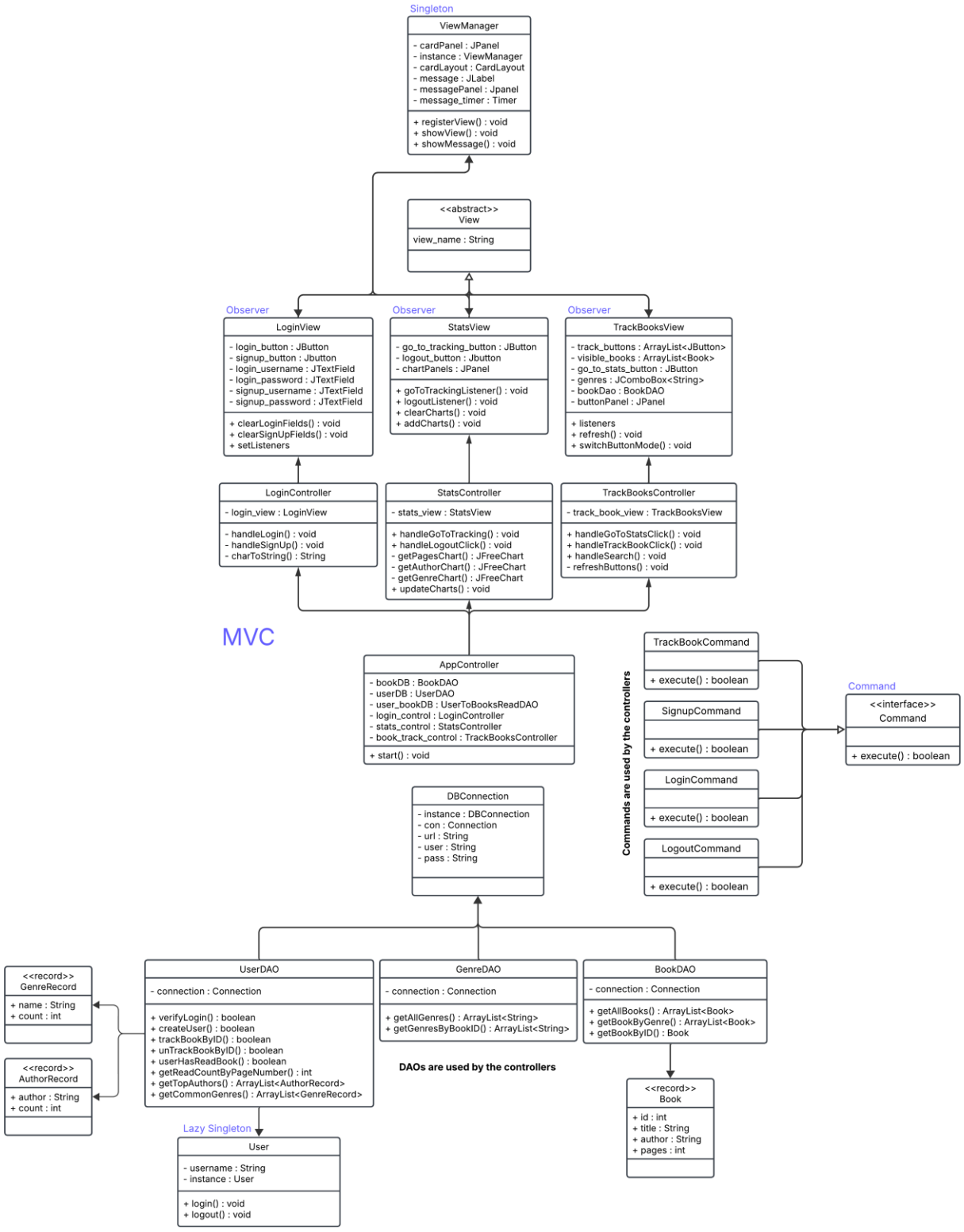
Project 5-7 Final Report

Developer: Linnea Jones

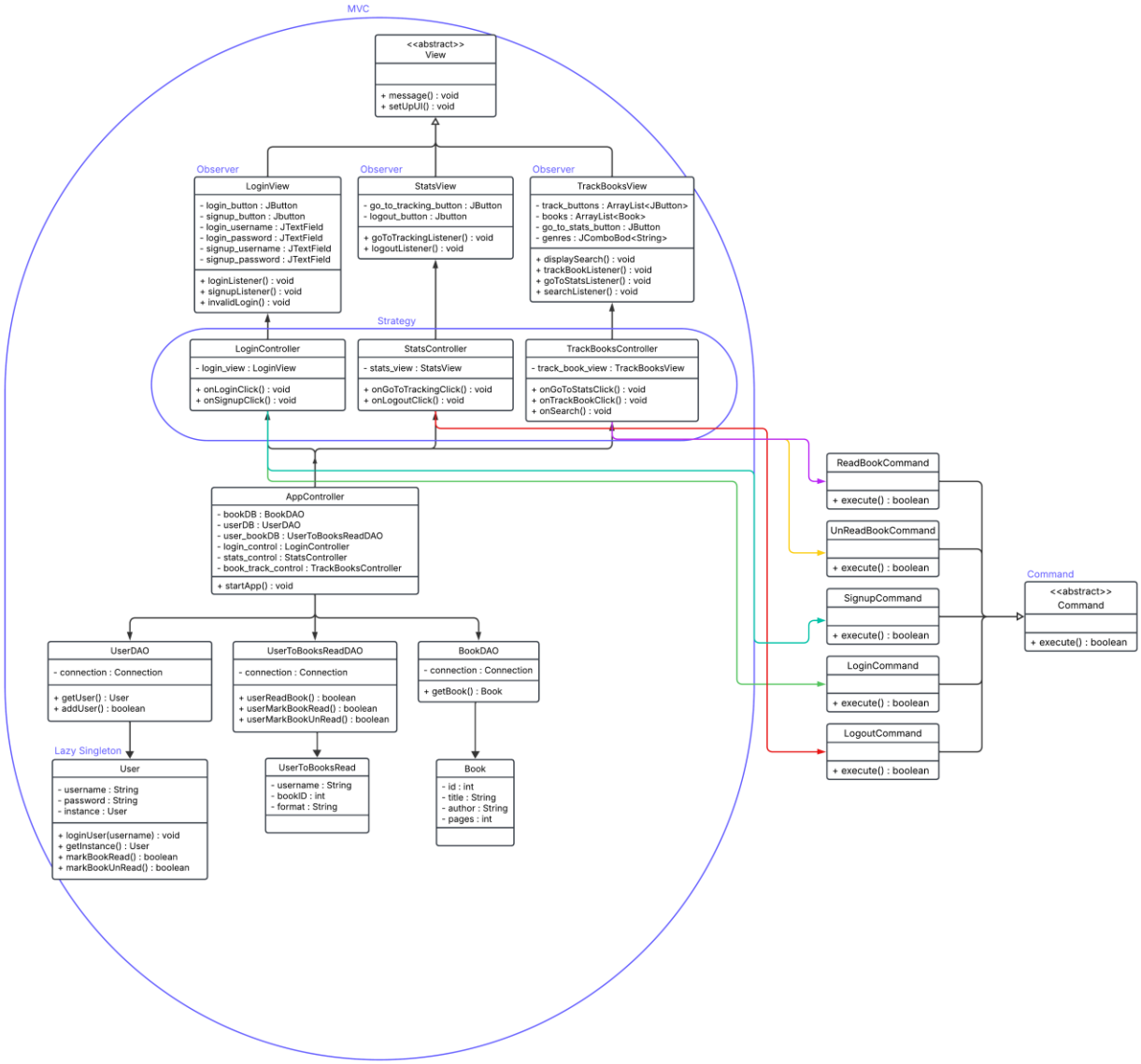
State of System: Every initially defined feature was completed for the project. The user can login and is directed to a statistics page displaying statistics about their book history. They can also navigate to a page where they can track which books they've read. The only other things that would have made the app a little bit better would be some nicer formatting on the statistics and book tracking pages, because the formatting turned out to be rather difficult and there were some problems with text overlapping with buttons that didn't end up fixed. Additionally, finding another way to preload books with their genre relationships would have been nice. As it is now, the database has about 115 books loaded into it because I had to individually write multiple MySQL insert statements per book to add them and their genre relationships to the database.

Final Class Diagram: Key changes primarily consist of the classes having many more methods than they did when designed for project 5. Additionally, the track/untrack book command was consolidated into one command to reduce coupling. A view manager class was also created to maintain views across one page with similar overall formatting. Additionally, the Book object was converted to a record, and GenreRecord and AuthorRecord classes were defined in order to pass Resultsets as method returns in a way that Java understands. Records seemed like a good idea since none of the information in the objects were meant to be changed, and they come preprogrammed with necessary getters. The colorful arrows are missing from the commands in the final UML diagram because there were so many of them it got difficult to interpret, but each command is used once or twice throughout button handlings in the controllers. Finally, the user_to_books_read DAO and POJO were replaced with the Genre DAO, since many of the user_to_books methods seems as though they belonged more in either the User or the Book DAO because user_to_books never needed to instantiate its own object, it only had class methods used by other objects.

Final Diagram:



Project 5 Diagram:



Third Party Code Statement: There are some pieces of code that came from third party sources. These are notated within the code as well as listed here:

- In the stats and track book controllers, there is a component listener overwrite within the constructors which came from ChatGPT, and forces the controller to refresh every time the view is interacted with by the user
- The code for creating pie charts was modified from a code piece from the following site: https://www.tutorialspoint.com/jfreechart/jfreechart_pie_chart.htm
- The code for creating bar chars was modified from a code piece from the following site: https://stackoverflow.com/questions/23665260/bar-chart-with-exact-value-printed-on-top-of-each-bar

- ChatGPT suggested the form of the try-catch statements for running SQL queries which was used multiple times throughout the DAO files.
- The code for setting up the SQL connection was modified from code presented on the following site: https://medium.com/@samuelcatalano/integrating-java-with-databases-a-guide-to-connecting-java-applications-with-mysql-oracle-and-feaa92cb1da8
- Pieces of the front-end code within the view constructors defining the panels and panel components came from the following site: https://www.geeksforgeeks.org/java/introduction-to-java-swing/
- ChatGPT and Copilot were both used in putting together the code that allows vertical scrolling on both the statistics view and the book tracking view.
- ChatGPT also suggested some of the lines of code defining the look of the app within the view class constructors.

OOAD Process Statement: One element of the design process was the UML class diagram. While it was fairly extensive and included pretty much every class the project needed, it didn't define every class method that the app ended up requiring. It would have been a good idea to think a little bit more about the logic behind each class when making the UML diagram, so that there wouldn't have been so many more unexpected class methods added after the design.

One positive thing was having the UI wireframes completed before beginning the coding process. I typically jump right into coding and then get overwhelmed when I don't have certain things already laid out for me to implement in code. So when I got to work on writing the code for the views, it was really helpful to already have a plan for exactly what my Swing code needed to replicate.

I ran into a little bit of trouble maintaining the MVC pattern when I began implementing the track book and statistic pages. I knew what I wanted to do but was stuck on how to implement certain things such as updating the page and action listeners at the same time, without giving the view object access to the controller object. I learned from ChatGPT about the possibility of implementing a component listener, which can notify the controller anytime the view changes in any way. This was immensely helpful because it meant I could keep the controller logic and action listener updates inside the controller, and still have them reload whenever the view reloaded, without telling them to reload from inside the view class code.