationException.  
4. При попытке вызвать метод String remove(int index) должно возникать исключение типа UnsupportedOperationException.  
5. При попытке вызвать метод subList(int fromIndex, int toIndex) должно возникать исключение типа UnsupportedOperationException.  
6. При попытке вызвать метод removeRange(int fromIndex, int toIndex) должно возникать исключение типа UnsupportedOperationException.  
7. При попытке вызвать метод addAll(int index, Collection c) должно возникать исключение типа UnsupportedOperationException.

**Построй дерево(3)**

Класс описывающий дерево мы создали, теперь нужен класс описывающий тип элементов дерева:  
1.  В классе CustomTree создай вложенный статический параметризированный класс Entry<T> с модификатором доступа по умолчанию.  
2. Обеспечь поддержку этим классом интерфейса Serializable.  
3. Создай такие поля (*модификатор доступа по умолчанию*):  
— String elementName;  
— int lineNumber;  
— boolean availableToAddLeftChildren, availableToAddRightChildren;  
— Entry<T> parent, leftChild, rightChild;  
4. Реализуй публичный конструктор с одним параметром типа String:  
— установи поле elementName равным полученному параметру;  
— установи поле availableToAddLeftChildren равным true;  
— установи поле availableToAddRightChildren равным true;  
5. Реализуй метод void checkChildren, устанавливающий поле availableToAddLeftChildren в false, если leftChild не равен null и availableToAddRightChildren в false, если rightChild не равен null.  
6. Реализуй метод boolean isAvailableToAddChildren, возвращающий дизъюнкцию полей availableToAddLeftChildren и availableToAddRightChildren.

Любое дерево начинается с корня, поэтому не забудь в класс CustomTree добавить поле root типа Entry<String> c модификатором доступа по умолчанию.

Требования:  
1. Класс CustomTree.Entry должен быть объявлен с модификатором доступа по умолчанию.  
2. Класс CustomTree.Entry должен поддерживать интерфейс Serializable.  
3. В классе CustomTree.Entry должно существовать поле elementName типа String.  
4. В классе CustomTree.Entry должно существовать поле lineNumber типа int.  
5. В классе CustomTree.Entry должно существовать поле availableToAddLeftChildren типа boolean.  
6. В классе CustomTree.Entry должно существовать поле availableToAddRightChildren типа boolean.  
7. В классе CustomTree.Entry должно существовать поле parent типа Entry.  
8. В классе CustomTree.Entry должно существовать поле leftChild типа Entry.  
9. В классе CustomTree.Entry должно существовать поле rightChild типа Entry.  
10. В классе CustomTree.Entry должен быть корректно реализован конструктор с одним параметром типа String (смотри условие).  
11. В классе CustomTree.Entry должен корректно реализован метод checkChildren (смотри условие).  
12. В классе CustomTree.Entry должен корректно реализован метод isAvailableToAddChildren (смотри условие).

### Построй дерево(4)

Итак, основа дерева создана, пора тебе поработать немного самому.  
Вспомним как должно выглядеть наше дерево.

**Элементы дерева должны следовать так как показано на картинке:**  
http://info.javarush.ru/uploads/images/00/04/89/2014/03/21/ee9a9b.jpg

Необходимо написать методы, которые бы позволили создать такую структуру дерева и проводить операции над ней.

**Тебе необходимо реализовать:**  
1. метод add(String **s**) — добавляет элементы дерева, в качестве параметра принимает имя элемента (elementName), искать место для вставки начинаем слева направо.  
2. метод remove(Object **o**) — удаляет элемент дерева имя которого было полученного в качестве параметра.  
3. метод size() — возвращает текущее количество элементов в дереве.  
4. метод getParent(String **s**) — возвращает имя родителя элемента дерева, имя которого было полученного в качестве параметра.

Требования:  
1. После добавления N элементов в дерево с помощью метода add, метод size должен возвращать N.  
2. После удаления последнего добавленного элемента из дерева с помощью метода remove, метод size должен возвращать N-1.  
3. После удаления второго элемента добавленного в дерево, метод size должен возвращать N/2 + 1 (для случаев где N > 2 и является степенью двойки), N - размер дерева до удаления.  
4. Метод getParent должен возвращать имя родителя для любого элемента дерева.