**Familiarity Review**

**Name:** Jeremy James Alkire

**Date:** March 16th, 2019

**Week:** 10

**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:**

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| CreateData.java |  | This class creates a new session, session factory, and transaction, and then proceeds to create an object instance of each Java bean designed to be converted into a table. Lines 22-26 display the construction of an instance of the Employee class, and lines 28-31 display the construction of an instance the Student class. Both will automatically create corresponding tables and insert data into those tables according to the values of the static variables of each object. Lines 33-36 simply insert a new row into the table, given that the table (upon code execution) should already exist. | My code. |
| GetAllData.java |  | Lines 16-18 demonstrate the creation of a new session, session factory, and transaction. Lines 20-22 demonstrate the creation of two different lists, populated by the results of a generated query. Lines 28-32 use a for loop to iterate through the employeeList collection (which should contain equivalent contents to the employee table) and calls the class’ getter methods to display data for each object entry in that table. Lines 34-37 do the same for the studentList and student table. | My code. |
| Employee.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. |
| Student.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. |

**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:**

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| FetchSite.java |  | This class utilizes the unique methods of the HttpUrlConnection subclass, in addition to a few methods from the UrlConnection class. The program itself requests the source code from a URL, creates a new folder locally, writes the HTML source code to a new .htm file, and then launches it in a web browser. In addition to this, the sendGET() method in particular utilizes HttpUrlConnection methods specifically. Lines 59-65 open a new HttpUrlConnection, and lines 67-73 call get methods from the connection, displaying it to the console. | 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:**

**Starting at:**

**Also Integrated with:**

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| ExecuteThreads.java |  | This class constructs two Thread objects and one Runnable object (lines 10-11), before calling a few class methods from each and configuring the objects’ variables. The first thread plays instrumental audio, the second plays vocal audio. The two threads (subclassed from Thread) are named, before the instrumental thread is started via the start() method. The thread that implements runnable (delayThread) is executed with an executor, on lines 29-31. Finally, the last thread (vocal) is again started with the start() method. In this program, two threads (instrumental and timer) run concurrently, and the final thread (vocal) waits for the second (delay) thread to complete before beginning. From that point it runs concurrently with the first (instrumental) thread. The two create a song that plays both instrumental and vocal audio after waiting 20 seconds. | My code. |
| Exec.java |  | This class implements the Executor interface, and functions as an executor. It has one method only: execute(), which takes a runnable object as an argument, and calls its run() method. execute() does not run on its own, and is called non-statically in the former ExecuteThreads class. |  |
| Delay.java |  | This thread class implements the Runnable interface, and merely counts a rough estimate of seconds passed. It counts up to whatever quantity of seconds is set by the setTime() method. It does not run on its own, and is called non-statically in the ExecuteThreads class. | My code. |
| Audio.java |  | This thread class is a subclass of Thread, as it extends Thread. It simply plays whatever .wav file is referenced in the songPath variable, which is intended to contain a URL. It does not run on its own, and is called non-statically in the ExecuteThreads class. | My code. |

**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 has to implement the Servlet interface.

**Teaching Video:**

**Starting at:**

**Also Integrated with:** Threads

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| ServBase.java |  | This class is an extension of HttpServlet. The page displays a single checkbox form that lets the user choose any of four options: RZX, SOUL, CHRISTMAS, or STOP MUSIC. The first three play different audio files, and the final option stops the audio. Each audio file is played via a thread created from the Audio class. In order for this servlet to run, it has to be deployed to a server. In my case, I ran it on Tomcat v9.0. | My code. |
| index.css |  | Referenced inside of the HTML that is embedded into ServBase.java is the index.css file, which is used to format the front-end of the servlet. | My code. |
| web.xml |  | This XML file is used to reference the ServBase servlet both by name and location. It additionally maps the servlet to a particular URL pattern, so that it can be referenced by an action value or typed into a web browser. | My code. |
| PlaySong.java |  | This class creates and manages the threads that play the audio in the servlet. The servlet is designed to allow the user to seamlessly switch back and forth between different songs, depending upon which checkbox they’ve checked and submitted. In order to accomplish this, a ‘running’ atomic boolean variable was created in the Audio thread class. To prevent any IllegalStateExceptions(), thread states are checked each time a request is submitted to the server. If the thread has been started already, the running flag is set to ‘true’ to resume the thread (with its new parameters). If it hasn’t yet been started, it’s started with the start() method. Each time a request is sent to the server to play a new song, all currently running audio threads are stopped by setting their running flag to ‘false’, before being resumed with their new settings. | My code. |

**Coding Topic:** Application Controller Pattern

**Description of Understanding:** The application controller pattern is a design pattern used to centralize and modularize action and view management of an application. It enables reusage of action and view-management code, improves request-handling extensibility, and improves code modularity and maintainability. Ideally, the client sends a request to the server, which gets delegated to the application controller. The application controller invokes the target, resolves the mapper, and uses a map to determine what response to provide for the request. Upon determining which response is appropriate, the response is sent back to the client in the form of the corresponding view.

**Teaching Video:**

**Starting at:**

**Also Integrated with:** Servlets, JSP, Collections

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| index.jsp |  | This file functions as the first view that the user sees on the client end of the program. Unlike a standard servlet (which embeds HTML in Java), this is a JSP, which does the inverse: Embedding Java in HTML. It’s a simple and short file that only contains 15 lines of code. This displays a form to the user containing a single text field and a submit button. Line 10 defines the form action as “ClientServlet” which forwards their request to the ClientServlet, which handles it from there. | My code. |
| ClientServlet.java |  | This class is a servlet. It simply takes a request, creates a new instance of the ApplicationController, and asks it to process the request. Processing, in this case, means using a map to determine which page is appropriate for the given request. Upon receiving this from the application controller, a dispatcher is used to forward the page and response back to the client. | My code. |
| ApplicationController.java |  | This class functions as the program’s application controller, and has but one method: process(). Process accepts a single string argument (containing the client’s request), and passes that request on to CommandHelper, which references the commandMap to determine which page response is mapped to this particular request. Upon receiving this information from CommandHelper, it returns the response to the caller. | My code. |
| CommandHelper.java |  | This class acts as the application’s mapper. It creates a map that is intended to map anticipated requests to their corresponding responses. It has a single method called getPage() used to serve those responses up to anything that provides it with a request. | My code. |
| Dispatcher.java |  | This class takes three string arguments: The request, response, and page. Upon creating a new request dispatcher, the page is included in the response, and forwarded to the client. | 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?** |
| Converter.java |  | This class uses the QCJSON library (I imported the entire package into my project directly), and converts a Java object into JSON, and compatible JSON strings into the same type of Java object. Line 25 creates a blank .json file to be written to in the future. Line 28 creates an instance of a Java object (MiniSchnauzer), and lines 31-34 configure the attributes of that object via its setter methods. Line 37 converts this object into a JSON string. Line 40 writes this JSON string to the file created on line 25. Line 43 reads the JSON string from the file, and lines 47-49 convert it back into a Java object. Lines 56-60 display the attributes of this Java object via its getter methods, to verify whether or not it looks the same as the data saved in the JSON string (and file) earlier. | My code. |
| MiniSchnauzer.java |  | MiniSchnauzer is a simple Java bean object. The class is serializable to allow for saving and conversion to JSON. It has private variables, and public getter & setter methods to configure their values once an instance of this object has been created. It is used in Converter.java. | My code. |
| JacksonTest.java |  | JacksonTest uses the Jackson JSON library. This class, unlike Converter.java, begins by referencing an existing .json file, and using it to generate a corresponding Java object. Line 24 references an existing .json, assigning it to a File variable. Line 30 creates the Java object with the JSON string read from this file. Inversely, the contents of the testJavaToJson() method beginning on line 51 does the opposite. Line 52 creates an empty .json file, and line 58 uses a mapper to convert it to a Java object, only to save it directly to the file created earlier. | My code. |
| Employee.java |  | This class is a simple Java class, intended to be instantiated in JacksonTest for sake of testing its ability to convert Java objects to JSON, and vice versa. | My code. |

**Coding Topic:** JUnit

**Description of Understanding:** JUnit is a testing framework for the Java programming language. It promotes the idea of testing before coding: Ideally to encourage a pattern of testing, followed by coding, and so forth. Continual testing throughout the development process can help ensure stability of the program, and help prevent the programmer from creating a pile of errors that they only discover after a rigorous few hours of programming (without testing). It saves time and resources, both. JUnit is linked as a jar file, and resides under package org.junit for versions 4 and later.

**Teaching Video:**

**Starting at:**

**Also Integrated with:** JSON & QCJSON

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Git Link** | **What should I be looking for?** | **Sandbox or Your code?** |
| JacksonTest.java |  | This class is contained in the src/test/java/Jackson folder, and uses @Test annotations to indicate what parts of it are to be tested with JUnit. Running this class as a JUnit test will test it, although it currently isn’t running any assertions, so its merely testing if it runs successfully. | My code. |
| QcjsonTest.java |  | This class is contained in the src/test/java/qcjsontest folder and also uses @Test annotations to indicate which parts of it are to be tested with JUnit. On the contrary to the above example, this one runs a single assertion method: assertEquals(). This method compares two strings and determines whether or not they are equal to each other. If they are, the test succeeds, if not, it fails. It uses this method to compare the ideal JSON string (hypothetically) to the actual JSON string generated from a Java object, and written to a file by Converter.java. I definitely could have ran a few more tests for either one of these classes, but have been pressed on time – which is why I did the bare minimum here. But I can promise that I understand the concept of JUnit testing thoroughly. | My code. |

**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** | **What should I be looking for?** | **Example**  **Or Your code?** |
| <https://github.com/alkire-jeremy/CIT360/blob/master/Familiarity%20Requests/Request%201%20-%20Week%204/Diagrams/Use%20Case%20Diagrams/use_case_diagram_athlete.jpg> | The link on the left links directly to a .jpg image of my use case diagram. | My code. The same code used to demonstrate the MVC architectural pattern. |