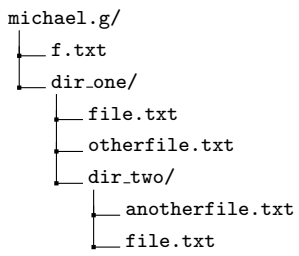
Find

This assignment will emulate the UNIX tool: `find`. `find` provides lots of functionality, but primarily we will be concerned with the search by filename. You will design a class called `Find` which will mimic this particular function. It will be given a filename and a particular starting point in a directory tree and it will search through the tree for any files with the given name. Your `Find` class will be packaged in the `cli` package from the previous assignment and make use of your `FileList` class, so it is important that you complete the previous assignment before beginning this one.

Commmand Line Behavior

Your `Find` class should implement a `main()` method, so that it can be run from the command line. If no arguments are given, then it simply displays the entire directory tree starting from the current directory. All displayed names will be shown in canonical form. This last part is slightly different than the UNIX `find` command. In the traditional `find` the filenames are displayed with names relative to a search point as opposed to being in canonical form – we will not emulate this aspect of `find`. To see this difference, you can compare the system `find` to the Java version that I have written as an example.

Consider the following directory structure:



Assuming that we are located in the directory: `michael.g/`, running the `Find` class from the command line with no arguments yields the following output:

```
[user@notnotbc]$ pwd
/CS3120/MW2/michael.g
[user@notnotbc]$ java cli.Find
/CS3120/MW2/michael.g/f.txt
/CS3120/MW2/michael.g/dir_one/
/CS3120/MW2/michael.g/dir_one/file.txt
/CS3120/MW2/michael.g/dir_one/otherfile.txt
/CS3120/MW2/michael.g/dir_one/dir_two/
/CS3120/MW2/michael.g/dir_one/dir_two/anotherfile.txt
/CS3120/MW2/michael.g/dir_one/dir_two/file.txt
[user@notnotbc]$
```

If an argument is given, then `Find` uses that as the root of the directory tree through which to search. For instance, if the directory `dir_two/` is given as the first argument, then all the files in that directory tree are displayed:

```
[user@notnotbc]$ pwd
/CS3120/MW2/michael.g
[user@notnotbc]$ java cli.Find dir_one/dir_two/
/CS3120/MW2/michael.g/dir_one/dir_two/anotherfile.txt
/CS3120/MW2/michael.g/dir_one/dir_two/file.txt
[user@notnotbc]$
```

Furthermore, your `Find` class should account for a single command-line option:

```
-name  :  proceeding argument specifies the name to match
```

The argument given after `-name` will be used as the name for which `Find` is searching. Only files which match the string given after the `-name` option will be displayed in the output. For instance, consider the scenario in which we are searching for all files named `file.txt` in the directory tree with root at `michael.g/`:

```
[user@notnotbc]$ pwd
/CS3120/MW2/michael.g
[user@notnotbc]$ java cli.Find -name file.txt
/CS3120/MW2/michael.g/dir_one/file.txt
/CS3120/MW2/michael.g/dir_one/dir_two/file.txt
[user@notnotbc]$
```

Notice that all files named `file.txt` are displayed. This feature can also be used in conjunction with the previous one – i.e. you may specify the root of a directory tree (as opposed to using the current directory).

```
[user@notnotbc]$ pwd
/CS3120/MW2/michael.g
[user@notnotbc]$ java cli.Find dir-one/dir_two/ -name file.txt
/CS3120/MW2/michael.g/dir_one/dir_two/file.txt
[user@notnotbc]$
```

Again, all files named `file.txt` are displayed; however, this time, we have started the search deeper down the directory tree so we do not see the `file.txt` which is located in `dir_one/`.

If no files match the specified name, then nothing is displayed. For instance, consider the following search:

```
[user@notnotbc]$ pwd
/CS3120/MW2/michael.g
[user@notnotbc]$ java cli.Find -name hello
[user@notnotbc]$
```

Finally, if no argument proceeds `-name` then an error message should be displayed stating so. The same should be done if an unrecognized option is given:

```
[user@notnotbc]$ java cli.Find -name
Find: missing argument to -name
[user@notnotbc]$ java cli.Find -r
Find: Error: unrecognized option: -r
[user@notnotbc]$
```

Minimum Requirements

The class should contain: a single `private` constructor, a `public static` method `in()` that returns a `FileList` object and a `main()` method. It must also be packaged in: `cli`. This means you should use the same directory in your `submissions/` directory that you created for `FileList`.

Below is the skeleton of the `Find` class to which your code must conform. You are free to add any helper functions as you wish, *but they must be* `private`. Remember that any `public` methods are part of the API in virtue of being `public`.

```
package cli;

public class Find {
    // constructor
    private Find();

    // methods
    public static FileList in(String path, String name)
                                throws FileNotFoundException, SecurityException;

    public static void main(String[]);
}
```

Find API

```
public static FileList in(String path, String name)
   throws FileNotFoundException, SecurityException
```

Not quite a static factory method, but something similar. It is given a path and a name and it returns a `FileList` object containing all the files in the directory tree rooted at `path` which match the string given by `name`. If `name` equals the empty string – i.e. `name.equals("") == true` – then all files in the directory tree rooted at `path` are put into the `FileList` object. This is the equivalent of running `Find` from the command line without the `-name` option (see section above).

This method should not explicitly throw any exceptions. Exceptions should only be a by product of creating a `FileList` object. Keep in mind that the `FileNotFoundException` should only ever happen if the path given does not exist. The `SecurityException` may happen at any point during the descent of the directory tree. You may simply catch the `SecurityException`s and ignore them, if you like. In other words, a `FileNotFoundException` should cause your program to halt, while a `SecurityException` should not.

```
public static void main(String [] args)
```

Parses command line arguments according to the guidelines given in the previous sections. Usage is as follows:

```
java Find [path] [-name pattern]
```

This means `Find` may be initiatied with any of the following conditions:

(1) no path, no pattern (contents in dir. tree rooted at current dir.)
(2) path, no pattern (contents in dir. tree rooted at given path)
(3) no path, pattern (contents in dir. tree rooted at current dir. that match given pattern)
(4) options, arguments (contents in dir. tree rooted at given path that match given pattern)

Your code may ignore any extraneous text.

After parsing the command line arguments, `main()` should create a `FileList` object using the `in()` method. It should then pass this `FileList` object to the `format()` method which should return a `List<String>`. This `List<String>` may be printed one at a time to the console, or joined with `String.join()` and then printed all at once.

Extra Credit

You may earn extra credit for any of the following improvements on your `FileList` class:

(1) Implement a fuzzy finding feature as was described in the extra credit section for the `FileList` class. This would only apply to the pattern associated with the `-name` option:

```
java Find -name *.txt
java Find -name file.???
```

(2) Display messages embedded into the output that inform the user of directories that were not searched because of permissions errors for that particular directory. This behavior is exhibited in the traditional `find` command which is available on the class server.

(3) Implement another command line option: `-prune`. This option is immediately proceeded by an argument similar to that which proceeds `-name`; however, the argument proceeding `-prune` is a pattern which specifies directories which should not be searched. For instance, consider the following command:

```
java Find -name Foo -prune Bar
```

The command above would search through the directory tree rooted at the current working directory for any files called `Foo` and would not search any through any directories called `Bar`.

Remember that the name of the file must be the same as the name of the class with `.java` appended to it. In other words, your file should be named `Find.java`.

Since it is packaged with the `cli` package, it must be in a directory named: `cli`. Submit your assignment by copying a directory named: `cli` along with the `Find.java` file into your personal submissions directory. There is no need to place the `Find.class` file in the directory.