Friday 5/7/2021: N/A: I was taking an AP test, so I didn't do any work today.

Thursday 5/6/2021: III.8: Poor, poor me. It's never simple. The AI section is not modularized code, while the rest of the project is. This means that the rest of the project cannot access anything from the AI part if everything is compiled together. After much testing, I was able to do the following to fix the problem: compile and package the AI section into a jar file and install it into the local Maven repo, add a dependency on the AI section into the GUI POM, compile only the backend and GUI sections (not the AI), include the generated jar files from the AI into the classpath for java when running the program.

Wednesday 5/5/2021: III.8: I discovered that I had written the public method from yesterday incorrectly, so I had to reformat that method into the correct method: using the loadSingleSentence method of a blank CnnSentenceDataSetIterator instead of creating a CSDSI from a blurb. Tomorrow I'm going to fully integrate the AI part with the GUI part of the project, should be simple enough.

Tuesday 5/4/2021: III.8: I wrote a method to generate a CnnSentenceDataSetIterator from a given blurb, and then I wrote methods to load the prebuilt word model and the neural net itself from files. I wrote a public method to combine all of this, which takes a string and returns the predicted genre.

Monday 5/3/2021: III.8: I created a supplementary GUI for the prediction module; using my previously gained knowledge of aligning buttons correctly, it took only as long as expected. I modified some of the original components, making the text area wrap text, for example.

Tuesday - Friday 4/27/2021 - 4/30/2021: III.7: I did pretty much the same thing every day: try different values for different hyperparameters and retrain and evaluate the model. Training and evalutation takes a minimum 10 minutes, so this was a slow and painful step. The various hyperparameters I messed with were: minibatch size, learning rate, l2 regularization coefficient, and various optimizers/updaters.

Monday 4/26/2021: III.6, III.7: I did the initial first training session and evaluated the model. It performs far below what I had hoped, at around 50% accuracy. I started researching ways to troubleshoot neural networks. I sent an email to the people who provided the dataset because of a problem with the data: there are no linebreaks at all in the provided files, which means that many words are mushed together (e.g. hellothere); this messes with the neural network.

Friday 4/23/2021: III.5: I researched something called an early stopping trainer, which prevents you from having to specify manually a number of epochs to train the network over. Instead, you specify a maximum number of epochs, a maximum time (which is useful for my purposes, since my machine isn't exactly new or fast), and some other parameters, and it will train the network and automatically save the best version of the neural network. I started adding this to my code.

Thursday 4/22/2021: III.4, III.5: I tested the neural net today, and discovered that the LabeledSentenceProvider I had written didn't work. After some additional testing, I discovered that the input data files were using the UTF-8 format, and the Scanner I was using in the sentence provider automatically tried to read it in as pure ASCII, which resulted in an early unexpected null character or EOF or something. I specified that the Scanner should read in data as UTF-8, and voila! It works. After just one training cycle of the AI ("epoch"), it had an accuracy of about 50%; I'm not sure how good that is after just one epoch, but it seems decent to me, given that there are 42 different genres.

Wednesday 4/21/2021: III.5: I translated the academic article with the help of the example file into dl4j code, which mainly involved looking up what a particular ML/AI thing was, how it was represented in dl4j, and so on. This wasn't particularly hard work, but it was a lot of typing and took a while to get everything the way I/the article wanted it.

Tuesday 4/20/2021: III.5: Now, with an actually working data input system, I continued my research from earlier. I found one particular academic paper that claims its specific network design works well on text classification tasks with prebuilt word embedding models, so I read through that. It was mostly Greek to me, but with the help of a dl4j example, I think I'm going to be able to construct a version of that model in dl4j (even if I don't fully understand what the neural net does or why this one is particularly good).

Monday 4/19/2021: III.4: It doesn't matter what I'm going to have to use! I, in continuing my research, discovered that the way I have the data now isn't compatible with both easy-bert and deeplearning4j, so the last week of work was for naught. My work today took easily 3-4 hours, so I'm going to give a shortened version: I researched, found out that I could use a prebuilt word2vec model with dl4j, discovered a specific DataSetIterator for use with natural language processing CNNs, created a custom LabeledSentenceProvider, wrote a method to create the specific DataSetIterator from a file and some other parameters, tried about 40 different word2vec models that didn't work, discovered a bug in dl4j, and found a workaround to said bug.

Friday 4/16/2021: III.4: I finished this milestone today, as I was able to create a DataSetIterator from the RecordReader object I already have. I had some extra time after I did that, so I started my research for next week on what a neural net for text classification might look like. From what I'm seeing so far, this might turn out to be more complicated than I had thought. I had naively assumed that a 'normal' (I don't know the technical term) neural net would be sufficient, but from my research, I'm going to have to use a Convolutional Neural Net (whatever that is).

Thursday 4/15/2021: III.4: I created a CSVRegexRecordReader object to read in my CSV data files, using a custom delimiter (instead of a comma). I played around with the RecordReader to see how it functioned, and read more documentation. Have I mentioned how bad the deeplearning4j documentation is? It's very bad.

Wednesday 4/14/2021: III.4: I modified my CSV converter to associate each blurb with only genre. The original data file has multiple genres, of varying specificity, for each blurb, and I think the neural net is going to need each blurb to only have one genre, so I changed the CSV converter to do that. It wasn't super hard, as the genres are listed in the original file in least specific to most specific order, but I did have to make a custom list of genres that the AI would accept, as I don't really care about, for example, the difference between "American History & Culture" and "European History & Culture" as genres; both should just be under the history genre.

Tuesday 4/13/2021: III.4: Since I can't directly import my data into Java using deeplearning4j APIs, I wrote a separate class to convert my data into a form that deeplearning4j will take, namely CSV. It turned out to be a pretty simple program in the end, but I had to analyze the format that my data is in (some custom type of XML) to understand how to write the CSV converter, so it took longer than I expected.

Monday 4/12/2021: III.4: I mainly read the (poor) documentation for deeplearning4j about loading and processing data from files. I think that the end result of this milestone is going to be an object of type DataSetIterator, so I looked into how I can get one of those. I also researched the process for manipulating data, which doesn't look like it's going to be useful for the specific data that I have.

Friday 4/9/2021: III.2, III.3: easy-bert was much easier to install than deeplearning4j. I just had to add two dependencies to my POM, and then that was that. I could access and load a pre-built BERT model and use it to generate word embeddings for strings. After that, I just had to download the data set I'm going to use, which was in the form of a zip file.

Thursday 4/8/2021: III.1: Now I'm trying to use deeplearning4j without modules, but it still won't work. An example file training a simple NN won't run; it complains that it can't find a class in the library, which is troubling. I downloaded the entire git of deeplearning4j tutorials and ran the same example in their preset environment, and it worked, leading me to figure out what I had done wrong. I copied a couple of their files over into my environment and made a change to my build script, and it ran! Finally.

Wednesday 4/7/2021: III.1: That didn't work, I couldn't get the plugin to run before the compile step, which errors and stops the process. After a while of painstaking research and looking at XML, I realized that I had just put the code telling the plugin what to do in the wrong place. Silly me. Anyway, with that plugin working, I tested again. Surprise, it doesn't work! Apparently Java had some sort of reflection/access issue with the automatic modules, which I would have to edit the source of each of the dependencies to fix (there are 50+ dependencies). I decided to abandon the modular system for this one section of the project, and deal with the repercussions later (as I can't really do anything else -- deeplearning4j is pretty much the only useful neural network library for Java).

Tuesday 4/6/2021: III.1: I continued investigating why Java wasn't recognizing the nd4j package, and I discovered why. So, deeplearning4j is apparently not designed to be used with the JPMS modules (Java 9 and up), so Java is resolving automatic module names from the names of the jar files. This is a problem, as Java has a problem with the jar file nd4j-native-platform-1.0.7.BETA.jar, as the automatic module name is nd4j.native.platform, which isn't allowed, as native is a Java keyword. I think I can work around this: I researched some of the Maven plugins and used the dependency-copy plugin to create a copy of the nd4j-native-platform jar called nd4j-ntive-platform-1.0.7.BETA.jar, so that Java wouldn't complain.

Monday 4/5/2021: III.1: I looked up the deeplearning4j tutorial and its requirements/dependencies. I found their example POM file and copied the dependencies over into my POM file as well. I added "requires" statements to the module-info file, and tried to import some classes from the deeplearning4j and associated libraries. However, when I tried to build the project, I discovered that one of the dependencies (called nd4j-native-platform or something similar) was not being found. I couldn't figure out why.

Friday 4/2/2021: III.1: I created the Genre module in the file system, added a temporary blank module-info file, and copied the POM from the backend module over. I edited the POM and removed the dependencies and plugins that this module didn't need. I edited my build script, and copied my testing class over from the backend module as well. I made a placeholder test and built and ran the project to make sure everything was good to go for next week.

Thursday 4/1/2021: II.10: I connected the three buttons to their respective functions: the search and insert buttons connect to the search and insertion GUIs, and the help button pops up an information box with information about the program, the MySQL backend, some other technical information, and a basic rundown of the workflow.

Wednesday 3/31/2021:II.9: It's literally three buttons! Really. Why is this so hard?? I also looked into using a table-like structure with fixed column widths, but it was a CONSTANT -- the width doesn't depend on the text of the buttons. Magic numbers are bad. After that, I had an idea: what if I unbound the width of the buttons, but stick the button container into another container, and manage the button container's width. This works (finally).

Tuesday 3/30/2021: II.9: I'm back to the stupid buttons. I'm beginning to hate these buttons. Yesterday I found an Oracle article specifically about sizing GUI elements, but their suggestion, to make the maximum width of the buttons unbounded and let the VBox do the sizing, didn't work. Thanks Oracle. I would have thought that you would have known your own product better.

Monday 3/29/2021: II.9: Today I started working on creating the "main" GUI that the user is brought to when they first start the application. It just has 3 buttons: insert a book, search for a book, and help. I added the buttons into a VBox and figured that would probably be it for the layout, but no. The buttons all have different widths, which is unacceptable. I looked at some different ways of forcing the buttons to be the same width.

Friday 3/26/2021: N/A: I have a little extra time today, so I decided to add some logging functionality to the project, as working with databases is rather error prone. I made sure that whenever the user is presented with some kind of error message, the associated exception is logged to a file.

Thursday 3/25/2021: II.7, II.8: I added the logic to update the book in the database. I fixed a small bug from yesterday's code, where I had forgotten to call a method on a GUI element to actually get the text input from it. I also added an update button to the search results page, which I had forgotten to do. I connected all of the separate pages together.

Wednesday 3/24/2021: II.7: I added the logic to update the local Book object (not in the database) from the user's inputs. This is also more complicated than it seems, as there's a lot of error and duplicate checking to do, to ensure that the input is sanitized and suitable for the field it's for.

Tuesday 3/23/2021: II.6, II.7: I changed a Book method to return a set instead of a list, as that is more helpful, and I added logic to maintain sets of tags to add to a book and tags to remove from a book. It was harder that it sounds, and I had to do some tracing to finally create a suitable, working solution. I also added functionality for returning to the previous page (using the back button) to the update and view classes.

Monday 3/22/2021: II.6: I added a method to remove an Iterable<String> of tags, since there was already a method to remove an array of tags, but no Iterable method. I had finished the view GUI over the break, and I made it concrete (it had previously been abstract) to match the new update GUI and class (which are concrete because they need to store a lot of information per object). I created the update layout, which is very similar to the insertion layout, as they are similar operations.

<Over spring break>: II.5: I figured out how to correctly align the buttons. It took a little while, but I eventually got it and ended up using a few too many collections for my taste, but it works.

Friday 3/12/2021: II.5: I worked on creating the GUI to view books in. It's pretty simple, for the most part, so that wasn't too hard. I also now knew how to force a ScrollView to take up the available space, which I did for the tag ScrollView. However, I had some trouble getting the buttons (to go back, to update the book, and to delete the book) to align in the way I had wanted, so I wasn't able to completely finish this milestone.

Thursday 3/11/2021: II.5: I worked on creating the results list layout for the search GUI, so that you can see the available books and pick one to view or update. I had trouble getting the ScrollView containing the VBoxes of book information to fill the entire space it had, so it took a while longer than anticipated.

Wednesday 3/10/2021: II.4: I linked the Search GUI completely to the backend and created my final testing object and tested the entire process of searching for a book, which works. The program generates a correct BookQuery object, which then generates correct SQL!

Tuesday 3/9/2021: II.3, II.4: The code from yesterday with the visual links doesn't work, and it's going to be too time consuming to fix, so I just completely removed it. I also wrote a series of methods, first iterative, and then recursive (when I realized that iterative would just make it take up more storage than necessary) to translate the graphical SearchAtoms into a BookQuery object.

Monday 3/8/2021: II.3 : I made the height grow more slowly (by making the vertical padding smaller). I added code to allow operators to replace a previous SearchAtom with a new one after they take ownership of the SearchAtom, so that the list of operator and property SearchAtoms can refill properly. I also tried to make a visual link between a blank node and an operator, so that you could have part of an expression outside of the main operator, thus mitigating the problem of large height.

Friday 3/5/2021: II.3: Now I can actually work on the final layout for searching. I added a list of base elements and operators that the user can drag out into a canvas-like area to compose their query. I noticed that the NOT dragging wasn't working, however, and I found out that there was a bug revolving around a NullException, which I fixed. Then, I made the different search operators different colors, so as to more easily distinguish them. I then noticed that a compound search atom (with other search atoms in it) grows in height very quickly, so I thought about how to mitigate that.

Thursday 3/4/2021: II.3: I added methods to detect when the drag event is over one of the two possible drop locations for AND and OR operator objects, for which I had to do some geometry. I also found out that each node has its own coordinate space that is separate from the coordinate space of the scene, which was annoying, until I found out that I could convert scene coordinates into local coordinates, which solved the problem I was having. Then, I made the receiving areas change color when hovered over by a drag event, and finished the implementation of drag and drop.

Wednesday 3/3/2021: II.3: I added some more logic to the operator class to include NOT in the list of their accepted operators and did the basic drag-and-drop functionality, which works, yay! I had to add a bunch of event listeners to the classes, but I ran into a problem. I need to get the surrounding class (either SearchElement or SearchOperator) but I can only get the StackPane that holds the visual stuff for each class. To solve this, I made SearchAtom (the base interface) into an abstract class that extended StackPane, so that the surrounding class would be the StackPane, solving the problem. This also let me reuse some of the “accept a drag” operation code for both elements and operators.

Tuesday 3/2/2021: II.3: I started working on the search GUI. However, it's more complicated than I thought it would be. I want to have drag-and-droppable elements representing a condition (e.g. does the book have this tag?) or an operator (and, or, not) so that users can create complex queries. However, this is pretty difficult. I did a lot of research on custom components, the already existing drag-and-drop system, and more. I created a base interface for both the elements/condition and the operator classes to extend from, and designed the look of the spots.

Monday 3/1/2021: II.1, II.2: I had some aligning issues that I fixed today: Instead of trying to align many horizontal boxes, I instead used a GridPane, which, as the name suggests, gives a grid to align elements onto, which was very useful. This let me completely finish the layout, and then I connected the GUI elements to the backend, which wasn't super hard.

Friday 2/26/2021: II.1: This is the first GUI thing I'm doing in this project, so I had to set up another Java and Maven module, which took a little time to get right. I added another Test file to hold my tests for this module, and created the Insertion class to represent the insertion GUI. I then started working on the layout, since I have a pretty good idea of what I want it to look like.

Thursday 2/25/2021: I.10: I didn't particularly want to specify the password to start and stop the server on the command line, so I tried to get the stream to the stdin of the stopping program (starting works fine already) and stick the password through there, but that didn't work. So I wrote a method to replace {resource} with the absolute path to the resource, so that I can use another resource config file to pass to mysqladmin with the password in it. I tested the Database class today as well, and thankfully, everything works now.

Wednesday 2/24/2021: I.9, I.10: I added a method to update a Book without adding new flags, and I started actually working on programmatically starting and stopping the MySQL database. Since I want to abstract this as an AutoCloseable object (so that I can use it in a try-with-resources block), I created a new class to represent the database. Additionally, since I found no way to truly programmatically start the server, I have decided to use multiple configuration files that point to the executable and the necessary command-line options to start the server correctly. After the files are read in, they will be processed and passed to a ProcessBuilder or Runtime.exec().

Tuesday 2/23/2021: No work done in class; I took the practice UIL test.

Friday 2/12/2021: I.9, I.10: I continued looking into programmatically starting the server, which merited no further leads. I added a method to update a Book object in the database with its new properties and new tags, and I also added a method to remove tags from a book in the database. I would prefer that clients add and remove tags through these methods, and not by natively changing the Set of tags for each Book, so I changed some of the access modifiers.

Thursday 2/11/2021: I.8, I.10: I tested the removeAll method of BookQuery, which (thankfully) worked, and then I started researching how to start the MySQL server from within Java. I know that's kind of skipping the next thing on the plan, but this is much harder than updating a book. I'm still reluctant to call a console command from inside Java, which can fail for all sorts of reasons, but there might be no other way to start the server.

Wednesday 2/10/2021: I.8: I further researched multiple transactions with a single Hibernate Session, and found that if one transaction had an exception, then that session couldn't be used again later, which is a problem, I think. I also changed the structure of the removeIf function so that all matching books are removed in one transaction, so that either they're all removed, or none of them are. A partial removal would probably lead to bugs. I then made sure to assert that the Session isn't null, close the session, and remove it on an exception, so that we get a clean session later. I then tested removeIf, and it worked.

Tuesday 2/9/2021: I.8: I tested the methods for removing books that I wrote yesterday, and I was surprised to find that they didn't work. I did some debugging, and eventually figured out that I had forgotten to actually wrap the code that deleted the book in a transaction and commit the transaction to the database. So I fixed the code by adding the transactions, but then I was wondering about having multiple transactions in a single session, which might prove problematic.

Monday 2/8/2021: I.8: I considered how I wanted to write the function to remove all of the books in a query. I thought I might provide a method that just takes a list of books, but then books that aren't in the database or don't match books in the database (but have the same ID) could be deleted, so I decided to have the remove functions be a part of the query class, so that only books in the database could be deleted. I wrote the function to delete all of the books that matched the query, and I also wrote a function to take a Java predicate that would filter the stream() of results and then delete the matching results.

Friday 2/5/2021: I.6, I.7: All I had to do today was test the remaining methods in the BookQuery class, so I wrote a few test cases and made sure that the correct results were returned. Then, I looked at what I had to do next on the plan (write a function to get all of the books that match a query), and realized that the BookQuery class already does that, so I'm technically ahead now. Yay!

Thursday 2/4/2021: I.6: Today was a pretty calm day. All I had to do today was add the methods that test for equivalence for the rest of the book properties (e.g. is the book's title a certain string? or is the author's first name another string?). The below bug won't apply to these, as the properties are actually fields in the database, and not additional objects that are linked to book objects.

Wednesday 2/3/2021: I.6: I found the problem. Essentially, before, I was searching for books that had a tag that was "awesome" but wasn't "bad" (which is all of the books with the tag "awesome") or I was searching for books that had a tag that was both "awesome" and "great" at the same time (which is no books). Instead of a test for equality, I need a contains test. So, what I ended up doing, after a lot of extra research and experimentation, is creating a subquery that counts the number of tags that match a string ("awesome" for example) a certain book has. Then, I can test to see whether the count is >= 1 (the book does have that tag) or if the count is

< 1 (the book does not have that tag). I didn’t have to completely rework the entire BookQuery class, which made me very happy.

Tuesday 2/2/2021: I.6: I fixed a couple minor compile errors (since I haven’t compiled for a while) and wrote the code to test the BookQuery class, which inserts dummy objects into the database to allow me to test with. I also added another addBook method in the BookCatalogue class that takes an Iterable instead of an array of string tags, which was helpful. I started testing with different queries, and I quickly ran into an issue: it doesn’t seem to work. The query to search for books with the tag “awesome”, but not the tag “bad”, returned all books with the “awesome” tag, regardless of whether or not they also had “bad”. The query to search for books with the tag “awesome” and the tag “great” returned no results, even though there were books in the database that had both tags.

Monday 2/1/2021: I.6: Today was kind of like my pre-testing day: I added a method to the BookQuery class to get the results of the query and reworked my Test class. I changed the way a Test object is supposed to be constructed and refactored some of my code in HibernateUtil to better fit the new testing paradigm.

Friday 1/29/2021: I.6: I didn’t do any coding today, as I wanted to make sure that my idea from yesterday worked before I finished the rest of the query class. I also didn’t do any testing on the computer, but I instead traced several hypothetical well-formatted examples to see if the logic was correct. It all appeared to be correct, so I traced hypothetical garbage examples to see if the class could correctly detect them and throw appropriate exceptions. It all checks out so far; I’ll try some real tech testing next week.

Thursday 1/28/2021: I.6: I scrapped all of the code I wrote yesterday in favor of a pseudo-prefix system, although I have ‘end’ functions too, that are to be used to end an ‘and’ or ‘or’ block. This way, I can have multiple conditions (like ‘does the Book have this tag or this author’) that are each ‘and’ed or ‘or’ed and have an explicit order of operations. I worked on the implementation for this, which necessitated using several stacks to properly maintain the number of conditions per block, the types of blocks, and the predicates/conditions themselves.

Wednesday 1/27/2021: I.6: I continued working on the query class and wrote the functions that represent ‘and’, ‘or’, and ‘not’ for the query system. I continued looking into Criteria (part of the Java Persistence API or JPA for short) and CriteriaBuilders. Unfortunately, I have a sneaking suspicion that the way I’m doing it now (pseudo-infix style) won’t work, as the order of operations won’t be discernable.

Tuesday 1/26/2021: I.3, I.4, I.5, I.6: I retested the Tag and Book classes to make sure that I hadn’t broken anything, and they were fine. I also tested the new method I wrote yesterday, and it worked as well. Then I started working on the query class. I perhaps should have thought a little more first, as I was kind of lost as to where to begin. Soon enough, however, I got an idea implementation-wise and got to coding.

Monday 1/25/2021: I.3, I.4, I.5: I changed the Tag equals method to only compare the contents of the String tag, and added an additional method to test for equality otherwise (if the tags have the same Book owner and same String tag). This was because I wanted a Book to have a Set instead of a List of tags, so that were no duplicates. However, before assigning a Tag to a Book, it could not equal any other Tag that Book had, as the Tag to add would have no Book owner, while the already added Tags would be owned by that Book. Thus, I created another method. I also moved some code around between my utilities class and my testing class to reuse some code. I created the BookCatalogue API class (the class that will contain the methods the GUI will use), and added the function to add books. With the way my code is setup, that function was about 4 lines of code, so it took much less time than I had expected.

Friday 1/22/2021: I.3, I.4: I completed the remaining read, update, and delete tests for the Book class, which went well. I then wrote the Tag class and annotated it; it's even shorter than the Book class, so that didn't take very long. I started the create/insert test, but Hibernate yelled at me and I found out that I had forgotten to add an @Entity and an @Table annotation to the Tag class, so I added those. Then the CRUD tests worked fine for both the Tag and Book classes.

Thursday 1/21/2021: I.2, I.3: Alright, so what happened was something like this: for some reason, Java wouldn't recognize a class file in a dependency, and there were no replacement dependencies that worked (due to a bug in Java 9: the 1.1 and 1.2 versions of the javax transaction API jars all export duplicate javax.transaction.xa modules, which the java.sql module already exports, causing an error). Thus, I upgraded to Java 11 and hoped the bug was fixed. It was, thank the Silicon Gods. Once I was able to run Hibernate, I found that some of my annotations were incorrect, so I fixed those and was finally able to run my test class. After a successful test, I checked the database using the MySQL console, and was extremely pleased to see that the book I had inserted was in the database.

Wednesday 1/20/2021: I.2, I.3: With Hibernate installed, I created the configuration XML file, and I learned shortly thereafter that I could set all of the settings in Java, so I created a class to configure Hibernate dynamically. After that, I wrote the Book class and annotated it, minus one small part that I'll add in when I annotate the Tag class. The Book class is pretty much just a holder for variables that correspond to the fields of the book table in the database, so it wasn't hard to write. However, when I wrote a simple test class to insert a new book into the database, I discovered that while I could import from Hibernate, it wouldn't run. Long story short, I added a few more dependencies and tried again. It didn't work. I'll try some additional things tomorrow.

Tuesday 1/19/2021: I.2: At this point, I hadn't even started anything with Hibernate yet -- my build system just got set up. I created the multimodule folder structure that I'm going to use for the entire project and wrote the individual pom.xml (Maven configuration) files. Finally, I added the dependencies (Hibernate, the Java Persistence API, the MySQL Java connector, etc.). I tried to immediately import from the Hibernate library, which failed. I eventually figured out that I had forgotten to require the Hibernate module in the module-info.java file. I learned about automatic modules in Java 9 and was able to successfully require Hibernate and compile a program that imported from Hibernate. However, the program wouldn't run as it couldn't find the

Hibernate module at runtime. I learned about the Java 9 module path, which Maven doesn’t set automatically (apparently). Thus, each of my dependencies must be manually included as an option to the java command when running. I wasn’t about to do that, so I wrote a PowerShell script to automatically craft the correct module path and run the java command for me. Thank the Silicon Gods, it worked! I could import from Hibernate, after all of this time.

Friday 1/15/2021: I.2: I started working on installing Hibernate, and I don’t think I’m going to be able to finish today. I’m using Java 9 and the module system, as I remembered from last year that I needed to use Java 9 to make the Java runtime packaging system work later on. I’m also using the Maven build and dependency manager, which does support Java 9. However, NetBeans can’t make projects that are both modular and use Maven. I messed around for a while, trying to get it to work, but it doesn’t. I couldn’t find any other IDEs that explicitly said they supported simultaneous usage of Maven and modules, so I decided to build my project manually. I downloaded the standalone Maven install and Visual Studio Code. Then, I tried to get a simple module Hello World to run manually after building with Maven. After I fixed a myriad of different errors, I finally managed to get a Hello World program running.

Thursday 1/14/2021: I.1: I continued my installation adventure and initialized the data directory, and opted to have a blank password for the default account, as I’ll change it later. I started the MySQL server for the first time and created a blank database to work with. Then, I changed the password for the default account and shut the server down.

Wednesday 1/13/2021: I.1: I researched what type of MySQL installation I wanted to do. In this case, I chose to “install” MySQL from a zip file, as this would be easier to replicate later on when I have to automate the installation of everything. I looked at the tutorials for installation from a zip file, and I wrote down a step by step list of what I had to do, for future reference. Then, I started with the installation. First, I downloaded the necessary zip file, and extracted it to a directory. Then, I wrote a configuration file that told MySQL where its base installation was and where its data directory was going to be.