Project 5

Computer Science 2334
Spring 2009

User Request:

"Create an Applet to Find Sequences of Places Near to One Another and Map Them."

Milestones:

1. Create an address book, as done in Project 4, for places of interest that includes the global coordinates (latitude and longitude) of the places it contains. *5 points*
2. Create a GUI to allow the user to see a list of places to choose from, choose starting and ending places, choose the radius within which to search, and map the sequence found (if any). *5 points*
3. Create a recursive method to search through the address book and find sequences of nearby places and store them in an appropriate data structure. *20 points*
4. Use the MVC paradigm as exemplified by the **ExampleCleanMVC** code discussed in class. *20 points*
5. Make your software run as either an application or an applet. *10 points*
6. Make your software handle I/O exceptions. *10 points*

- ► Develop and use a proper design. *15 points*
- ► Use proper documentation and formatting. *15 points*

Description:

An important skill in software development is extending the work you have done previously. For this project you will rework Project 4, modifying the GUI to allow for the selection of two desired end points for a path and adding a recursive method to search for such a path between them. Recursion gives us a simple yet powerful way to deal with objects that are defined in terms of objects of the same type. For example, factorials are defined in terms of other factorials, directories (or folders) in computer file systems are defined in terms of what they contain which may include other directories (or folders) as well as other types of files, and places in the world can be defined in terms of their locations relative to other places. In addition, you will modify your input method to handle I/O exceptions and run as either an application or an applet.

For this program you may reuse some of the classes that you developed for your previous projects, although you are not required to do so. Note that much of the code you write for this program could be reused in more complex software, such as trip planning software or a social networking website.

<u>The Idea:</u>

The idea behind this program is simple. Imagine a friend of yours says, "Boy, there really are a lot of colleges and universities out there! In fact, I bet you could hop from one to the next to the next from a college in New York to one in Los Angeles and never have to make a hop of more than 60 miles." Rather than have to guess if your friend is right, you can use this program to check and, if your friend is right, even show a sequence of hops that would do just that.

Of course, your code won't be so rigid as to only do searches from NY to LA or only check for hops of 60 miles or less. In fact, it will allow the user to select as starting and ending (goal) places any two distinct places listed in its database of places and hops of many different lengths. (However, you may want to limit the maximum hop length, and the maximum number of hops allowed. See *Recursion*, below.)

The Code:

Model, View, Controller:

You should organize your code using the same Model, View, Controller paradigm you were required to use for Project 4. However, in this case you must determine for yourselves what classes this will entail and you should name them and organize them appropriately.

GUI:

Your graphical user interface should support the software described above. Exactly what the GUI will look like and the classes you use to create it are up to you, as long as they are logical, easy to understand, and follow established conventions. In your write up, you should include a diagram explaining what the GUI looks like and how it works (as was provided for you in Project 4).

Map Output Format:

As with Project 4, your software for Project 5 will interface with Google maps to display to the user a map in a web browser. The map that your program displays needs to clearly mark the starting and ending places that were selected by the user, as well as all of the places found in the path and the connections between them, and should be scaled to make good use of the map space available. Beyond that, exactly what your map looks like is up to you, as long as it represents the data in a way that is logical, easy to understand, and follows existing conventions. In your write up, you should include a diagram explaining each type of symbol used on your maps.

Data File Formats:

Your software will have import, export, load, and save functionality identical to that of your previous project. In addition, it should be able to save and load paths found from start to goal places; that is, when an ordered list of places is found in the search from start to goal, your software should be able to save and load that list along with the start and goal places, in addition to mapping it. All of these I/O methods should include exception handling.

I/O Exception Handling:

Your program should include `try-catch` blocks as appropriate to handle possible I/O exceptions. In particular, main should not re-throw any I/O exceptions; they should all be dealt with inside your code. You may re-throw exceptions within your code but you need to ensure that the exception handling setup is logical, intuitive, and follows accepted conventions. Be sure to minimize the size of each `try` block.

Recursion:

Your program will need to find a path, if it exists, from the selected starting location to the selected goal location. To do this, your program will use recursion to generate possible paths from the start to the goal with each hop in the path being equal to or shorter than the user-selected distance. You may choose to have this recursive search be either *depth first* or *breadth first*.

Depth-first search (DFS) means that from the start node, we look first at one of its connecting nodes, then one of the nodes connected to that node, and then one of the nodes connected to that node, etc. until we either find the goal or run out of unvisited nodes in the present path, then we back up only as far as necessary to go down another possible route. For example, imagine that the start node is node S, which is connected to nodes A, B, and C. (By "connected" here we mean within the search radius specified by the user.) So, we choose one of these nodes, say A and check if it is the goal node. Imagine that it isn't. Then we need to determine which unvisited nodes are connected to A. S is connected to A, of course, but S isn't unvisited, since we started there. Imagine that nodes D and E are also connected to A. So, we choose one of these nodes, say D. Imagine that D is also not the goal node. Now we need to determine which unvisited nodes are connected to D. Imagine there are none. So, we back up to A and look at its remaining connected node E to see if there are any unvisited nodes connected to E. Imagine that nodes F and G are connected to E. So, we choose one, say F. Imagine that F is not the goal node but is also connected to G and no other unvisited nodes. So, we visit G and determine that it is the goal node. Then the path we have found is S, A, E, F, G, so we report that.

Breadth-first search (BFS) means that we search more or less in parallel down all the possible paths we find. Given the same example, starting from node S which is connected to A, B, and C, we still need to look at one of these connected nodes first. So, as with DFS, we choose A. Since A is not the goal node, we choose another one of the nodes connected to S. Say that we choose B. Since B is not the goal node, we choose yet another one of the nodes connected to S. That one must be C. Since C is also not the goal node, we finally move on from the nodes directly connected to S and start looking at the nodes indirectly connected to S. So, we go back to A and look at the unvisited nodes connected to A, which are D and E. As before we choose D first and determine that it is not the goal node, so we look at E, which is also not the goal node. Now, however, rather than continuing on with the nodes connected to E, we back up to node B, which was connected to A. Imagine that B is connected to G. We check to see if G is the goal node, find that it is, and report the path found which is S, B, G.

BFS has the advantage that it will always find the path with the shortest number of hops from node to node from the start to the goal. DFS has the advantage that you don't have to save as much information in memory at the same time for paths of the same number of hops. Both are very easily implemented recursively and the only difference in the code between the two of them is when in the method you make the recursive call.

Note that either DFS or BFS may run out of stack space if asked to search through large search spaces. For this reason, you may wish to limit the maximum hop length a user can enter, which will, in turn, reduce the number of connected places for each place. If you choose to do this, you should notify the user of this fact and include it in your write up. You may still find that your searches run out of stack space if your starting or ending location is in a very densely populated part of the country. If you wish to handle this possibility more intelligently, you may do so for extra credit. In addition to limiting the maximum hop length, you may wish to limit the maximum number of hops that can be in any path. If you choose to do this, you should notify the user of this fact and include it in your write up.

Implementation Issues:

This project gives you more flexibility in design than your previous projects. If you have questions about your design, please work through them with your teaching assistants and instructor before moving on to implementation.

Due Dates and Notes:

Due Dates:

Your revised design and detailed Javadoc documentation are due on **Thursday, April 30th**. Submit the project archive following the steps given in the submission instructions **by 9:00pm**. Submit your revised UML design on *engineering paper* or a hardcopy using UML layout software, a hardcopy of the index page of your Javadoc documentation, and a hardcopy of the stubbed source code at the **beginning of lab on Thursday, April 30th**.

The final version of the project is due on **Thursday, May 7th**. Submit the project archive following the steps given in the submission instructions **by 9:00pm**. Submit your final UML design on *engineering paper* or a hardcopy using UML layout software, a hardcopy of the index page of your Javadoc documentation, and a hardcopy of the source code at the **beginning of lab on Thursday, May 7th**.

Sources:

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

Group Work:

As noted in the syllabus, you are required to work on this programming assignment in a group of at least two people. It is your responsibility to find other group members and work with them. The group should turn in only one (1) hard copy and one (1) electronic copy of the assignment. Both the electronic and hard copies should contain the names and student ID numbers of **all** group members. If your group composition changes during the course of working on this assignment (for example, a group of five splits into a group of two and a separate group of three), this must be clearly indicated in your write-up, including the names and student ID numbers of everyone involved and details of when the change occurred and who accomplished what before and after the change.

Each group member is required to contribute equally to each project, as far as is possible. You must thoroughly document which group members were involved in each part of the project. For example, if you have three functions in your program and one function was written by group member one, the second was written by group member two, and the third was written jointly and equally by group members three and four, both your write-up and the comments in your code must clearly indicate this division of labor. Giving improper credit to group members is academic misconduct and grounds for penalties in accordance with school policies.

Extra Credit:

There are many possibilities for extra credit on the mapping side of the assignment. The key is to make them useful, explain why they are useful, and implement them well. For example, you could associate additional data relating to each point with its marker so that it appears when the user “mouses over” the point. How many points you may get for any addition depends on how useful it is and how well it is explained and implemented.