9 Assignment

Eliza

Our goal is to train our computer to be a mock psychiatrist, carrying on a conversation with a patient. The patient (the user) asks a series of questions. The computer-psychiatrist replies to each question as follows. If the question starts with one of the following (key)words: Why, Who, How, Where, When, and What, the computer selects one of the three (or more) possible answers appropriate for that question. If the first word is none of these words the computer replies 'I do not know' or something like that.

1. Start by designing the class *Reply* that holds a keyword for a question, and an *ArrayList* of answers to a the question that starts with this keyword.

2. Design the method *randomAnswer* for the class *Reply* that produces one of the possible answers each time it is invoked. Make sure it works fine even if you add new answers to your database later. Make at least three answers to each question.

3. Design the class *Eliza* that contains an *ArrayList* of *Reply*s.

4. In the class *Eliza* design the helper method *firstWord* that consumes a *String* and produces the first word in the *String*.

   The following code may help you. Look up the details of how this works in the documentation for Java libraries.

   ```
   System.out.println("Type in a question: ");
   s = input.nextLine();
   Scanner firstWord = new Scanner(s).useDelimiter("[^a-zA-Z]");
   System.out.println("The first word is: " + firstWord.next());
   ```

   Make sure your program works if the user uses all uppercase letters, all lower case leeter, mixes them up, etc. (Again, let the Java documentation help you find the solution.)

5. In the class *Eliza* design the method *answerQuestion* that consumes the question *String* and produces the (random) answer. If the first word of the question does not match any of the replies, produce an answer *Don't ask me that.* — or something similar. If no first word exists, i.e.,

the user either did not type any letters, or just hit the return, throw an *EndOfSessionException*.

Of course, you need to define the *EndOfSessionException* class.

6. In the *Interactions* class design the method that repeats asking questions and providing answers until it catches the *EndOfSessionException* — at which time it ends the game.

Selection Sort

In the Algorithms class design a static method *SelectionSort* that consumes an *ArrayList<T>* and an instance of a class that implements *Comparator<T>* and mutates the *ArrayList<T>* so that it is sorted in the order given by the *Comparator<T>*.

It is possible to combine all parts into one method, but you must use the following helper methods:

- *swap* that swaps in the given *ArrayList<T>* the elements at the two given locations.
- *findMinLoc* finds in the given *ArrayList<T>* the location of the minimum element among all elements at the location greater than or equal to the given location. Of course, it also consumes the *Comparator<T>*.
- *selectionSort* method that is described at the beginning.

Variants

You can choose to use any of the loops we have seen (including the *Traversal<T>*, and its implementation for *ArrayList<T>*. However, as the second part of the problem you must convert your solutions for *minLoc* and *selectionSort* to use either *while* loop without the *Traversal<T>* or *for* loop without the *Traversal<T>*.

If you already used one of these, convert the code to using the other loop. Rename your methods as *minLocV1* and *selectionSortV1*.