CIS 41B Assignment 5 - GUI and Threads Part 1: Enhance lab 4

Enhance the file search application of assignment 4 by letting the user find all files in a user defined directory that have a certain filename and contain a certain text string within the file.

The application is made up of 2 files:

* lab5.py: which is an enhanced version of the findwin.py file of assignment 4.
* filesearch.py: a class that has methods to search for files

Enhance and modify the FileSearch class in the filesearch.py file

1. For the constructor:
   * In the os.walk(), rather than storing all files (no directories) in a dictionary, store the files in a list of tuples. Each tuple is (the path from the os.walk, the filename from the os.walk).
2. For the *searchName* method:
   * Change the input arguments. In addition to the regular expression (same as assignment 4), add 2 more input arguments: a search string and a list of results.
   * Import the module strsearch.py (downloaded from module 5). This module has a function called strIsInFile( ) which accepts 2 input arguments: a text string and the name of an existing file. It will return True if the text string is in the file, returning False otherwise.
   * Modify the search for files that are in the list of tuples: append to the list of results (an input argument) all tuples in which: the filename matches the regex and (the search string is empty or strIsInFile( ) returns True)
   * After the search is done, sort the list of results by paths.
   * Since the list of results is passed in as input argument, there is nothing to return from this method.

Enhance and modify the FindWin class in the lab5.py file

1. Add a main at the end of lab5.py, which will instantiate an object of the FindWin class and run its mainloop.
2. For the constructor of the FindWin class

* After the label that prints "Regex Filter" and its corresponding Entry object, add one more row of widgets.   
  This row has a label that prints "Search String" and has a corresponding Entry object.
* This Entry object behaves just like the "Regex Filter" Entry object:
  + Accepts a user input string
  + Binds to the <Return> key with a callback function that is the same search function as the "Regex Filter" Entry object.
* Before the call to the update() method, add a line of code to create a result list and initialize it to an empty list. This result list is the list of filenames that will be shown in the listbox after the search.

1. For the private search method that's the callback for both the "Regex Filter" and "Search String" Entry boxes:

* For the call to searchName() of the FileSearch object, pass in: the user input regex, the user input search string, and the result list.
* After the call to searchName(), call an updateListBox() method.
* Write the updateListBox method:
* Move the if else statement that compares the length of the listbox to 1000 into this method.
* If the number of files is 1000 or larger, show a pop up warning message that there are too many files
* Else put the result list into the lisbox and put the count in the count label

To test your program:

* Comment out the line in the FindWin class that sets the default search directory to the user's home directory. You will eventually put the line back, so don't delete it.
* Change the default search directory to the current directory or any directory where you know there are only your files and directories.
* Because both the "Regex Filter" and the "Search String" Entry objects bind to the <Return> key, it means that the search for matching files will happen when the user hits Enter from either Entry object. It also means that the user can tab or click from one Entry object to another without initiating a search.

Part 2: Add 2 threads to help the GUI become more responsive.

The 2 bottlenecks in the program occur in:

1. The FileSearch object's searchName method. It can take a long time to generate the resulting list of files that match the regex and contain the search string.
2. The FindWin object's updateListBox method. Because the searchName method is slow in sending back results, the updateListBox becomes a blocking method. It can't finish until all results or 1000 file paths are shown, and while it's waiting to display results, the GUI is unresponsive.

To solve bottleneck 1:

* Outline of solution:
  + Instead of calling the searchName method, create a child thread to run searchName.
  + Because searchName runs in a thread, it means the GUI is now responsive to the user: while searchName is still running, the user can enter a new regex or a new search string or change directory.
  + If the user does any of the above, we need to cancel the current search thread and start a new thread with a new search.
* To cancel the current search and start a new one:
  + Select a way for the main thread (the GUI) to communicate with the child search thread, such that the main thread can tell the child thread to stop the search for matching files. Set up this synchronizing mechanism in the main thread and child thread.
  + Create a private cancelSearch method in the FindWin object so that the main thread can:
  + cancel the after() thread (discussed in bottleneck 2 below)
  + tell the child thread to stop the search
  + wait for the child thread to end (*don't forget this part*)
* To start a search thread, the FindWin object's private search method has some additional steps:
  + Call cancelSearch before starting a new thread, if this is not the first time the search runs.
  + Clear the result list by using the list class clear() method. Don't create a new result list.
  + Start the search thread.
* The searchName method runs as described in part 1 in the previous page. It will keep updating the result list that's passed in as an input argument. But there are 2 additions to the code:
  + the synchronizing mechanism between the search thread and the main thread
  + stop the search when 1000 file paths have been found (reach the max limit)

To solve bottleneck 2:

* Outline of solution
  + Use Tkinter's after() to create a thread for the updateListBox method. This means the slow update of the listbox will be run as a thread in the GUI.
  + This thread will keep running every 100ms until there's no more file paths to insert in the listbox. At each 100ms interval that it runs, it will update the listbox with *additional,* *new* file paths that are found by searchName() during the time interval.
* To create a thread to update the listbox with after(), in the updateListBox method:
  + Check to see if there is a search thread that's currently running.
  + If a search thread is running, it means the listbox will need to be updated with data. Therefore, use after() and a delay of 100ms to ask the mainloop() run updateListBox() again. The 100ms delay is a recommended time, it's short enough that the user won't notice that there's a delay in the output, but it's long enough that the GUI can appear to respond quickly to any user request.
  + If there's no search thread running, then it means one of 2 scenarios:
  1. the search is done: display the total number of files found (same as lab4)
  2. there are more than 1000 entries: pop up a warning box that there are too many files to display (same as lab4)
  + Regardless of whether or not there is a current search thread, write code to update the listbox with the *newest* file paths.

Up until now we've put the file paths in the listbox one by one in a loop. To speed up this process, the listbox can accept a list of file paths: listboxObj.insert(tkEND, \*list\_of\_filepaths)  
(Question that could have been on the first midterm: what's the \* for?)

Wrapping up odds and ends of threading

* In the constructor of FindWin, after the set up of the widgets but before self.update(), add this line of code:

self.protocol("WM\_DELETE\_WINDOW", self.\_exit)

The Tk base class will close the window and end the mainloop if the user clicks X on the window.

We can't abruptly end the mainloop. Why?

* The self.\_exit method is a callback that runs when X is clicked. It overrides the default behavior of the Tk base class. Create the \_exit method with 2 lines of code:

1. cancel the search thread

2. self.destroy() # clean up and close window