Overview: This is the second (and final) part of the project. It is due on Wednesday, December 4th, at 11:59pm.

Github: As with Part 1, your files should be hosted in a private Github.

Coding: Your job is to extend the functionality of the provided code. Do not delete or alter any of the given code. Your code must have the same return values and function prototypes as what is provided, as well as the same level of information hiding and encapsulation.

The object of your program is to provide a GUI that allows a user to generate Hexagon Crosses. To do this, it will need to be able to generate prime number efficiently, as well as load prime numbers from a text file. Full specifications are below.

Testing: You will want to test your code to ensure it is both correct and highly functional. You will want to generate sufficient test cases and validate your code to ensure it completes all code paths successfully.

Style: In programming, it is very important to use a specific and consistent programming style. Use the refactor tool to rename class names as per the Google Java Style Guide: https://google.github.io/styleguide/javaguide.html When in doubt, use this style guide.

Notes: Indents in these files are set to 2 spaces. You can change this setting in your Eclipse file by clicking Window Preferences. -> Expand Java Code Style ->Click Formatter -> Click the Edit button -> Click the Indentation tab. Under General Settings, set Tab policy to Spaces only, set spaces to 2.

What You Turn In:

1. The completed .java files in a ZIP file on eCampus.
2. Your github repo address. Include this in a text file in your zip file. Remember to set your repo private and add Glen-TAMU as a collaborator.

Grading: Grading will be based upon completeness and correctness. If your code is fully functional and passes all tests while adhering to the design guidelines, you get full points for this section. If your code is partially functional, you will get partial credit. If your code does not compile or run, you will get minimal credit. If you turn nothing in, you will get no credit.

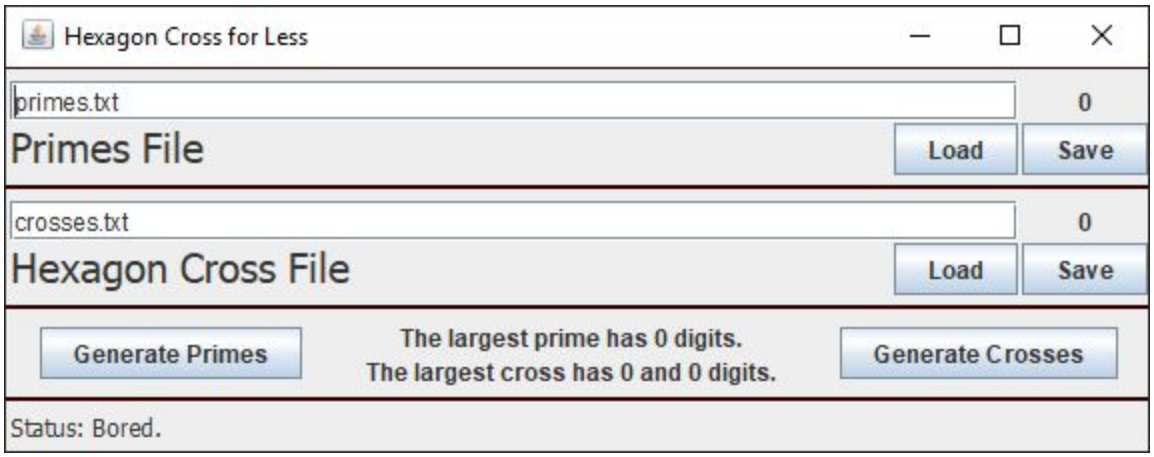
Academic Honesty: As with most programming project, student code will be subjected to extensive anti-cheating monitoring. Algorithm design will be checked, as well as code layout.

Late Work: Late assignments **will not** be accepted. You may make multiple submissions. If you do so, only the most recent submission will be graded. You are encouraged to submit your project early and then make additional submissions as the deadline approaches so that you do not end up with nothing turned in.

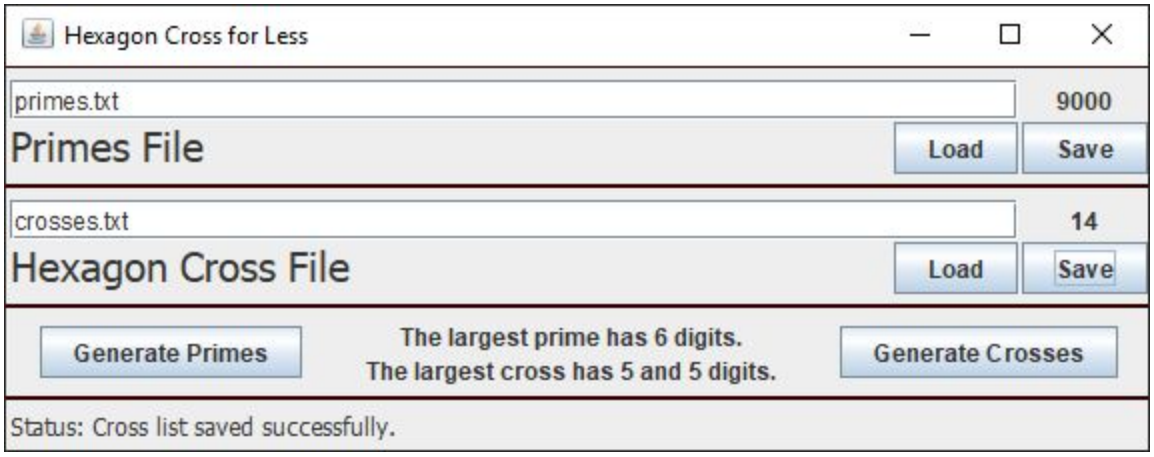
Program Specification

GUI

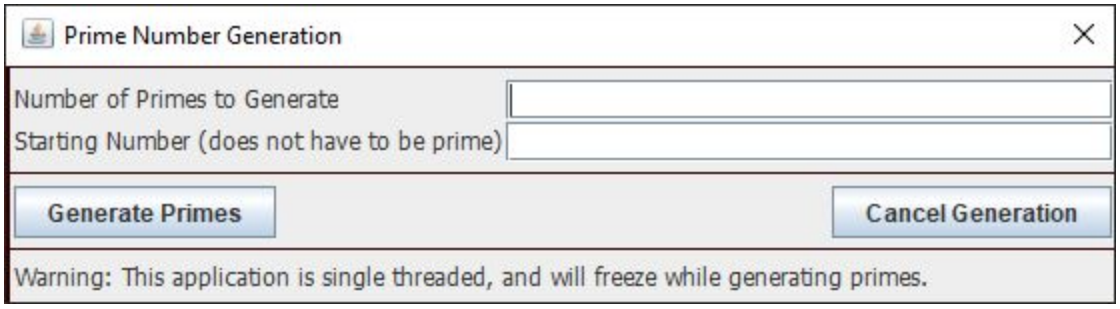
Your program should create a primary window for the program that is 1000x400 pixels in size. The background color should be the Texas A&M specific shade of maroon used by the school. The application may be given any name decided upon by the developer. This window should contain a textbox for entering the Primes file name and the Hexagon Cross file name. Buttons for loading and saving each of these files should be specified, and the default file names should be initially set for each of these items.



The main window should contain a status bar to keep the user informed of the success or failure of any given operation. It should also contain metadata about the work being done, such as the number of primes generated or loaded (9000), the number of Hexagon Crosses generated or loaded (14), the length of the largest prime (6 digits) and the length of the two numbers in the largest Hexagon Cross (5 and 5 digits).



Buttons for generating primes and generating crosses should be implemented. The Generate Primes button should create a dialog box for the user to enter the number of primes to generate and a starting number. The program should be able to accept and process numbers of arbitrary length. The program may be single-threaded.



Programming

Your program should use the Swing library in Java to generate the GUI. You may use any aspect of Swing except the Pair class.

All your data files should be read from and written to a data subdirectory. You may assume that the data files are always error free, with the exception that the prime list may contain duplicates.

All Java classes for this project should be part of the *default package*.

You are free to add any **private** helper classes, methods, or attributes you feel are needed to complete the project.

Your code should be efficiently designed using best software engineering practices. This includes ample and useful comments. The provided code intentionally has neither of these features.

Config.java

A Configuration class is provided to allow you a central location to configure your application name, data path, prime file name and cross file name. These defaults and values must be in Config and not hard-coded into the body of the program.

FileAccess.java

A FileAccess.java file is provided for you to create static functions to load/save primes and crosses. They should return true if the operation succeeds and false if it fails. This success or failure should be used to update the main window's status bar with the appropriate notification for the user. Your save functions must use the "**for-each**" Java language construct and iterators to work through the prime and cross lists. All saves and loads should be to the "data" subdirectory, but should allow user-specified filenames from the GUI. This custom filename ability can allow the user to specify additional path information as part of the custom filename.

MainWindow.java

This is the main file for creating your application window. It should create the window described in the GUI and add ActionListeners to all the GUI elements that need user interactivity. Complete code is provided for the “Generating Prime Numbers” dialog box. You may optimize this code or use it as-it. It is intended as a starting point example of writing a GUI, and is intentionally uncommented to force the student to research and understand the code, rather than just using cookie-cutter copy-and-paste.

NaiveTest.java

A NaiveTest.java file is included with a simple, functional isPrime function. You should replace this function with a more efficient test for primality. This test **must** be absolute. This means you can not use any of the probabilistic tests as your sole determiner of primality. (This includes but is not limited to Fermat’s, Miller-Rabin, and Solovay-Strasson, Adleman-Huang, Baille-PSW, and Frobenius.) You must use a deterministic primality test. You can use a fast probabilistic tests as an initial qualifier, as the NaiveTest uses BigInteger’s isProbablePrime function, but you must then use an absolute test. You are not only allowed, but encouraged to search Stack Overflow and other online resources for ideas and code to implement this one feature. However, you must cite your sources and give appropriate credit for any code that you do not come up with solely on your own.

Pair.java

A Pair.java class is provided to give pair functionality. You may use this class or the one you wrote for Part 1. If you use the one your wrote for Part1, take advantage of Eclipse’s refactoring tools to automatically move it to a new class.

Primes.java

A partial Primes class is provided. You may use this as a starting point or use your own. This is a deliberately “not optimal” solution, but it is a viable one. You will need to add code for the blank public functions in this class.

You will also need to add two Iterators to the Primes class, one for Primes, one for Crosses. You will use these in your save functions to iterate through the relevant ArrayList. You will implement these iterators as private classes that implement the *Iterable* interface.