P07 File Finder

Programming II (CS300) Spring 2020
Department of Computer Sciences
University of Wisconsin-Madison

Due: 9:59PM on March 25, 2020
Pair Programming **is NOT Allowed.**

Overview and Learning Objectives:

Through this assignment you will build a utility application that allows you to search through your file system for files containing a specific substring within their relative path name.

This assignment will give you practice working with iterators that can be used to step through the contents of a linear array, or through files organized into a directory tree. Independent of how this data is stored, the iterator provides a common interface for stepping through these different kinds of collections of data.

Grading Rubric:

5 points	**Pre-Assignment Quiz:** You should not have access to this write-up until after you have completed the corresponding pre-assignment quiz through Canvas.
16 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is correct. To become more confident in this, you must write and run additional tests of your own.
24 points	**Manual Grading and Supplemental Automated Tests:** When your final grade feedback appears on Gradescope, it will include the feedback from these additional automated grading tests, as well as feedback from human graders who review the code in your submission by hand.
50 points	**TOTAL**

Requirements:

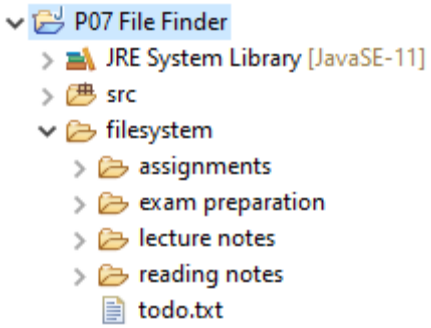
The only types outside of java.lang that can be used for this assignment are:

- java.util.Iterator
- java.io.File
- java.io.FileNotFoundException
- java.util.NoSuchElementException
- java.util.Arrays (only to call Arrays.sort)

The appendix includes links to some of the methods within these classes that are likely to be helpful while implementing this assignment.

0. Project Files and Setup

Create a new Java Project in eclipse with a suitable name, like P07 File Finder. Since this project involves stepping through folders and files within a filesystem, we are providing the following p07filesystem.zip to practice with and to demonstrate the expected behavior for this assignment. Download and extract the contents of this zip file into your project folder. The resulting hierarchy of folders should look as follows:



1. The ShallowFileIterator

Create a new public class called ShallowFileIterator, which implements the java.util.Iterator interface. The constructor for this class should take a java.io.File reference as input, which refers to a directory that exists within your filesystem. Each time the next method is called on this iterator, it will return a reference to a different java.io.File that is contained within the provided directory. After all contained files have been returned in this way, the iterators hasNext() method should begin to return false and the next() method should being to throw a NoSuchElementException with a descriptive message.

The only fields that you are allowed to define within this class are the following private instance fields:

- folderContents – a sorted array of File references which this iterator steps through: containing references to all files in the specified directory (see Appendix for File.listFiles() reference, and for the Arrays.sort() method that can be used to sort them)
- nextIndex – the int index specifying the next File within folderContents that this iterator's next() method will return

When the argument passed to your ShallowFileIterator's constructor does not exist within the local filesystem, then this constructor should throw an FileNotFoundException with a descriptive message about the problem encountered.

This class must define an implementation for the methods within the java.util.Iterator interface: both next() and hasNext(). Note that the two default methods listed in these JavaDocs can be ignored and do not need to be implemented for any part of this assignment.

2. Testing the ShallowFileIterator

Create a new public class called P07Tester. This class should define a main method that calls each of the other methods in this file, and prints out whether each test returns true or false. These other methods should all be public static methods that return a boolean value and take a java.io.File reference to the provided filesystem directory as their only argument. Since the location of this filesystem may be

different on gradescope than on your local machine, ensure that your tests do not rely on the absolute path of this folder. One way to test this, is to copy your filesystem folder to another location on your computer, and attempt to pass a reference to that alternate location to each of your test methods.

The first required test method for this assignment is named testShallowFileIterator. As stated above, this should be a public static method that returns a boolean and takes a File argument as input. This specific test method should create a new ShallowFileIterator using a reference to the provided filesystem directory. It should then repeatedly call next, and collect the names of the files returned into a single comma-separated String. The contents of this string should match the following:

```
String expectedResults = "assignments, exam preparation, lecture notes, "
    + "reading notes, todo.txt, ";
```

Notice that this example left an extra comma at the end of the list. This is not required, but does make the test a bit simpler to implement (since there is a comma after every filename, and no special case for omitting or removing the final one).

3. Testing the DeepFileIterator

Before defining our next class: DeepFileIterator, let's write a test for it. DeepFileIterator is used in a similar fashion to the ShallowFileIterator, but additionally returns the contents of directories, and of directories within directories, etc. For this reason, our test code will look very similar for these two iterators.

Start by copy-pasting your ShallowFileIterator code into a new public static method called testDeepFileIterator, which also returns a boolean and takes a filesystem File reference argument. Then make the following changes to this test code:

- Each of these test methods receives a reference to the filesystem folder, but we would like this specific test to focus on the assignments folder within this filesystem. So we'll add the following first line to update this folder reference as follows (assuming that folder is the name of the java.io.File reference parameter that your test method takes as input):

```
folder = new File(folder.getPath() + File.separator + "assignments");
```

- Next we'll update the expectedResults to account for not only the directories directly within this assignments folder, but to also account for their contents:

```
String expectedResults = "P01, PiggyBank.java, P02, CalendarPrinter.java, P03, "
    + "ElasticBank.java, P04, ExceptionalPiggyBank.java, P05, ExtendedVersion, "
    + "WinterCarnival.java, WinterCarnival.java, P06, AlphabetTrain.java, ";
```

- Finally, change the type references in your code from "ShallowFileIterator" to "DeepFileIterator" (without the quotes). These changes won't compile until you implement the DeepFileIterator in the next step.

4. The DeepFileIterator

Create a new public class called DeepFileIterator, which implements the java.util.Iterator interface. This class should have one constructor with the same argument and exception conditions as the

ShallowFileIterator that you implemented in Step 1. As described in step 3, this class will iterate through the contents of the specified directory, and will also iterate through the contents of any directories contained within that directory (no matter how deeply nested those contained folders might be).

The only fields that you are allowed to define within this class are the following private instance fields:

- folderContents – a sorted array of File references which this iterator steps through
- nextIndex – the int index specifying the next File within folderContents that this iterator's next() method will return
- contentsIterator – a DeepFileIterator reference that is used to step through the contents within any directory that is contained within this folder

Implementation Tips: Whenever your next() method returns a File reference to a directory, it should update the contentsIterator field to be a new DeepFileIterator that steps through the contents of that directory. Subsequent calls of this same next method should then return the File references returned from this contentsIterator, until it becomes exhausted (its hasNext() returns false). After it is exhausted, the next method can proceed to return any remaining files from the folderContents (along with the contents of any of those files that happen to be directories). Creating new DeepFileIterator objects in this way should not typically result in FileNotFoundExceptions being thrown. But since you'll need to catch this checked exception, set contentsIterator back to null to handle such exceptions.

As with the ShallowFileIterator, the DeepFileIterator's next() method should begin to throw a NoSuchElementException when it's hasNext() method begins to return false.

5. Testing the FilteredFileIterator

Before defining our last class: FilteredFileIterator, let's write a test for it. FilteredFileIterator steps "deeply" through the contents of folders within folders like our DeepFileIterator does. But this iterator's next method only returns files with names that contain a specified search pattern. Make a copy of your testDeepFileIterator method, and rename this copy testFilteredFileIterator. Then make these additional changes to your test:

- For this test we want the entire filesystem folder's contents to be traversed, so remove the first line that changes the folder reference from this new test method.
- Next we'll be searching only for files that include the ".java" extension within their name, so the expectedResults for this search should be updated to the following:

```
String expectedResults = "PiggyBank.java, CalendarPrinter.java, ElasticBank.java, "
    + "ExceptionalPiggyBank.java, WinterCarnival.java, WinterCarnival.java, "
    + "AlphabetTrain.java, codeSamples.java, ";
```

- Finally, change the type references in your code from "DeepFileIterator" to "FilteredFileIterator" (without the quotes). Since the FilteredFileIterator constructor takes an extra argument: after the filesystem File reference, pass the search string ".java" as the second argument to this constructor. These changes won't compile until you implement the FilteredFileIterator in the next step.

6. The FilteredFileIterator

The last public class that you will implement for this assignment is a FilteredFileIterator, which implements the java.util.Iterator interface. The only public constructor for this class takes two arguments: first) a java.io.File reference to the directory that it iterates through, and second) a String searchPattern that is used to filter the results that are returned from this iterator. If the specified file does not exist within the local filesystem, then this method should throw a FileNotFoundException (similar to what was done in both the ShallowFileIterator and DeepFileIterator classes).

The only fields that you are allowed to define within this class are the following private instance fields:

- fileIterator – a DeepFileIterator that steps through all files within the initial directory (and all contained sub directories).
- searchPattern – a String that must be part of a file's name in order for that file to be returned from this iterator's next method
- nextMatchingFile – a File reference to the next file that this iterator will return (or null, if there are no more files containing this string in their file name left to be returned). Note that this iterator must look ahead for such files, in order for it's hasNext() method to reliably indicate whether another matching file can be returned or not.

Implementation Tips: It may be helpful (not required) to implement a private helper method within this class. This method can repeatedly call next on the fileIterator, until it returns a file that contains the specified searchPattern. This file can then be stored within nextMatchingFile. If the fileIterator is exhausted before a matching file can be found, then null can be stored in nextMatchingFile instead.

As with the ShallowFileIterator and the DeepFileIterator's next() method, this classes next() method should begin to throw a NoSuchElementException when it's hasNext() method begins to return false.

7. Assignment Submission

Congratulations on completing this CS300 assignment! After reviewing your code and ensuring that it conforms to the requirements of the course style guide, submit all four files to gradescope: ShallowFileIterator.java, DeepFileIterator.java, FilteredFileIterator.java, and P07Tester.java. Your score for this assignment is based on your submission marked "active" and made prior to the hard deadline.

Appendix:

Helpful java.io.File Methods for this Assignment (with links to Java API)

- File.exists()
- File.isDirectory()
- File.getName()
- File.listFiles()

Helpful java.util.Iterator Methods for this Assignment (with links to Java API)

- **Arrays.sort()**

Helpful java.util.Iterator Methods for this Assignment (with links to Java API)

- Iterator.next()
- Iterator.hasNext()