Project 2 - Chatbots

Due Date: 5:00 p.m., October 21
Any changes made to the assignment after posting it will be in red.

Description

One of the most famous programs in the history of artificial intelligence is Joseph Weizenbaum's program Eliza. Written in the early sixties it emulates, or parodies, a Rogerian psychoanalyst who tries to make the patient talk about his or her deepest feelings by turning everything said into a question. Eliza's fame is due to the remarkable observation that many people found her seem so friendly and understanding. In fact some said she was the first analyst they had met who really listened.

Over the years Eliza has had many successors and the phenomenon she represents is now known as a chatterbot. A possible definition is:

> *A chatterbot is a program that attempts to simulate typed conversation, with the aim of at least temporarily fooling a human into thinking they are talking to another person.*

Program Flow

You are going to build a program that carries on a 4 way conversation with a user. You will use:

- the observer pattern to set up a push notification relationship between the user and chatbots
- the template method when the chatbots enter the conversation
- the strategy pattern to decide how to respond.

You must use the ruby Observable module in your User class. There are 3 different chatterbots (response dictionaries are provided for each) that all follow these steps when entering a chat:

1. save the user as an instance variable
2. register as an observer of the user
3. read in and store their response dictionary
4. say a greeting

You must use the driver code provided below. The driver initializes the chatbots and calls each one's template method (enterChat). Each chatbot has a corresponding personality:

1. happy
2. sad
3. angry

Responses

The conversation will continue as follows:

5. The user will enter a chat string
6. The user will then notify all observers that a chat string is waiting
7. The chatbots will decide how to continue conversation based on the contents of the chat string
8. If the conversation continues, the bot will choose an appropriate response strategy
9. otherwise, end the conversation and say a unique goodbye.

The responses will be selected by using the following strategies:

- You should first attempt to match with a list of responses which are contained in the provided chatter dictionary files
    - The file format is as follows: partial match responses are followed by a colon and a list of legitimate responses separated by semicolons
    - Read the various chatter files in when you initialize your chatterbot, and store it in an internal data structure
        - *Do not change the structure of the chatter dictionary files*
    - When you attempt top match, you should select the closest (or longest) match
    - Once you have a match, you should select a possible response. You should select the least recently selected response out of the response options.
        - You may use some other algorithm of your own devising that relies on some other factor, but it shouldn't just be random.
- If there is no match, and if the user enters a question (anything followed by a question mark), simply respond with a deflection.
    - A deflection is a conversational technique where you always turn a question back around on the asker. For example, if the user asks,

"How's your mother?", the chatterbot could respond, "Actually, I was wondering about your mother."
        - This does not have to be too complex. For example, you could just have 2 or 3 deflection phrases that you cycle through.
    - If no partial match is found and the user did not ask a question use one of the 'change the subject' phrases under the 'change subject' label in the chatter file.
    - If the response drops down to 1 word, you end the conversation by saying goodbye in your chatbots own unique way and **deregister as an observer.**

For the response dictionaries, you must do the following:

- Alter each chatterbot dictionary textfile to give your bots a name (just replace the question mark).
- Add to each chatterbot dictionary textfile at least 3 additional partial match phrases with at least 3 additional responses to select from (9 total additional responses).

You can also alter existing responses to give your chatterbot a unique personality; however, **don't alter the match phrases.**

Additional credit will be offered for more complex/robust chatbots with clear personalities.

Design

In a Readme, you should provide a brief explanation regarding how you used the Template Method, Strategy, and Observer patterns.

Lastly, you may make any changes to the requirements you wish to make the chatterbot more sophisticated or interesting. Just provide an explanation in the Readme. The only rule is that the changes must not simplify the project.

Some design guidelines:

- Create a User class with an Observable module to represent the user's current state.
- Whenever the user enters a response, it should trigger an event that notifies that chatbots.

- Each class or module must go into a separate file. The main driver code should also be in a separate file.

NOTE: It is almost impossible to have the chatbot's reponses make sense all the time. You are not being evaluated on how good the conversation is or even if the response makes sense.

Resources

- main.rb
    - Use this file for your driver code. Make sure all your code works with an unaltered version of this driver code.
- chatter_angry.txt
- chatter_happy.txt
- chatter_sad.txt

Submission

- Required code organization:
    - Project2
        - Readme
            - Create a readme file containing an explanation of your project structure and any additional information you may need to add, and add it to your project folder
        - User.rb
            - contains your user class
                - This is your subject in your Observer pattern and should include the observable module
        - main.rb
            - Contains the provided main driver code
        - Chatbot.rb
            - Abstract Template Method Class
            - The chat class contains your template method along with necessary abstract methods to read in the dictionary file, select a response strategy, and update when notified by the Observable subject
        - DepressedChatbot.rb

            - Implements Chat and Observer Pattern
            - reads in sadchatter.txt
        - HappyChatbot.rb
            - Implements Chat and Observer Pattern
            - reads in happychatter.txt
        - AngryChatbot.rb
            - Implements Chat and Observer Pattern
            - reads in angrychatter.txt
        - Response.rb
            - This is just an interface
        - QuestionStrategy.rb
            - Implements Response
        - MatchStrategy.rb
            - Implements Response
        - ChangeSubjectStrategy.rb
            - Implements Response
        - EndConversationStrategy.rb (optional)
            - Implements Response
- Create a zip archive of your project 2 folder with the following command
    - zip -r project2 <<foldername>>
- This creates an archive of all file and folders in the current directory called project2.zip
- Upload the archive to Blackboard under Project 2.
- You may demo your project by downloading your archive from Blackboard. Extract your archive, then run your code, show your source, and answer any questions the TA may have.

Grading Guidelines

- **Project (55 Points Total):**
    - Patterns (25 points)
        - Correctly uses the template method to implement entering the conversation as described (5 points)
        - Correctly uses the strategy pattern for response strategies (10 points)
        - Correctly uses the observer pattern with a user class that triggers an event for the Chatterbots (10 points)
    - Design Requirements (25 points)
        - Uses the observable module from the Ruby library (5 points)
        - Each Chatbot has it’s own greeting and goodbye (2 points)

        - Added 3 additional pattern matches with 3 responses each (3 points)
        - Selects the closest or longest match when matching responses (3 points)
        - Has a logic for selecting a response once a match is found. It's not just random. (2 points)
        - All classes in a separate file (2 points)
        - Reads in and stores the chatter dictionary files on initialization of the chatbot (8 points)
    - Extra Credit
        - Extra Credit assigned for significantly increased complexity and functionality with an explanation provided in the readme (5 points max)
- **Submission (5 points):**
    - Follows requested project structure and submission format (2 points)
    - Follows formatting guidelines (3 points)