**Familiarity Review (FINAL)**

**Name:** Jeremy James Alkire

**Date:** April 11th, 2020

**Week:** 5

**Coding Topic:** Hibernate

**Description of Understanding:** Hibernate is an ORM (object-relational mapping) Java framework that maps Java objects to a relational database. It maps Java classes to database tables, and Java data types to SQL data types. It automatically creates tables, removing the necessity for manual table creation, and generates SQL calls as well. Java classes are mapped to database tables by the usage of an XML file called hibernate.cfg.xml.

**Teaching Video:**  None.

**Starting at:** N/A

**Also Integrated with:** Servlets, Threads, Collections, HTTPUrlConnection, JSON

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| MobileServlet.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/MobileServlet.java> | This class is a servlet. This contains all the logic for the server side of the application, meaning it also does all the database operations.  Lines 61 and 62 generate a random number and initialize an integer variable with it. This is used to generate a unique primary key for the object that we will insert into the database. Hibernate, of course, has its own approach to doing this with @GeneratedValue(strategy=GenerationType.IDENTITY) and other annotations, but I preferred to approach it this way.  Lines 65-74 create an instance of the Project class and configure it by calling its setter() variables. The Project class is the POJO that the Projects table in the database will be based upon.  On line 77 a session factory variable is created via a static method call to getSessionFactory() in the HibernateUtil class. On line 79 the program gets the current session from the session factory variable. On line 81 the program begins the transaction. On line 84 it passes the project object to the save method of the session variable, and on line 87 it calls commit() to insert the object into the database.  Lines 95-99 create a new session factory variable, session, and transaction to prepare to make a query to the database. Lines 102-104 actually make the query and save the results inside of a List of Projects. | My code. |
| SystemTest.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/SystemTest.java> | This class is another servlet that is nearly identical to the above class, however, it does not query the database after inserting a test object into it. | My code. |
| Project.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/hibernate/Project.java> | This is simply a standard Java Bean class. It contains private variables, public getters & setters, and is serializable. It additionally includes annotations to specify column names (in preparation for the creation of a corresponding table in a relational database). | My code. |
| HibernateUtil.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/hibernate/utils/HibernateUtil.java> | This class configures a StandardServiceRegistry object with the hibernate.cfg.xml file, creates a Metadata variable, and then returns a sessionFactory variable to the caller of its only public method: getSessionFactory() | My code. |

**Coding Topic:** HttpUrlConnection

**Description of Understanding:** HttpUrlConnection is a subclass of UrlConnection that supports HTTP-specific features. Each HttpUrlConnection instance is used to make a single request, but the network connection itself can be shared with other instances. Network resources can be freed by calling the close() method on InputStream or OutputStream of a HttpUrlConnection, while disconnect() closes the socket. To create a new HttpUrlConnection, call openConnection() on a URL object, and then cast the result to HttpUrlConnection.

**Teaching Video:**

**Starting at:**

**Also Integrated with:** Collections, JSON

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| AddProject.java | <https://github.com/alkire-jeremy/CIT360/blob/master/ProjectTracker/app/src/main/java/com/example/android360x1/AddProject.java> | This class handles all the client to server communication of the application. This is where the user enters data into the application and sends it to the server via a POST request.  Line 118 creates a private subclass of Android’s AsyncTask class called ConnectToServlet. This class has an overridden method called doInBackground(), which is where the connection is made.  **THIS SECTION DOCUMENTS HOW THIS CLASS MAKES A POST REQUEST:**  On line 124 a URL variable is declared. On line 127 that URL is initialized with the target servlet URL (<http://10.0.2.2:8080/CIT/MobileServlet>). This is surrounded with a try-catch block that catches MalformedURLException. Line 132 declares a HttpURLConnection variable. Line 135 initializes this variable by calling openConnection() on the url variable and casting it to type HttpURLConnection. On line 141 we allow output, and on line 144 we set the connection method to POST. Line 158 declares an OutputStreamWriter variable, line 161 initializes it by calling getOutputStream() on the connection variable. Line 167 calls write() on the request variable to send a POST request to the server.  **THIS SECTION DOCUMENTS HOW THE CLASS RECEIVES AND PARSES A RESPONSE FROM THE SERVER:**  Line 180 creates a StringBuffer variable called ‘response’. Line 186 gets the response code. Line 192 checks if the connection is OK. Line 194 creates another BufferedReader variable, and line 197 initializes it by calling getInputStream() on the connection variable. Line 203 creates a string called “inputLine”, line 208 initializes it by calling readLine() on the BufferedReader variable called “reader”. Line 227 initializes a String array called output[] by parsing the JSON response sent by the server. To accomplish this, it calls the toString() method and the split() method, which splits the string by the delimiter symbol. From here, a static reference is made to the String array projects[] inside the ViewProjects class, and the variable is initialized with the output[] array. | My code. |
| MainActivity.java | <https://github.com/alkire-jeremy/CIT360/blob/master/ProjectTracker/app/src/main/java/com/example/android360x1/MainActivity.java> | This class is the main activity for the application. It has an identical copy of the private class ConnectToServlet that subclasses AsyncTask contained in AddProject.java.  MainActivity uses this to connect to the servlet and send a request to insert a test object into the database. This is done exclusively for System Testing, and its function call is bound to the System Test button. | My code. |

**Coding Topic:** Threads, Executors, and Runnables

**Description of Understanding:** Threads, sometimes called lightweight processes, provide a smaller execution environment than a process, and run within a process. Java provides the capability to code concurrently, allowing the programmer to create and manage multiple threads at once in a single process. Thread is an implementation of the interface called Runnable, and Executor is an interface that takes a runnable object as an argument. Most executor implementations use thread pools, which can help with management of large quantities of threads. One benefit of creating a subclass of Thread is that it can be started right away simply calling the start() method, where as a Runnable object either has to be passed into a new Thread constructor – followed by a call to start(), or it has to be passed to an executor, which calls its run() method. Threads can be joined, interrupted, and synchronized. It’s important to try and avoid thread collision issues such as deadlock.

**Teaching Video:** <https://www.youtube.com/watch?v=2vQsDmLkHXU>

**Starting at:** Entire video.

**Also Integrated with:** Servlets, JSON, Hibernate

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| TimerThread.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/TimerThread.java> | This is a simple example of a class that subclasses the Thread class. The code is extremely minimal and merely counts upwards from a value of 1 by incrementing an integer variable indefinitely while it is running. The values of this integer are displayed in the console (only to the developer) with System.out.println() and are spaced out by calling Thread.sleep(1000). | My code. |
| MobileServlet.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/MobileServlet.java> | An instance of the TimerThread class is created in MobileServlet’s constructor on line 34.  On line 35 the thread is started by non-statically calling its start() method on the object instance of the class.  This runs concurrently with the other instance of TimerThread that is started by the SystemTest servlet, which should run alongside MobileServlet when this application is being tested. When it is not being tested, only the instance started by MobileServlet will run.  In addition to this, all attempts to connect to the servlet via Android are made within a class that subclasses Android’s AsyncTask class. This is a helper class around Thread and is used for short operations. It runs asynchronously alongside the main UI thread in Android. This is implemented both in MainActivity.java in Android Studio and AddProject.java in Android Studio. | My code. |
| SystemTest.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/SystemTest.java> | An instance of the TimerThread class is also created in SystemTest’s constructor on line 34.  On line 35 the thread is started by non-statically calling its start() method on the object instance of the class. |  |

**Coding Topic:** Servlets

**Description of Understanding:** A servlet is a technology that extends the capabilities of servers and responds to incoming requests (of any kind). It’s a web component that is deployed on the server for sake of creating web applications and dynamic web pages. It’s also an API that provides many interfaces and classes. To create a servlet, a class must implement the Servlet interface.

**Teaching Video:**

**Starting at:**

**Also Integrated with:** Threads, Collections, Hibernate, HttpURLConnection

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| MobileServlet.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/MobileServlet.java> | This class is an extension of HttpServlet. It contains a constructor, a public void doGet() method and a public void doPost() method. They’re public as I attempted to implement JUnit testing into the servlet and public access was required.  It is designed to be hosted on Apache Tomcat in Eclipse.  The doPost() method is entirely responsible for all communication between the servlet and the client (Android).  Lines 61-62 generate a unique primary key value to be assigned to the object that we will insert into the database.  Lines 65-74 create a new instance of the Project class and configures it by calling its setter methods and passing the values returned by request.getParameter() into them as arguments.  Lines 76-81 create a new session factory, session, and transaction with Hibernate. Line 84 saves the object in the session and line 87 commits it to the MySQL database (hosted with XAMPP).  Line 92 declares and initializes a variable that will contain the response that will be sent to Android.  Lines 94-99 create a new session factory, session, and transaction with Hibernate. Lines 102-104 query the database for all the contents of the Projects table and assigns the result to a List of Projects.  Lines 107-110 iterate through the List of Projects and converts each Project object to a JSON string, only to append this to the end of the “responseToAndroid” String variable.  Lines 117-118 send this response back to Android. | My code. |
| SystemTest.java (ECLIPSE) | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/SystemTest.java> | This class another servlet, and it is nearly identical to the above, and thus its documentation should be the same, barring one difference:  It does not query the database after inserting a test object into it. Instead, it simply converts the original request to JSON and sends it back to Android. |  |
| web.xml | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/webapp/WEB-INF/web.xml> | This XML file is used to reference the servlets that will be used in the application. It additionally maps the servlets to specific URL patterns, so that they can be referenced by Android’s connection attempts. | My code. |

**Coding Topic:** JSON & QCJSON

**Description of Understanding:** JSON stands for JavaScript Object Notation. It is a language independent open standard for exchanging data on the web. It is a text based method of data transfer, and competitor to other similar approaches such as XML. It allows for different types of elements such as arrays and objects. Arrays are enclosed in square brackets, while objects are enclosed is curly braces. Individual elements are separated by commas. A major benefit of JSON is that it allows data transfer between different languages. Data can be converted into JSON, transferred across platforms, and converted back into a usable format on the other side. In Java, JSON is a method of serializing (or saving) objects, although it is not the only way. Objects can be converted into JSON strings, and then converted back into Java objects.

**Teaching Video:**

**Starting at:**

**Also Integrated with:** JUnit

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| MobileServlet.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/MobileServlet.java> | This class uses the QCJSON library (I imported the entire package into my project directly), and converts Java objects into JSON.  Lines 121-129 contain a method which utilizes this library to convert instances of the Project class to JSON strings. It accepts a Project object as an argument to its method and returns a JSON string equivalent to the caller.  This method (convertToString), which should’ve more aptly been named convertToJsonString is called on line 109 of this class. | My code. |
| SystemTest.java (ECLIPSE) | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/SystemTest.java> | Lines 121-129 contain a method which utilizes the QuickConnectFamily JSON library to convert instances of the Project class to JSON strings. It accepts a Project object as an argument to its method and returns a JSON string equivalent to the caller.  This method (convertToString), which should’ve more aptly been named convertToJsonString is called on line 109 of this class. | My code. |
| ViewProjects.java | <https://github.com/alkire-jeremy/CIT360/blob/master/ProjectTracker/app/src/main/java/com/example/android360x1/ViewProjects.java> | Lines 80-103 of this class contain a method called converToJava(). This method uses the QuickConnectFamily JSON library to convert JSON strings to Java objects, specifically: Instances of the Project class. It accepts a string argument and returns a Project object to the caller.  This method is called iteratively on line 54 of the class to convert the concatenated JSON string array sent by the servlet to Android. |  |

**Coding Topic:** Java Collections

**Description of Understanding:** A collection, sometimes called a container, is simply an object that groups multiple elements into a single unit. Collections are used to store, retrieve, manipulate, and communicate aggregate data. These typically represent items that form a natural group, such as telephone directories. A collections framework is a unified architecture for manipulating collections. Collection frameworks contain interfaces, implementations, and algorithms.  
  
Interfaces are abstract data types that represent collections. These allow collections to be manipulated independently of the details of their representation, and generally form a hierarchy. Nearly all interfaces inherit methods from the Collection superclass and pass these methods down to their implementations. Core interfaces in Java include: Collection, Set, List, Queue, Deque, Map, Sorted Set, and Sorted Map.

Implementations are concrete implementations of the collection interfaces. They are reusable data structures, and children of the interfaces, whom are their parents, and whom they inherit methods from. General-purpose implementations in Java include: HashSet, HashMap, ArrayList, ArrayDeque, TreeSet, TreeMap, LinkedList, LinkedHashSet, and LinkedHashMap.  
  
Algorithms are methods that perform useful computations, such as searching, sorting, inserting, and deleting elements inside of a collection. They’re described as polymorphic, meaning they’re able to be used in many different implementations. These are typically implemented via methods, such as add() and put().

**Teaching Video:**

**Starting at:**

**Also Integrated with:** JSON, Hibernate, Servlets

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| ViewProjects.java | <https://github.com/alkire-jeremy/CIT360/blob/master/ProjectTracker/app/src/main/java/com/example/android360x1/ViewProjects.java> | This class utilizes the ArrayList implementation of the List interface.  ArrayLists take up as much memory as is allocated for the specified capacity, regardless of whether elements have been added. Adding/removing from a position in an ArrayList that is anywhere but the end requires it to shift elements over.  ArrayLists have a capacity that is at least as large as the list size, but typically larger. It is always as large as the array used to store elements in the list. The size of an ArrayList grows automatically as elements are added to it. It is not fixed, like an array.  Unlike a normal array, in which methods are accessed with [], elements in an ArrayList are accessed by methods or iterators.  ArrayLists only support object entries, not primitive data types. If you insert a primitive variable into an array it is converted to an object.  In this case, I created two ArrayLists: An array list of Projects, and an ArrayList of TextViews.  Line 39 in this class declares an ArrayList of TextViews. Lines 40-47 add TextViews to the ArrayList.  Line 50 in this class declares an ArrayList of type Project. Lines 52-61 use a for loop to iteratively call the add() method of this ArrayList. Inside of the parameters for these add method calls, I am iterating through a normal array of strings titled “projects” which contains my JSON string output provided by AddProjects.java (which was previously provided by the server). In doing this, I call convertToJava() on each element of the projects array to convert it from a JSON string to a Project object, and then add it to the ArrayList of type Project, on line 54.  ALL of this is done to prepare for lines 64-77. This is where I use another for-loop to iterate through both ArrayLists, dynamically calling the setText() method of each view in the ArrayList of TextViews and passing a very complex string into its arguments that was created by calling all getter() of each Project object in the ArrayList of Projects.  This allows me to display to the user: A pretty, human-readable version of the massive JSON string sent by the server. | My code. |
| SystemTest.java | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/SystemTest.java> | This class does the same thing (lines 56-72), however it skips out on the ArrayList of Projects and instead uses a simple array, as no Projects are involved, or JSON conversions. |  |

**Diagram**: Use Case Diagram

**Description of Understanding:** Use case diagrams are used to display the requirements of a system. It is a very high-level representation of the system that doesn’t portray many specific details at all. The components of a use case diagram are actors, associations, system boundaries, and use cases. Actors are individuals who interact with a use case. They are named by nouns, and they trigger the use cases. They often provide input and expect output. Use cases are system functions, either automated or manual, and are named by verbs. Actors are linked to use cases, though not all use cases are linked to actors. Communication / associative links connect actors to use cases, to represent their relationship. System boundaries may outline the entire system, or simple modules. Actors lie outside of the system boundary, and use cases lie within it.

**Teaching Video:** None

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Example**  **Or Your code?** |
| project\_tracker\_use\_case\_diagram.jpeg | <https://github.com/alkire-jeremy/CIT360/tree/master/Familiarity%20Reviews/Diagrams/Use%20Case%20Diagrams> | The link on the left links directly to a .jpg image of my use case diagram.  This diagram is extremely simple, predominantly because use case diagrams reveal very little about application architecture to the viewer.  On the front-end of this application, everything looks EXTREMELY simple.  There is only one Actor (the user), and there are only two use cases:  - Add a project.  - View all projects.  Hence, to the user, this is all magic that is happening in the back end, and thus this diagram displays very little. | My diagram. |

**Diagram**: Sequence Diagram

**Description of Understanding:** A sequence diagram demonstrates the interaction between messages and objects within a system. It displays these relationships in a sequential manner, so that people can clearly see how messages flow throughout the system. Components of a sequence diagram include objects, messages, and sequence numbers, which indicate how methods are called one after another.

**Teaching Video:** None

|  |  |  |
| --- | --- | --- |
| **File** | **What should I be looking for?** | **Example**  **Or Your code?** |
| <https://github.com/alkire-jeremy/CIT360/tree/master/Familiarity%20Reviews/Diagrams/Sequence%20Diagrams> | This diagram displays the sequence of events that occur in my ProjectTracker application. It displays the most critical methods that are called to navigate between views and send data between different components of the program. While it does not list *every* function called, such as those necessary to parse data, it does list those used to send data from one place to another. It states how the user starts the application, what function is called when they select a view, and how data gets into the database and back. | My code. |

**Coding Topic:** System Level Tests

**Description of Understanding:** System level tests test each component of a program (ensuring its functionality as a whole) rather than testing a single part. Complex systems that consist of multiple components such as a database, web application, web server, and more are tested individually to verify their functionality. A system level test must conduct multiple tests in order to verify complete system functionality. A system level test should be able to be executed by someone who knows little to nothing about the inner workings of the system. It typically runs through a script, and the tester follows up upon running this script by documenting the results of the test in a spreadsheet.

**Teaching Video:**

**Starting at:**

**Also Integrated with:** Hibernate, Servlets, HttpURLConnection

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| MainActivity.java | <https://github.com/alkire-jeremy/CIT360/blob/master/ProjectTracker/app/src/main/java/com/example/android360x1/MainActivity.java> | This class contains the method bound to the System Test button in the activity\_main.xml file.  Lines 55-74 compose this method, titled “clickTestProject()”.  This method calls the encode() method statically from URLEncoder and uses it to create a large String that will be sent to the server as a POST request.  This request will contain the information necessary to create a simple test Project object that will be inserted into the database.  The server should accept this request, create an instance of the Project class and insert the object into the database. From there, it should take the initial request, convert it to a JSON string, and send it back to Android.  The response will be displayed on SystemTest.java |  |
| SystemTest.java  (ANDROID) | <https://github.com/alkire-jeremy/CIT360/blob/master/ProjectTracker/app/src/main/java/com/example/android360x1/SystemTest.java> | This displays the results of the SystemTest, although the test itself is initiated by a user pressing the “System Test” button on the main view of the Android application.  This is of course strictly for people testing the system, and not for users, but this isn’t an application being developed for deployment. | My code. |
| SystemTest.java (ECLIPSE) | <https://github.com/alkire-jeremy/CIT360/blob/master/src/main/java/servlets/SystemTest.java> | This is the servlet that handles all the back-end operations necessary to run the test. It accepts the POST request, creates the Project object, inserts it into the database, converts the original request to a JSON string, and sends it back to Android. |  |
| Systems Level Test (Documentation).xlsx | <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Reviews/Documentation/Project%20Tracker%20(Systems%20Level%20Test).xlsx> | This contains written documentation of what steps one must take to execute the system test, and what results they can expect to see. This is where they document pass, fails, and other pertinent information. |  |

**Diagram**: Use Case Document

**Description of Understanding:** A use case document offers a story of how an actor utilizes a system (via use cases) to achieve its goals. It provides a detailed step by step description of how the actor will use the system to achieve its desired outcome. Unlike a use case diagram, it provides a written representation of this information, rather than illustrated. It is typically in table format and is composed of a general description of the use case, its name, detail about the author and date in which this information was documented, a list of actors, preconditions, postconditions, an explanation of flow, alternative flows, exceptions, and requirements.

**Also Integrated with:** MVC, Collections

**Teaching Video:** None

|  |  |  |
| --- | --- | --- |
| **File** | **What should I be looking for?** | **Example**  **Or Your code?** |
| <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Reviews/Documentation/Project%20Tracker%20(Use%20Case%20Document).docx> | The link on the left links directly to a .doc file, which is my use case document for my entire ProjectTracker application.   The document specifies the following:   * Actors. * Three preconditions. * Three postconditions. * Flow. * Alternative flows. * Exceptions. * Requirements. | My code. |