Project #4

Computer Science 2334

Fall 2007

User Request:

"Create a stack-based implementation of a Fibonacci sequence calculator and a Graphical User Interface."

Milestones:

1. Implement a simple *Graphical User Interface* (GUI) in *Swing* that allows for inclusion of the functionality described below. *(5 points)*
2. Write iterative code to calculate the numbers in the Fibonacci sequence using a *Stack* to keep track of intermediate states (approximating recursive calls). *(20 points)*
3. Provide a GUI component to display the current value of the answer at each step. *(5 points)*
4. Provide a GUI component to display the contents of the *Stack* at each step. If the *Stack* becomes too large to fit in the space allocated in the window, your GUI should provide the user with the ability to scroll up and down in the display of the *Stack*. *(10 points)*
5. Create a dialog using *JDialog* that allows the user to select (1) the Fibonacci number to calculate, (2) the maximum size of the *Stack*, and (3) the number of seconds to pause between each step in the GUI display of the *Stack*. *(10 points)*
6. Use event-driven programming to have the values entered into the *JDialog* above effect the running program immediately. *(15 points)*
7. Create buttons for the user to click to pause and resume the GUI display of the *Stack*. *(5 points)*

Develop and use a proper design. *(15 points)*

Use proper documentation and formatting. *(15 points)*

Background:

As discussed in class, any recursive program can be turned into an iterative program in many different ways. One way to do this is to use a data structure to keep track of values that would be passed as arguments in a recursive call.

If we use this approach for the Fibonacci sequence, we only need to keep track of two values at a time –

the values of the two previous Fibonacci numbers (if we are working our way up to the Fibonacci number in question). This means we can simply declare two temporary variables to hold these two values and keep replacing their current values with new values as we move our way up the sequence.

However, for other recursive programs, we may not know in advance the number of intermediate values we need to retain because the number of recursive calls needed to for a given input may be based on that input. In such cases, we cannot define a fixed number of temporary variables to hold our "arguments." Instead, we must use a data structure that allows for growth to an arbitrary size.

One such data structure is a *Stack*. It makes intuitive sense to use a stack, since the recursive calls we are trying to mimic store the arguments on the call stack. Of course, we could use another data structure that can grow arbitrarily large (like an *ArrayList*) but, for this assignment, we'll stick to using a *Stack*.

Naturally, using such a flexible data structure means that we suffer from the same sorts of problems that plague recursive programs, such as a slow execution speed and the possibility of expending available memory, so it does not make much sense to implement code for calculating Fibonacci numbers this way, but we are going to do it anyway, since I don't want to make you implement stack-based iterative code for a more complex recursive function. Indeed, it doesn't make much sense to have you implement stack-based iterative Java code for *any* recursive function whatsoever, since Java supports recursion. Still, this assignment will give you practice with some important concepts, so we're doing it anyway. Alternately, if you don't buy the pedagogical reasoning behind this assignment, just think of me as your Pointy-Haired Boss who is making you do inexplicable things. (But remember, Dilbert or Asok will review your work, so don't think you can pass off on your PHB code that doesn't meet spec.)

Description:

Project #4 will be a GUI-driven demo to show the user a representation of a stack keeping track of intermediate "argument" values, much the way such values are stored in the call stack during recursion.

When the user starts up your program, it will present him or her with a dialog box that will allow for selection of three values: (1) the Fibonacci number to calculate, (2) the maximum size of the *Stack*, and (3) the number of seconds to delay between each step in the GUI display of the *Stack*. The default value for the maximum *Stack* size is 100 and the default value for the display delay is 1 second. There is no default value for the Fibonacci number to calculate and your program will not calculate any number until the user has entered a value for this.

Note that this dialog should remain open, even while the program is calculating and displaying the Fibonacci values. Indeed, the user should be able to select new values at any time and have the program make use of these new values immediately, as follows:

1. If a new Fibonacci number is chosen, your program should clear out the stack and its GUI presentation of this information and start the calculations all over again.
2. If a new maximum *Stack* size is chosen, your program should check to see if that number has

been exceeded already. If so, it should present the user with an error message saying that the user cannot select a maximum stack size smaller than the current size of the stack. Otherwise, it should simply replace the current maximum stack size with the new maximum stack size, to make use of a discussed below.

3. If a new value is specified for the display delay, your program should immediately begin using the new value, rather than the old value, as *delayTime*, as discussed below.

Once the user has entered a Fibonacci number to calculate, your Project #4 code will display two additional items of interest to the user: (1) The contents of the stack, so that the user can see the intermediate "argument" values as the answer is being calculated, and (2) the current value of the answer as it is being calculated. The first of these should provide the user with the ability to scroll up and down the displayed values, if the size of the stack is too large to view within the existing window all at one time.

Once these components are all visible, your program will begin its calculations of the Fibonacci number in question, keeping and displaying the running value for the answer and the state of the stack. Between each pass through the iterative loop there will be a delay of *delayTime* seconds, so that they user can see what has changed.

The other GUI components that your Project #4 code will possess are two buttons, which will appear once the calculations begin and will be alternately enabled. One button will be labeled "Pause" and will be enabled as soon as it is displayed. When it is clicked, it will halt the calculations and the updates of their display and disable itself until the other button is pressed. The other button will be labeled "Resume" and will be enabled as soon as Pause is clicked. When Resume is clicked, it will cause the calculations and the updating of their display to begin again where they left off, enable Pause, and disable itself.

Due Dates and Notes:

1. No third-party GUI packages may be used. Only standard Java classes and packages are allowed on projects. ***Using a non-standard class will result in the program not compiling on the graders' computers, and as the course syllabus specifies, a non-compiling program will receive a grade of zero.***

2. Make sure to start early and budget your time well. Once you've got a good design, you can write a part at a time and test it before moving on to the next part. ***If you do not understand the design you developed in lab, it is your responsibility to attend office hours early in the project to get help with your design.***

3. Your revised design and detailed Javadoc documentation are due on ***Friday, November 9th.*** Submit a

hard copy of your revised UML design ***on engineering paper or using UML layout software*** at the ***beginning of class***. Also submit hard copies of both the stub code and the JavaDoc documentation generated from the comments in your stub code at the ***beginning of class***. Finally, submit a soft copy of your stub code using the submit tool on *codd.cs.ou.edu* by ***9:00pm***. This submission counts as part of the design and documentation portions of the project grade.

The commands for submitting your stub code on *codd.cs.ou.edu* are:

```
cd design4
/opt/cs2334/bin/submit cs2334-010 project4-design <.java filenames>
```

where `<.java filenames>` is a list of the .java files you are submitting.

3. The final version of the project is due on ***Monday, November 19th.*** Submit your final UML design ***on engineering paper or using UML layout software*** at the ***beginning of class***. Also submit hard copies of your source code, the JavaDoc documentation generated from the comments in your code, and COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt. Finally, submit a soft copy of your source code files and the COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt files using the submit tool on *codd.cs.ou.edu* by ***9:00pm***.

   The commands for submitting your final project on *codd.cs.ou.edu* are:

```
cd project4
/opt/cs2334/bin/submit cs2334-010 project4-final <.java filenames> <other files>
```

   where `<.java filenames>` is a list of the .java files you are submitting and `<other files>` are the other files you are required to submit. Make sure that the COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt files are present in the directory along with all of your .java source files.

4. The *Get Out of Jail Free* due date for this project is ***Monday, November 26nd***. Submit your final UML design ***on engineering paper or using UML layout software*** at the ***beginning of class***. Also submit hard copies of your source code, the JavaDoc documentation generated from the comments in your code, and COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt. Finally, submit a soft copy of your source code files and the COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt files using the submit tool on *codd.cs.ou.edu* by ***9:00pm***.

   The commands for submitting your final project on *codd.cs.ou.edu* using Get Out of Jail Free are:

```
cd project4
/opt/cs2334/bin/submit cs2334-010 project4-goojf <.java file names> <other files>
```

   where `<.java filenames>` is a list of the .java files you are submitting and `<other files>` are the other files you are required to submit. Make sure that the COMPILATION.txt, EXECUTION.txt, and OBJECTIVES.txt files are present in the directory along with all of your .java source files.

5. 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.

6. 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.