SE 450: Object-Oriented Software Development
Chatbot Project Requirements
Due May 30, 2018

Introduction

The SE 450 project gives you the opportunity to apply the object-oriented software development skills you have acquired during the course to a significant but moderately-sized application. The following sections describe the general behavior of the application, the simulation, and the deliverable requirements for the project.

The final project for this class is a simple *chatbot*, a program which mimics the way a person would engage in simple conversation with another (human) person. The original and best-known example of a chatbot is ELIZA, developed by Joseph Weizenbaum as a parody of a therapist. A good discussion of ELIZA can be found at http://en.wikipedia.org/wiki/ELIZA.

You will develop a chatbot using good software engineering practices, including:

- Unit testing of the application classes.
- The ability to dynamically define the chatbot conversation using a configuration file.
- Use of design patterns to simplify and standardize the code.
- Proper internal and external documentation for the application.

Project Description

The chatbot application should have the following features:

- The program must be started and run from the command line.
- The program must read the complete vocabulary (participant names, queries, answers, etc.) for the conversation from a configuration file.
- The program must take a single command-line argument, the name of the configuration file.
- The precise format of the configuration file will evolve over the remaining weeks of the course as new topics are introduced. That is, the format of the configuration file for the final assignment for the project will differ from the format of the first configuration file that will be provided.
- Students will be provided with sample configuration files for each assignment. However, the instructor will use configuration files with the same format as the sample file, but with ***different*** data, to test and grade the submitted assignments.
- After the initial program banner is displayed, the program must provide the user with options to either (a) display the complete conversation configuration file and terminate or (b) proceed directly to the conversation.
- The program must then start the conversation with the user using one of the queries defined in the configuration file.
- Thereafter, the program must respond to replies from the user either with a query or a statement.
- When the user enters a specified terminator (e.g. "Bye!"), the program must respond with a specified terminator and the end.

Coding and Deliverable Requirements

The Final Project should demonstrate your ability to apply the concepts, techniques, and skills acquired during the course to a fully-integrated application. The general requirements for the project are:

- The application must be written in Java version 8 or 9.
- As with all previous assignments in the course, you must use the versions of the tools specified in the first class in order to ensure that your code will be compatible with my testing environment.
- The application must be a stand-alone Java application, not an applet. It should be started and configured from the command line. The output should be displayed on the command line.
- Only features and capabilities that are part of the Java SDK may be used in the application. The application *may not* use third-party components such as databases or class libraries.
- The application must use the MVC architectural pattern.
- The application must be grouped into at least three packages.
- The application will probably consist of approximately 1000-1500 lines of Java code, excluding comments.
- The application must use at minimum four design patterns that we have discussed in class. You may *not* use patterns built into the Java API, such as `Observer` and `Observable`. For use in the MVC pattern, you must code the Observer pattern yourself. Note that this means you only need to incorporate three other patterns, since use of Observer is required to use MVC.
- The application will be evaluated based on program correctness (compilation and execution), quality of code and documentation, adherence to standards, and satisfaction of requirements. The application need not be flashy or innovative, though you may expand on the basic requirements as long as the basic requirements are met satisfactorily.
- The application should conform to the Java coding conventions.
- The application should conform to Java documentation conventions. The `javadoc` utility will be run on your code to demonstrate that proper documentation is generated automatically and the documentation will be reviewed for completeness.
- You will be expected to keep a development notebook, which will document your design artifacts, design explorations, and decisions during the project's development. These are described in the *Development Notebook Content* section, following.

Development Notebook Content

Note: All diagrams must conform to the UML notational conventions presented in class. If the tool you use varies from this, please document it clearly.

- *Cover Page*. Include project title, submission date, and developer name.
- *Brief Project Summary Statement*. Include a brief project summary statement, describing the general features of your implementation. These should include a bulleted list of all the design patterns you used and how and where they were used, any extensions to the original requirements you incorporated into your implementation, and any other important aspects of your implementation that you wish to note. *This item should be no more than two pages in length.*
- *Sequence Diagram*. You will write sequence diagram and include this in your development notebook. You may include any other design scenarios and sequence diagrams you generate for the project, as well.
- *Design class diagram*. Include the design class diagram for your project. Be sure to include significant class attributes and methods and all significant class relationships, such as navigability, dependency, specialization, implements, uses, and any forms of aggregation. *Note*: I am not requiring you to create a

diagram which is 100% consistent with your final implementation. You should include any new classes, interfaces, and packages in your design class diagram, and I expect you to **mostly** reverse engineer changes in your code back to the diagram.

- *Other Design Notes*. Keep track of any other design issues that arise during development, such as specific design decisions, rationale for use of design patterns, failures, successes, etc.
- Although no specific style guidelines are being enforced, the design notebook must be presented in a neat, legible, and consistent format. Note that 'Notebook Quality' is one of the grading items for the final project.

Grading

The following table provides a detailed breakdown of final project component values.

Project Item	Item Components	Point Value
Application	• Satisfies requirements • Correctness (compilation & execution) • Quality of code & documentation • Adherence to standards	**60**
Development Notebook	• Brief Project Summary Statement • Use Case (optional) • Sequence Diagram • Design Class Diagram • Design Notes	**35**
Notebook Quality		**5**
Total		**100**

Submission

Zip up your project folder and development notebook into a single Zip file and submit to D2L by 6:00 PM on 30 May 2018. Do NOT include your build or docs branches in the Zip file. This means: do not submit your class files. Tell me if you're using Eclipse or IntelliJ. If you properly submit the project file, there should be issues for me to just open up the project in my IDE and run the project.