SE 450: Object-Oriented Software Development
Final Project — Due June 6, 2018

This is document concludes the requirements for your Final Project. There are no other homework assignments that you are required to complete. Please make sure to read all of this document as there are some changes to the final Development Notebook requirements.

Part I

1. The chatbot must read its conversation vocabulary from a configuration file. A configuration file is included with this assignment. Your program must be able to read this file, parse it, and use the vocabulary in the file to generate the conversation for the chatbot.
2. Using the configuration file provided with this assignment, the output for the program should look like the following for a **negative** response to the query *`Do you wish to display the configuration file (y or n)?`* (Statements preceded by a '>' indicate user input):

```
c:\> java Chatbot Chatbot_Vocabulary.xml
Welcome to Chatbot v. 0.1
Do you wish to display the configuration file (y or n)?
>n
Hello Luai!
My name is Eliza.
How are you?
>I am fine, thank you.
>How are you?
I am fine, thank you.
>Bye!
Goodbye!
```

3. Using the configuration file provided with this assignment, the output for the program should look like the following for a **positive** response to the query *`Do you wish to display the configuration file (y or n)?`* (Statements preceded by a '>' indicate user input):

```
c:\> java Chatbot Chatbot_Vocabulary.xml
Welcome to Chatbot v. 0.1
Do you wish to display the configuration file (y or n)?
>y
Configuration file Chatbot_Vocabulary.txt:
<user>
      <first_name>Luai</first_name>
      <terminator>Bye!</terminator>
</user>
<chatbot>
      <first_name>Eliza</first_name>
      <terminator>Goodbye!</terminator>
</chatbot>
<query>Hello ${first_name}!</query>
<query>How are you?</query>
<response>Hi ${first_name}!</response>
<response>I am fine, thank you.</response>
<statement>My name is ${first_name}.
```

4. **IMPORTANT!** I may be testing your code using both the sample file provided to you AND with a different test configuration file. The other file will have the same structure, order, and tags (such as <response>) and will use only one variable parameter, ${first_name}, but may

change the details of the text. For example, I may use “Je m’appelle” instead of “My name is” and your program should work just fine. In other words, don’t hard-code your conversation!!!

Part II

5. You will enhance the chatbot to intelligently handle two types of responses from the user: fixed, *expected* responses (“How are you?” “I am fine, thank you”) and *unexpected* responses (“How are you?” “I’m doing OK”). Note that you are already doing this in a very basic form in order to determine if the chatbot needs to reply “I don't understand” and terminate.
6. *Required Design Pattern:* Refactor your code to use the **Strategy design pattern**. Encapsulate your existing code for an expected response into one strategy and your existing code for an unexpected response into another strategy.
7. Next, refactor your code so when a response is received from the user, the chatbot decides if it is an expected or an unexpected response then chooses and uses the correct strategy.
8. Make sure your program is working properly after this refactoring and BEFORE trying to add additional, new functionality. Remember, refactoring means making internal modifications to a program WITHOUT changing its observable behavior! Don’t make enhancements until the refactored chatbot is working according to the specifications. Once you have the refactored version of the chatbot working properly, you can move on to enhancing its ability to respond to unexpected user responses.
9. Implement the following enhancements to the chatbot’s strategy for handling unexpected user responses when the chatbot’s asks the question “How are you?”:
    - Give the chatbot a series of predetermined, fixed answers to unexpected user responses. The simplest is “I don't understand” and continue execution. You should also add two new chatbot responses of “Why do you say that?” and “Could you restate that so I understand better?” You may use any method for choosing one of these predetermined responses for a given unexpected user response: cycle through the list, choose one at random, etc.
    - Give the chatbot limited ability to echo the user’s input. Specifically, if the user responds “I am…” and does not use the fixed response “I am fine, thank you” or the user responds “I’m…”, take the predicate of the sentence (the ‘…’ part) and have the chatbot reply “I understand that you are…”. For example:

```
c:\> java Chatbot Chatbot_Vocabulary.xml
Welcome to Chatbot v. 0.2
Do you wish to display the configuration file (y or n)?
>n
Hello Luai!
My name is Eliza.
How are you?
>I’m doing OK.
>How are you?
I am fine, thank you.
I understand that you are doing OK.
>Bye!
Goodbye!
```

Part III

In order to make the chatbot more realistic, it must be able to make small talk with the user. Almost everybody enjoys movies, so add the following questions and answers to the chatbot’s vocabulary:

1. The chatbot should ask, “What is your favorite movie?” User will respond with a movie name. The chatbot should maintain a short list of movies that you have seen. If the user response is in the list, the chatbot should say “I’ve seen that movie.” If the user response is not in the list, the chatbot should say “I haven’t seen that movie.” The list of movies should be included in the chatbot vocabulary file under a new tag called ‘chatbot_movies’. For example:

```
<chatbot_movies>
        <movie>Saving Private Ryan</movie>
        <movie>Big</movie>
        <movie>Cast Away</movie>
</chatbot_movies>
```

2. The chatbot should ask, “Who is your favorite actor?” User will respond only with a person’s name. The chatbot should maintain a short list of actors that you know, along with whether you like the actor or not. If the user response is in the list, the chatbot should say “I like that actor” or “I don’t like that actor.” If the user response is not in the list, the chatbot should say “I don’t know that actor.” The list of actors should be included in the chatbot vocabulary file under a new tag called ‘chatbot_actors’. For example:

```
<chatbot_actors>
        <actor>
                <name>Tom Hanks</name>
                <eval>like</eval>
        </actor>
        …
</chatbot_actors>
```

3. Same as item 2, but for actresses.
4. The chatbot should ask, “Who is your favorite director?” User will respond only with a person’s name. The chatbot should maintain a short list of directors that you know, along with whether you like the director or not. If the user response is in the list, the chatbot should say “I like that director” or “I don’t like that director.” If the user response is not in the list, the chatbot should say “I don’t know that director.” In addition, if the user response is **not** in the list, the chatbot should ask, “Tell me one film that director made.” User will respond with a movie name. If the film is in the chatbot’s list of films (item 1), then the chatbot should say, “I’ve seen that movie.” If the user response is not in the list, the chatbot should say “I haven’t seen that movie.” The list of actors should be included in the chatbot vocabulary file under a new tag called ‘chatbot_directors’, with a structure similar to that of ‘chatbot_actors.’

Development Notebook Content Update

This list replaces the original list in the Chatbot Project Requirements document. Note that some requirements have been dropped.

- *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.*
- *Design class diagrams.* 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*:** All diagrams must conform to the UML notational conventions presented in class. You can use Visio or the Lucid Charts (https://www.lucidchart.com) to create the diagram. If you use a tool different than these, you must make sure to output your diagrams to PDF.

- *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 development 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.

Assignment submission

Your development notebook must be in **MS Word format or Adobe PDF**. Any figures (such as the design class diagram) must be embedded directly in the development notebook document.

Place your development notebook, Chatbot_Vocabulary.xml, and Java source files into a single Zip file and submit to D2L by the submission time on the due date. **Do NOT include javadoc or class files in the Zip file; these will be generated by me.**