CSC 385

Module 2 Homework: Enumerating a File System Using Recursion

30 pts.

Introduction:

For this programming assignment you will write a recursive algorithm to display the contents of a directory in a computer's file system. You will also get a little experience using the API documentation for several components of the JDK library, including the `File` class. Additionally, you will get some exposure to a graphical user interface (GUI), which you may or may not have had in your previous Java work.

The Explorer program that serves as the default file system browser for the Windows XP operating system allows you to see the first-level contents of a given folder. It does not allow you to see the contents of any nested subfolders—you have to click on each subfolder, one at a time, to see the contents of these folders. Suppose you wanted to see the entire contents of a given folder; i.e., not just the first-level files and subfolders, but everything that is contained in all the nested subfolders, regardless of how many nested subfolders there are.

You could write an algorithm that iterates through all the subfolders using a loop, but since you have know way of knowing beforehand how many levels of nested subfolders there are, implementing an iterative algorithm to do this would be tricky. This problem is perfectly suited for a recursive algorithm, however. Each subfolder constitutes a problem that is similar to the initial problem (i.e., the root-level folder), only smaller. Recursion can theoretically (though not practically) handle an infinite amount of nesting.

Your goal for this assignment is to implement a recursive method that will enumerate the contents of a chosen folder in a computer's file system, and display the following information about all the files and subfolders in the chosen folder:

1. absolute path name (i.e., the path name beginning with the drive letter or network share; e.g.: "C:\My Documents\myFile.txt")
2. size, in kilobytes
3. type (file or folder)
4. the date and time on which the file or folder was last modified

Download Files

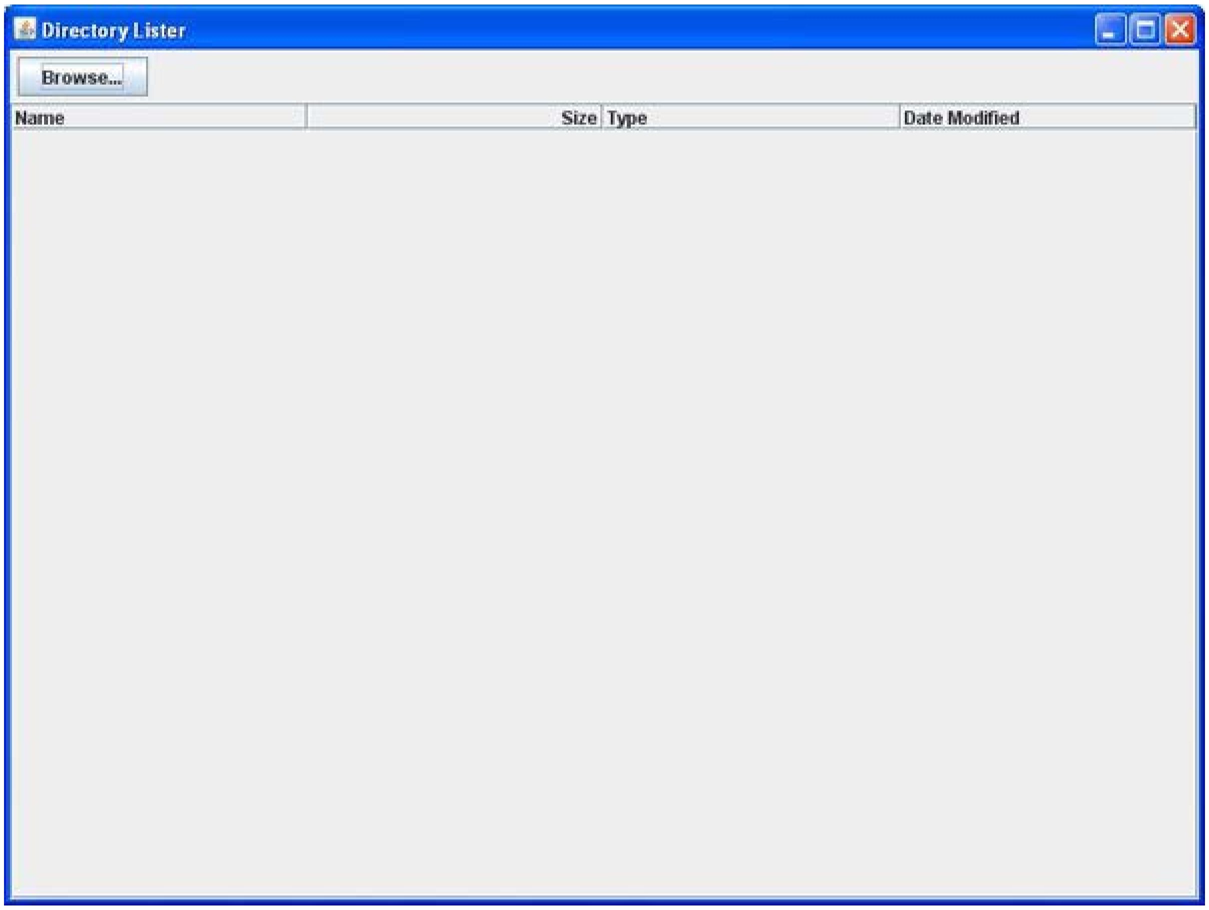
Download the DirectoryLister.zip file. It contains 3 files you will need to use for this assignment:

1. Driver.java
2. GUI.java
3. DirectoryLister.java

The only file you will need to modify for this assignment will be DirectoryLister.java. You will not need to create any additional classes for this assignment. You should be able to compile the project as it is. It will allow you to select a directory, but it won't do anything else.

How the Program Should Work

A screen capture of how the user interface appears when the program is first launched is shown below. This is what you should see if you compile and run the provided source files as they are, with no modifications. Clicking the “Browse...” button will bring up a JFileChooser dialog where you can select a folder from the file system. Once you have selected a folder, a table will be generated showing the name, size, type, and date last modified for every file and folder in the selected folder, as well as all nested subfolders. With the files you download in the directoryLister.zip file, selecting a folder will do nothing. You must complete the implementation to process the selected folder and display all the required information.



What you need to do

All you need to do for this assignment is complete the following 2 methods of the `DirectoryLister` class. The method declarations have been provided for you. All you need to do is provide the implementations. You will need to understand how the GUI works, though in order to get your results to display properly.

```
/**
 *    Show the directory listing.
 *    An error message is displayed if basePath does not represent a valid directory.
 *
 *    @param      basePath    the absolute path of a directory in the file system.
 */
public void showDirectoryContents(String basePath)
{
    // TODO
}
```

```
/**
 *    Recursive method to enumerate the contents of a directory.
 *
 *    @param      f     directory to enumerate
 */
private void enumerateDirectory(File f)
{
    // TODO
}
```

The value returned by the `JFileChooser` dialog is a string representing the absolute path of the selected folder. There are a few exceptional situations you should try to make sure are handled by validating the input returned from the `JFileChooser` dialog:

1. If the user clicks the "Cancel" button or the "Close" button on the `JFileChooser` dialog, without choosing a folder.

2. If the user enters an invalid folder path directly into the textbox on the `JFileChooser` dialog, or modifies a path selected by clicking on a folder in the dialog's file/folder view.

3. It is possible, albeit , for a chosen folder to exist at the time of choosing it in the `JFileChooser` dialog, but not exist after the user clicks the "Open" button on the dialog. An example of how this could happen would be a concurrently running process that happens to delete or rename the chosen folder in between the time the folder was chosen, and the "Open" button is clicked.

You may find that setting up a `try-catch` block to handle these situations is helpful. In any event, the program should not exit or crash when one of these situations occurs, but it should behave appropriately; e.g., by displaying a message using a `JOptionPane` dialog to inform the user what has occurred.

If the chosen directory is valid, the `enumerateDirectory` method should be called, which should recursively enumerate the contents of the chosen folder. To get full credit for this part of the assignment, your `enumerateDirectory` method **must be recursive**, and it should update the display with all the required information. You can use the `updateListing` method of the GUI class to do this. The arguments for this method are 4 references to to `String` objects that represent the absolute path, size, type (folder or file), and the date last modified for the file or folder currently being processed.

Relevant Classes in the Java API

- `File`
- `JOptionPane`

The `File` class is an abstract representation of either a file or a folder in the file system. To create a `File` object, you will probably want to use the constructor, `File (String pathname)`, since this is the type of information returned when the initial folder is chosen from the `JFileChooser` dialog. The `File` class provides many useful methods, including ones to retrieve all the information you need to display for each file or folder. I will leave it to you to read through the API docs to figure out which methods you need.

The `JOptionPane` class is used to display a simple modal dialog box containing a message. It can also display things like warning icons to indicate the seriousness of the message. You don't need to create an instance of the `JOptionPane` class in order to use it. You can use one of the many static methods the `JOptionPane` class provides. For example, the following statement will display an error message that reads, "The specified filename is invalid!":

```
JOptionPane.showMessageDialog(null, "The specified filename is invalid! ", "Error",
                                    JOptionPane.ERROR_MESSAGE);
```

The first argument here refers to the parent component of the `JOptionPane`. Since we're calling one of the static methods, we don't need to specify a parent component, so we pass a null value. The second argument is the error text that will be displayed in the main part of the message dialog. The third argument specifies the text that should appear in the title bar of the dialog. Finally, the fourth argument specifies which type of icon should be displayed along with the error message (warning, information, or error).

Hints

1. Don't make this project harder than it really is! I've given you 2 weeks to finish it, but you don't really have to do that much work—you only need to implement the two methods I've indicated.

2. You should not need to create any additional methods or classes. The user interface is built for you, so you do not have to create any additional UI components. You will need to understand how the user interface works, though, so you can make your implementation communicate with it.

3. Your showDirectoryContents method needs to call your enumerateDirectory method, which requires a File object as a parameter, so at some point you will need to create a File object in your showDirectoryContents method.

4. Your enumerateDirectory method must be recursive, so you will need to determine what the base case(s) are, and what the recursive case(s) are. If you think about how a computer's file system is structured, you should be able to figure this out pretty quickly.

5. Test your program before you submit it! Make sure your test cases cover all the possible scenarios:

   a. a non-existent folder (you can do this by typing an invalid path in the text box of the interface, instead of choosing an existing directory)

   b. an empty folder

   c. a folder containing only one file

   d. a folder containing only one subfolder

   e. a folder containing multiple files (but no subfolders)

   f. a folder containing multiple, non-nested subfolders (but no files)

   g. a folder containing both files and subfolders (all folders are non-nested)

   h. a folder containing multiple, nested subfolders (but no files)

   i. a folder containing an arbitrary mix of files and subfolders (some nested, some non-nested)

6. Don't worry about exhaustively testing your program on a folder containg a huge number of files and subfolders. While recursion is elegant, it can quickly eat up all the stack space in your computer's memory, and if this happens the program will crash. Test your program on folders with moderate numbers of files and subfolders, and assume that if your program works on those folders, it will work for all folders.

7. If you have questions, please ask!

Rubric:

showDirectoryContents method	
creates `File` object	1 pt.
handles errors using `JOptionPane`	1 pt.
calls enumerateDirectory method	3 pts.
enumerateDirectory method	
uses recursion, and recursion works properly	8 pts.
displays absolute paths	2 pts.
displays file/folder sizes	2 pts.
displays type (file or folder)	2 pts.
displays date last modified	2 pts.
Tests	
non-existent directory	1 pt.
empty directory	1 pt.
directory containing only one file	1 pt.
directory containing only one subdirectory	1 pt.
directory containing multiple files (but no subdirectories)	1 pt.
directory containing multiple, non-nested subdirectories (but no files)	1 pt.
directory containing both files and subdirectories (all directories are non-nested)	1 pt.
directory containing multiple, nested subdirectories (but no files)	1 pt.
directory containing an arbitrary mix of files and subdirectories (some nested, some non-nested)	1 pt.
Total:	**30 points**

What to submit:

A .zip file containing your Driver.java, GUI.java and DirectoryLister.java source files. **Do not send me JCreator project or workspace files (the files ending in .jcp, .jcw, and .jcu), or your compiled .class files.**