**Familiarity Review**

**Name:** Jeremy James Alkire

**Date:** March 16th, 2019

**Week:** 10

**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?** |
| ThreadsRevisted.java | <https://github.com/alkire-jeremy/CIT360/blob/master/FR2/src/main/java/threads/ExecuteThreads.java> | This class singlehandedly demonstrates simple multithreaded programming. It creates two different Threads, and demonstrates the two different approaches available for creating each. The first thread created is called Evens, and is subclassed from Thread. The second thread created is called Odds, and is an implementation of Runnable.  Both threads are executed within the main() function of the program, however: Evens is executed via a call to the method start(), and Odds is executed via the execute() method of an executor, called SimpleExecutor.  When running correctly, the two threads run concurrently, resulting in a (hopefully) seamless counting effort from 1-100. It’s possible that they may display out of order, but in most cases it’s a simple count to 100. Either way, it should be displaying all numbers in that range, both evens and odds, without issue. If one thread is commented out, it will only display evens, or only display odds, depending upon which was commented out. | 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?** |
| AssertionsExample.java | <https://github.com/alkire-jeremy/CIT360/blob/master/FR2/src/test/java/jackson/JacksonTest.java> | This class is a simple demonstrate of JUnit. It utilizes various JUnit assert methods, including assertEquals(), assertTrue(), assertNotSame(), and assertNotNull(). When this test is run, it should only result in one failure, as the comparison between strings hashOne and hashTwo are not equal, on line 24.  It creates three strings, two of which are identical, and one of which is different. It tests the first and second against each other (unequal), which should result in a test failure, in addition to comparing the first and third (equal) which should pass the assertEquals() test.  This class also solves a simple addition math problem. It has a hardcoded answer to be compared against the actual answer. When correct, a boolean variable called flag it set to true, and that variable is passed through as an argument for the assertTrue() method. The answer should be correct, meaning this should pass the assertTrue() test.  Additionally, this class creates three objects, none of which are identical, and uses the assertNotSame() method to determine whether or not any of these object references point to the same object. They don’t, and thus it passes the test. The final test it makes, is calling the getter() methods of one of these objects, and determining whether or not its variables have null values. They do not, and hence this test is also passed. | 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.

**Note:** You requested that I put my system level test into table format, although, from looking at it again, it seems to already be in table format, in an excel spreadsheet. It was very similar to the example you linked to me in your feedback, on the rubric. So I modified my existing table to look a little bit more like it.

**GitLink:**