MEMORANDUM

To: Prof. *David Green*

From: *Hunter Holmes hholmes1@uab.edu*

Subject: EE 333 P3 Written Report

Date: *10/15/2018*

Summary

P3 is the third major assignment of Engineering Programming Using Objects in the Fall of 2018. This project has several major goals that contribute to the overall understanding of Object-Oriented Design. One goal was simply to expand on the work done in previous projects for the course. P1, P2, and now P3 have all been building on the modeling of a room with air handlers and a controller that decides how to maintain desired temperatures. P3 seeks to add a blower, a new exception, and new tests to verify the new features.

Aside from adding to the room's temperature controls, this project is designed to teach skills of Object-Oriented Design which are key to this course. Using the Java programming language, which is very heavily object oriented, real life objects such as rooms, heaters, blowers, and more can be modeled as implementations of classes those types. While objects like rooms and clocks are useful for this problem, many other problems can be broken into its respective "objects" and solved using similar techniques.

While modern programming can lend itself to some complex problems, there are also the conveniences of modern programming tools. One that is readily available is the NetBeans IDE. NetBeans and other development tools like it contain tools to aid in the development process. These tools do not write solutions to problems, but they offer a great deal of guidance. P3 is designed to increase interaction with tools so they can be comfortable used to solve future problems.

Problem Solution

After reading and understanding requirements for this assignment in the P3.pdf [1] it was time to prepare the emolument in which the problem would be solved. Before any programming could begin, this assignment involved creating a Narrative Log [2] which details all steps taken to build and test the components. The Narrative Log includes screenshots to provide details of tools that were used.

A new java application was then created in NetBeans with the title "hholmes1_P3". Since P3 takes the structure of P2 and adds new features, files from P2 were moved and refactored into P3. This included all source code and tests written for P2. For clarity, TestP2 was renamed to TestP3. To provide verification that the new project was a stable base for P3, all unit tests were run and passed as a part of new P3. [2.2 – 2.4]

With a functional base established, the processes of building and testing could begin. The first new class to be created was Blower. In previous iterations of this design the controller would simply turn the heater on or off. The state of heater would have a direct effect on the room it was a part of. In this new design a blower would move air from the heater to the room. The heater could only change the temperature in the room if the blower was on. The blower could only be turned on the heater was already on. Fortunately, the properties of the blower were similar to the heater. The blower became and object that is added to the room and the heater became an object that is added to the blower. The blower was also given the property of output temperature. If the heater and blower are running the blower will output hot air and warm the room. This output property makes the system more flexible. If the room needed to be cooled, simply connect an air conditioner and the blower could output cool air. [2.5]

With this new addition of the blower some tests had to be updated and some had to be created. A Junit test for the blower was added. This test verifies the blower's UID, operation in relation to the heater, and toString function. Since the controller can now connect to the blower and only turn it on if the heater is on it needed an updated test as well. [2.6-2.8]

Some updates were made to TestP3 as well to test the blower's capabilities. When TestP3 was run, a NullPointerException was encountered. The exception error message indicted the line at which the exception was thrown, and a breakpoint was set at this line for the debugger. Debugging the project highlighted the issue very quickly. TestP3 needed information about a heater that was not yet added to a blower, thus the heater was null. Implementing the blower's add method solved the issue. [2.9-2.10]

As part of this assignment a custom MissingComponentException was created to be thrown when a controller connection was left open. If a heater, blower, or temperature sensor were not connected to a controller it would be difficult or impossible to detect and maintain the room's temperature. Therefore, a special exception is needed to detect a connection that is not made and end the program before more errors occur. This is also useful because it provides specific information that is directly relevant to this problem. By using this exception, it is also possible to log the error so that it may be reviewed for debugging later. To have this exception reach the compiler level and stop the build process, the keyword "throws" must be in the header for all methods that need to throw the exception up to the next level.

The Controller unit test was used to validate the functionality of the MissingComponentException. First it was necessary to prove that no exceptions were thrown if all components were connected properly. Next taking each object and skipping it in the connection process one at a time it was shown that the custom exception message could be used to indicate which component was not connected with accuracy. Currently the MissingComponentException only deals with one component missing at a time. [2.11-2.14]

The Room class needed an update so a blower could be added. First the Room's add(Heater) method was replaced with an add(Blower). The room was also modified to accept an output from the blower instead of a steady 95 degrees when the heater is on. This last step contributes to overall flexibility. To test the new Room class the RoomTest was updated to check that the heater does not influence the room when the blower is off. [2.15-2.16]

Once again unit tests were performed to check that all updates made to classes and unit tests had not broken key components of the project. [2.17]. Using the NetBeans “Generate JavaDoc” html files were created in the style of JavaDoc that is found with any of the classes in the JDK. The created JavaDoc files were reviewed and documentation changes were made until JavaDoc appeared satisfactory. [2.18-2.19]. With all the modifications there was a possibility that P3 would no longer pass its own test, TestP3. Once again TestP3 was run and all state tests were passed. [2.20]

By default NetBeans is set to create a .jar file for a java application when it is compiled. To verify this I inspected the properties of the project and saw that the box was checked to build a jar file from the project. To ensure proper operation the NetBeans tools Clean and Build, and Generate Javadoc were used. With the JVM installed it is possible to run this .jar file from the command line. Changing directories to the dist folder of the hholmes1-P3 allows access to the .jar file. From Command Prompt (Windows) the jar file was executed and the output was redirected to a .txt for neat packaging of a demo run of the program.[2.21-2.22]

References

1. David Green *P3.pdf* 20180915 - Original version

2. Hunter Holmes *Narrative Log hholmes-P3.pdf*