46-935 Final Homework Assignment

This assignment involves more work than previous assignments. Therefore, in the computation of your final grade, this assignment will count twice as much as a normal assignment. We have covered the basics of what you will need to do in lecture, however you will probably need to refer to the JDK documentation and your textbook for details about specific API functions. As the semester is coming to a close, we cannot grant extensions lightly – please start early!

Setup:

Download the packages from the Assign3 handout directory via FTP (/afs/andrew/course/46/935/handout/Assign3) or download the zip file from the 46-935 web page. Make sure to download all of the files (some files have changed since Assignment #2).

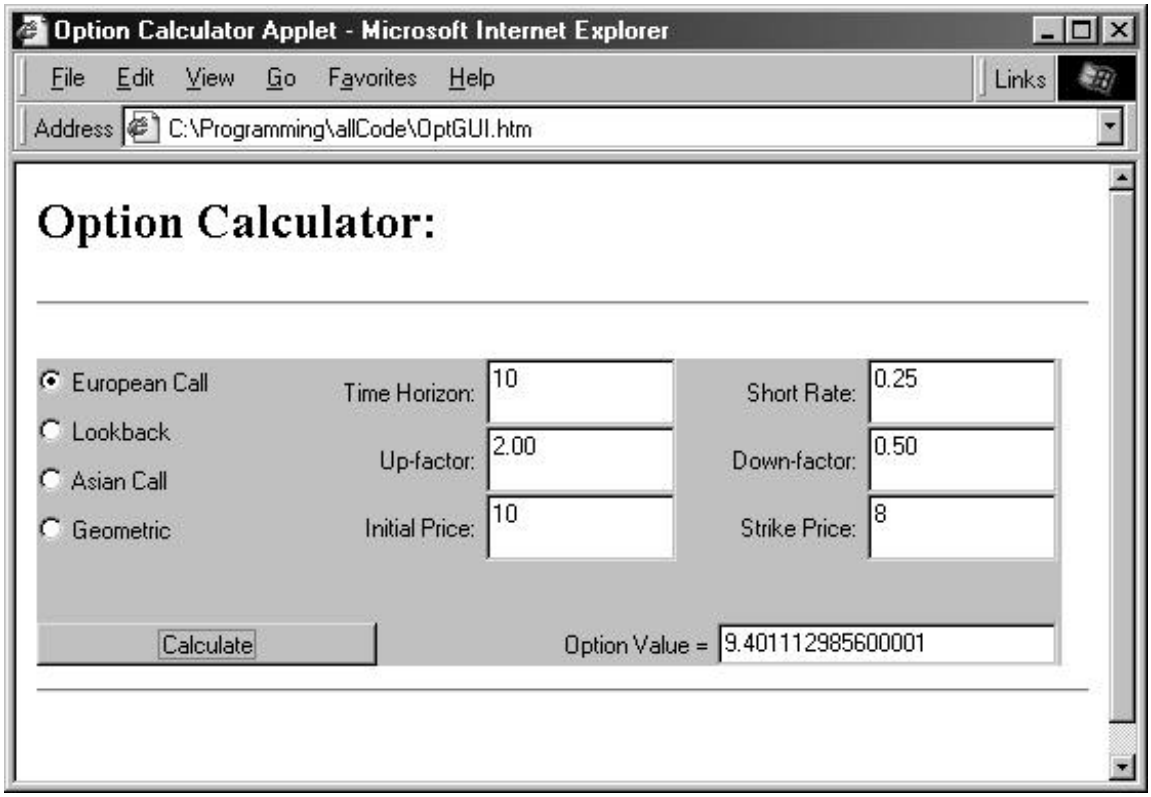
Part 1: (40 points)

In this portion of the assignment you will be asked to create an applet that provides a graphical user interface (GUI) for the Option Calculator code that we used in the first assignment. To create the applet, you will use Java's Abstract Windowing Toolkit (AWT) which was covered in the fourth lecture.

This applet will (ideally) allow you to access the Option Calculator code from any web browser that supports Java applets. The GUI interface that we are asking you to code is fairly simple. You must allow the user to choose what type of option they would like to evaluate from the four that we have available (European Call, Lookback, Asian, and Geometric). In addition, you must allow the user to input the parameters of the option – the time horizon, short rate, up-factor, down-factor, initial asset price, and strike price (notice how these parameters correspond to the parameters of the option class constructors!!!). Upon pressing a "Calculate" button, you must display the calculated option value to the user. If the user enters garbage values in the parameter fields and presses "Calculate", you should display "ERROR" in the option value field.

(next page)

The following is an example of what your interface **could** look like. Feel free to design your own interface, however it must meet (at minimum) all of the functional requirements listed in the previous paragraph.



For this interface, I chose to use Checkbox components for selecting the option type (arranged with a GridLayout in their own panel). For entering the parameters, I used TextField components with Label components (again arranged in their own panel with a GridLayout). These panels (along with a panel containing the "Calculate" Button and "Option Value" Textfield) were arranged in the *applet container* using the BorderLayout. Again, this is just an example. You may design and code your interface any way you like as long as it meets the functional requirements.

I have given you a skeleton file for your applet code in the optionCalc package (OptCalcGUI.java) and an html file that loads the applet (OptGUI.htm). The OptGUI.htm file must be in the parent directory of the optionCalc package.

Development Tips: First plan your applet interface on paper, then code your applet in stages. Use the "appletviewer" program that comes with the JDK to view your applet at each stage (appletviewer OptGUI.htm). When satisfied with your applet, view it in Internet Explorer or Netscape.

Part 2: (30 points)

In this part of the assignment, you will write a test program (in the testPrograms package) for the Interest Rate code. In class, we showed a test program (p. 31 of Lecture 3 notes) that generated an interest rate lattice by hard-coding the time horizon, yields, and yield volatilities. This approach is inflexible and clumsy!

Your test program must input the desired time horizon, yields, and yield volatilities from a file and must write the output (the printed lattice) to a file. The name of the input file and the name of the output file will be supplied as *command line arguments* to your test program (don't hard-code the file names!).

We have defined a simple file format that you must be able to parse as input.
The first number in the file will be the (integer) time horizon. The next line will be a sequence of doubles separated by spaces representing the yields. The final line will be a sequence of doubles separated by spaces representing the volatilities.). The file "term1.txt" in the testPrograms directory is a sample input file.

Your program should handle incorrectly formatted input files gracefully (print a message and exit rather than breaking). The file "testTermIO.java" is a skeleton for your program.

Part 3: (30 points)

In this part of the assignment, you will use the idea of compound state prices (as described in Lecture 3) to create a faster version of the AbstractTermStructure class.

The files FasterTermStructure.java and YieldVolObject.java are currently just copies of the code from the now familiar AbstractTermStructure and SlowYieldVolObject classes. You must modify these source files (in the interestRate package) to use the faster compound-state prices algorithm from the lecture. For a description of this algorithm see pages 47-50 in the lecture 3 handout.

Once you've implemented the FastTermStructure and YieldVolObject classes, extend FastTermStructure to implement the Normal and LogNormal models (call these classes NormalFast and LogNormalFast respectively). Use the NormalFast and LogNormalFast classes to verify that your compound state prices algorithm works correctly.

Extra Credit

These extra credit problems are a chance for you to learn more and improve your grade in the process. Each extra credit problem can raise your final grade in the course by as much as five percent. These problems can only help your grade, not hurt it. You may choose not to do the problems. *Partial solutions will receive partial credit.*

Extra Credit Problem 1:

Write an extension to the Option Calculator applet that allows the user to visualize the generated lattice. Upon pressing "Calculate", the applet should show the generated lattice (nodes and edges) drawn on a Canvas container. Clicking on a node in the lattice should bring up a window with the key values stored in that node.

Extra Credit Problem 2:

Recall that the generalized term structure class needs the volatilities of bonds maturing at various time horizons. One popular way of getting the volatilities is by looking at the cap. Black's Formula (see Hull) can be used to find the implied volatility of a bond maturing at a certain date by matching it to the observed cap. Using Black's Formula, modify the term structure code to take caps as input instead of volatilities.

Handin Procedure:

Pittsburgh/NY: FTP the modified source files to the course handin directory (where *userid* is your andrew user id):

/afs/andrew/course/46/935/handin/*userid*/

London: Email the modified source files (preferably in a zip file) to jeffreys@andrew.cmu.edu **AND** pavani@andrew.cmu.edu.