Angel Ramirez

To implement my homework 1 DePaCoG as a plugin specifically for IntelliJ, I made it using one Action class and one window (DialogWrapper) class. Upon building my plugin by running “gradlew buildPlugin”, the plugin is built and stored in the “build/distributions” directory where it can then be installed via the normal plugin installation option “from a disk”. Once done the plugin will be the first option found under “tools”, should be found easily with the distinct logo.

The plugin begins by prompting the user to select a design pattern from the available patterns and then a directory where the user would like the design pattern files to be placed. Then, a window pops up asking the user to input information in CSV format. I made the plugin’s internals work with CSV because I thought it would be the quickest way to get user input all at once and then allow the parsing function for the selected design pattern to take care of the rest to create the necessary files. Having that said, the code uses code from homework 1 to organize the design patterns in the same way as homework 1 and store the respective information similar to homework 1. It made the most sense to me at the moment so I used my classes for the implemented design patterns and the internal code generation with the change that the user gets to select where to store the files and all questions for design pattern are asked at once. One very important note about my implementation is that the files are strangely only generated when the user clicks out of IntelliJ and then clicks back in. Same thing for the testing window when running the plugin project (not the installation). Another strange phenomena has been the log files when the plugin is installed. The log files only seem to appear sometimes when the user clicks out of IntelliJ and then back in (same as to get the files to show). I do not understand why but is a very important note when using this plugin.

The transition from a terminal based implementation to a plugin was difficult for me at first. Research had to be done and examples needed to be looked at so that I could come up with my plan of action. As previously stated, I ended up using one Action class (DePaCoGAction) and one pop up window class (DePaCoGPromptWindow which extends DialogWrapper). I reused the DePaCoGPromptWindow for the main window and the user input window. My reason for this was to simplify the amount of classes needed to be made and to reduce the amount of moving parts to keep track of in my code. This helped me organize myself and understand how things worked in my code. The DePaCoGPromptWindow class is only in charge of taking user input and to know which design pattern to make and then the internal design pattern classes (AbstractFactory, Template, etc…) take charge of parsing out the input and generating the correct code similar to homework 1.

The pros about my implementation is that I was able to figure out something that worked for me and I understood what was going on. The cons are that the plugin assumes that the user input is all correct and that I reused a great majority of my code from homework 1. The plugin does not prompt the user to retry in order to make a successful run and actually create the files without having the user rerun the plugin. Though not a big con but still tedious to type everything in if the user decides to generate a lot of files with a lot of information. The other con about me reusing most of my code from homework 1 I feel that is not good practice. I just did not know of another way at the moment. The positive side of this, is that I knew that I had to modify my code

from taking in user input from the console to taking user input from a text area so it was not like I had copied and pasted completely from one project to another. I had to modify this code a good amount to adapt it to a plugin instead of the console.

Throughout this project the most troublesome part has been the setting up. I had a hard time setting up logback and being able to use my configuration file. I am still unsure why the transition from a console project to a plugin project caused so much problem. The good thing was that I was able to solve it on time and learned how to set it up in the future.

The tests for the plugin test that the new parsing methods work for each implemented design pattern, the correct number of files are created in the right path, and that the correct number of functions are found in the classes being generated. I thought of how to test the plugin and its behaviors. This was what I could come up with at the time of programming this plugin.