Project Part IV: Final Report

Team Member:

Amber Womack
Mike Watson
Erik Eakins

Title: Tenjin Math

Questions:

1. What features were implemented?

From our use cases, we implemented the following features:

Create Account - for a personalized profile page. System displays page and user can select cancel or create. If username is not unique, system displays user name used. System displays create account page.

Login - to login to existing account to see profile and resume progress, by selecting from dropdown menu. Displays user page.

Select Subject - to select the subject of choice. Subject module completed by displaying subject and loading subject page for a lesson.

Do Lesson - to begin a lesson module in the selected subject area. Generated lesson and then displayed as user selects start lesson. The user also has the ability to resume lesson, quit, or ask for help options.

Input Answer - to input and submit the answer to the given questions. Once it's inputted, the user can select next and move to the next problem of the lesson. This answer is stored within the menu on the left side of the screen for later use.

Quit Lesson - to quit an active lesson and save current progress. System displays another check to see if user wants to save or cancel.

Request Assistance - to request assistance if I come across a problem that I don't understand. Displays help options.

Submit Answers - to complete lessons by submitting my answers at the end. Displays submit confirmation and save.

View Results - to view lesson results at the end of the lesson module. System grade lessons and displays results.

Other features that we were able to implement included using JavaFX for the Graphical User Interface to have CSS for user aesthetics.

2. Which features were not implemented from Part 2?

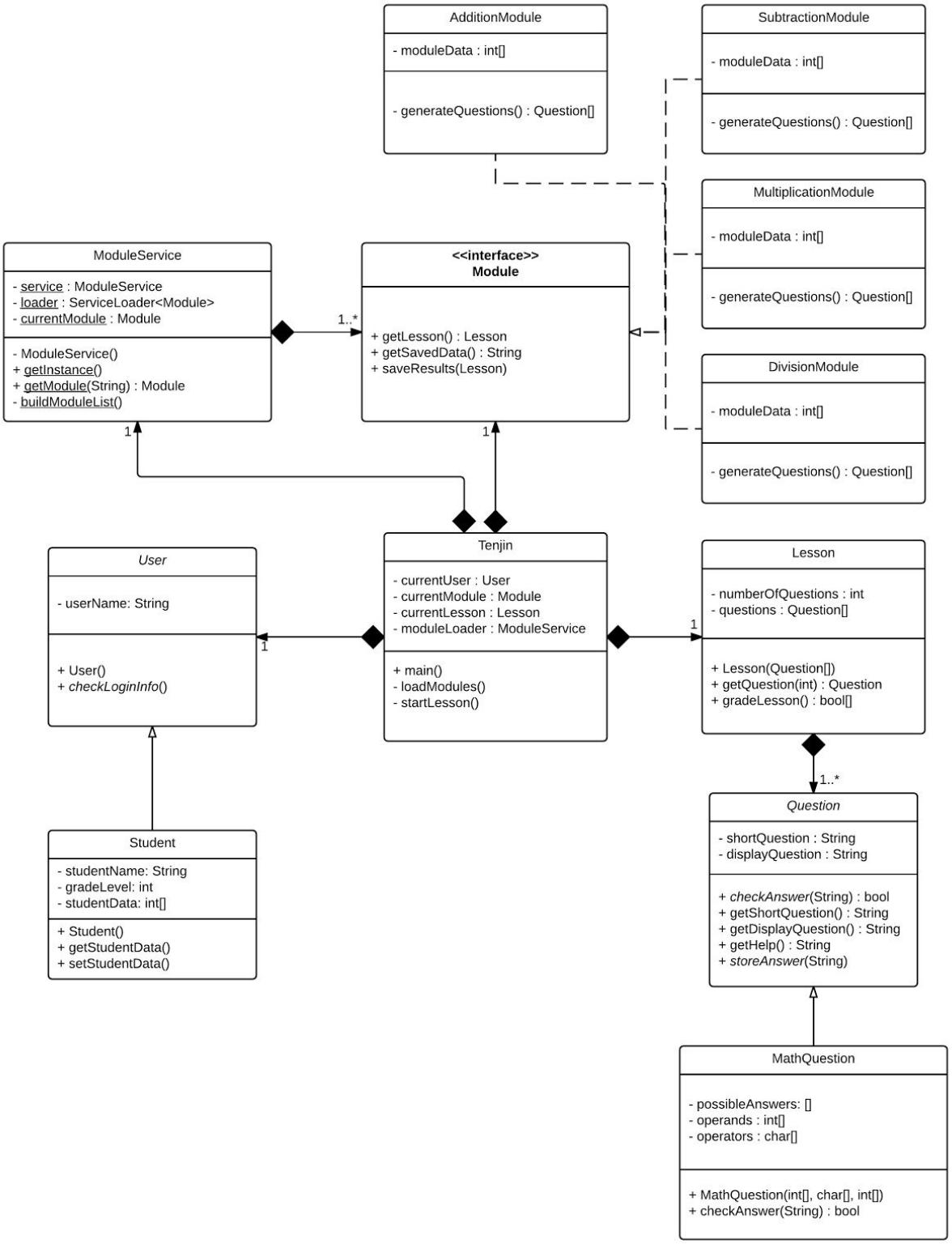
Some of our extendable features weren't implemented that included:

Saving preferences - Within our preference settings, we wanted to be able to change the background color of the windows. However, the user input drop-down functionality is in the program to be implemented in the future.

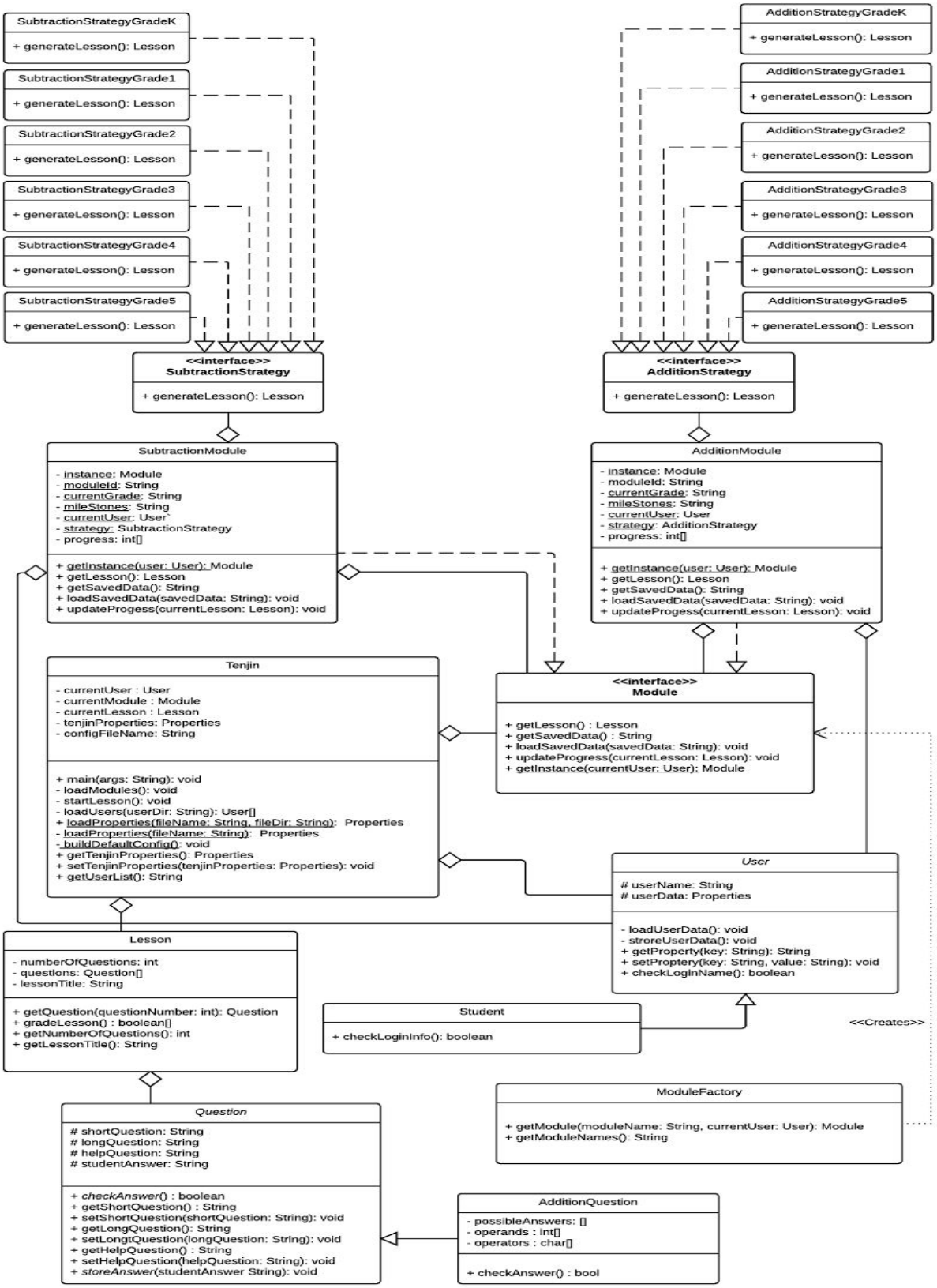
Review overall progress - We wanted to be able to review the progress of completed lessons, but will be a future expectation. The review button is in place.

All Subjects not fully implemented - We wanted to be able to also have multiplication or division lessons for those subjects. At this point, we have the ability for these other subjects to be implemented, but only addition and subtraction will load lessons.

3. Part 2 class diagram using Lucid Charts:



Final class diagram using LucidCharts:



What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

We changed our module service/moduleLoader class to moduleFactory which incorporated the Factory Design pattern. This is because we found that it would be difficult to design the program to dynamically load modules at run time. For the sake of meeting the deadline for the project, we left out the use of a Java ServiceLoader in favor of a factory design pattern. This allowed us to keep most of the functionality of the program while sacrificing some future expandability.

During the actual implementation process, there were several private methods that needed to be added in order to keep the code as clean and "non-smelly" as possible. We also had to temporarily omit the multiplication and division modules in the interest of time. The modular design utilized by the factory design pattern will allow these and other math modules to be easily incorporated in the future. We changed some method names to be more specific and understandable (refactoring), and added more methods for increased functionality. For the main Tenjin, we also had to add several other methods. Regardless of these changes, we still can appreciate the design process up front, which helped us in the development and planning.

4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

We ended up implementing three design patterns in our project: Singleton, Factory, and Strategy. The Singleton design was used in all the classes implementing the Module interface (AdditionModule and SubtractionModule). This was used in order to ensure that there would only be one instance of each of the modules, regardless of how long the program was being used or how many users have taken lessons.

The Factory design was used in loading the correct module in order to allow for the greatest flexibility for future additional modules. Any future math expansion module will simply have to implement the Module class with its own functionality, be imported into the ModuleFactory class, and add the module name and getInstance method call into the getModule method's switch statement. No changes to the GUI or any other module would be necessary.

The Strategy design is used in the Addition and Subtraction modules, in order to separate the question generation for grades K through 5. Each of the grade levels use its own algorithm to generate grade-specific questions to fill the lesson. When the AdditionModule or SubtractionModule singleton object is first instantiated, it will select an appropriate strategy based on the user's grade level.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

The process of design and analysis of a system is an important step of the development because it provides the guidelines and pre-information needed to support the organization's project objectives. Going through the process of design and analysis prior to initial coding, reduced the overall amount of time it took to implement the project because the design process essentially mapped out the entire project. However, the coding still changed (see Class Diagram) and it was difficult to get started on the designing portion initially because of previous coding habits. In addition, everyone was able to keep the same vision for the project and how it was to be implemented, due to the research done for the UML diagrams. The UML diagrams also helped the team communicate the plan of the project and the details of the system. It allowed us to visualize this structure and how it would be implemented. The use cases and other diagrams explicitly told us how we needed to implement the loading of the lesson modules instead of coding by trial-and-error methods. Design patterns are a way to think about, organize, and eventually solve your program or problem. We initially thought we would use certain ones, but throughout realized that the Factory and Strategy seemed to work the best. Understanding these patterns and learning their operation, we were able to visually represent the structures we wanted to implement and see these interactions in our code structure.