**Comp 453 - Intermed Obj-Orient Devel**

**Group 4**

**Project 4**

**Dec-8, 2018**

**Report**

**1. Development Journey**

**Beginning Stage**

* At this stage, our group decided to apply the State pattern to this project because we noticed this project had separate states with different functions attached to each state.
* We created a repository in Bitbucket and started implementing the project based on the state diagram that was designed during the in-class activity.

**Development Stage**

* We created classes based on the state diagram. In these classes, we implemented each state: Stopped, Counting, Running, Alarming.
* We decided that each state would be a class implementing one common interface. This interface ensured that each state would have the ability to process any upcoming events declared within the interface.
* The four states were connected and interacted around one state-management class; this class received the events from the user and sent it to the appropriate state. Then, any state that received the event could decide the next step based on the application scenario.
* Then, we the state management class with the activity. In our project, activity interacted with one facade-pattern implemented class which was in charge of creating other necessary components.
* After implementing every fundamental feature, we applied testing to ensure the correctness of the project.

**Final Stage**

* We started refactoring our codebase. We removed unnecessary classes, all dummy print outs, we also added in-line comments and generated a Javadoc.

**2. Version control and workflow**

* Our group used Bitbucket and Git as our version control system.
* We used a workflow similar to GitFlow. Instead of feature branches, a branch was created for each team member.
* Individuals worked on their personal branches to complete project requirements. They then submitted pull requests to share their changes with the team.
* Changes were finally merged with the master branch.

**3. Testing**

* In this project, we mainly focus on unit testing, which was applied to testing every state of the program.
* We also used mock dependency methods to simulate the activity and time models, so it would be easier to test the state diagram.
* Unit tests were written for each functional requirement. In addition to helping us detect bugs, this unit testing process led to a fine-grained understanding of the requirements and a stable application that could be easily refactored.

**The relationship between your extended state machine model from this project and our actual code**

* The differences between our model and code and models and code in general is that our model is practically a skeleton framework for creating, adjusting and improving upon the code of our project.
* Every piece and connecting line of the state diagram gives us a general idea of where to start, where to forge in the middle, and where to end. It would be more effective to model first to plan your thoughts and coding actions in an easier and, arguably, more efficient way.
* We would not consider making any changes to our model now that we have our code. The model and the code seem to complement each other just fine.