Introduction to Comp. Sci., Homework 10: Globs

Due at 10pm on Monday, May 13

Readings from Liang

Read chapter 20, "Recursion."

Optional readings and exercises from HFJ

Read chapter 15 of Head First Java, "Networking and Threads," and do the exercises.

RecursiveFinder.globMatches

Last homework, you wrote some code that matched strings like `foobar` with simple "glob" patterns like `foo*` and `*bar`. In this homework, you'll extend this scheme to support patterns with `*` in arbitrary places.

This may be easier to understand with examples than with prose. See the positive examples `match01` through `match16` and the negative examples `noMatch1` through `noMatch8` in `Hw10Test.java`. This will also be discussed in lecture.

Suppose you want to match a string `s` against a pattern, `pattern`, and `pattern` has no `*` characters. If both `s` and `pattern` are the empty string, then they match. If one is empty and the other is not, or if their first characters differ, then they do not match. Otherwise, they match if and only if the characters after the first character match. Write a matching method that does this as a recursive method `globMatches` in a class named `hw10.RecursiveFinder`: A correct solution will not involve any `for` or `while` loops.

```
package hw10;
public class RecursiveFinder {
  public static boolean globMatches(String pattern, String s) {
    ...
  }
}
```

Now, suppose `pattern` might have `*` characters in it. If both `pattern` and `s` are empty, then they match. If `pattern` does not begin with a `*`, the situation is the same as above. If `pattern` begins with a `*`, the `*` might match no characters of `s`, or it might match one or more. In the version of `globMatches` described above, you always removed the first character from both string and pattern when you made a recursive call. In this step, when the pattern begins with a `*`, you want to remove either the first character of the string (`*` matched its first character and can match more) or the first character of the pattern (`*` matched no more characters of the pattern, move on to the next character). Modify your

implementation of `globMatches` to do this, so that it can match patterns with `*` characters anywhere in the pattern.

`globMatches` should not have any loops. It should call itself recursively, once if `pattern` does not begin with a `*` and twice if it does. In the latter case, if either recursive sub-match returns true, the call as a whole should return true.

RecursiveFinder.findMatches

Add a version of `FileFinder.findMatches` from homework 9 to `RecursiveFinder`. It should have the same arguments and return value as the one from homework 9. Its implementation should have two differences:

- It should find files inside subdirectories of the file identified by its `directory` argument. To accomplish this, it should call itself recursively. `File` objects returned by this version of `findMatches` should all refer to regular files, not directories, even if the directory names match `pattern`. It should throw a `FileNotFoundException` if any of the directories inside `directory` are not readable.
- It should call `globMatches` instead of `patternMatches`, so that a call like this:

  ```
  ArrayList<File> files = findMatches(new File("dir"), "A*B");
  ```

  will return a list of all regular files that begin with the letter "A" and end with the letter "B."

Nowhere in your code should you declare a static variable. If you prefer, you can base your implementation of `findMatches` on the solution to homework 9 instead of your own code. As usual, there are automated tests, `Hw10Test`.