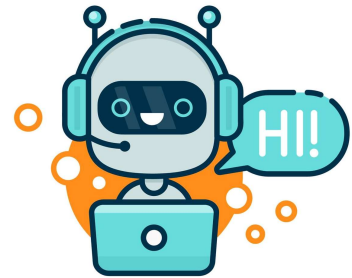


Virtual Agent Project

A virtual agent (often called a “chatbot”) is an application that uses rules (i.e., decision-making), string manipulation, and (increasingly) artificial intelligence, to interact with a human user. For example, a company might provide a “Virtual Help Desk Agent” on a web page to interact with a customer to provide support or direct them to other resources.



In this assignment, you will apply what we have learned so far in the course to create a chatbot. You choose the area of expertise for your chatbot. Some ideas:

- Entertainment: general small talk, joke telling, sports information
- Health/Wellness: advice on stress management, health diagnosis
- Troubleshooting: hardware troubleshooting, Java debugging

Assignment Requirements:

1. Modify the VirtualAgent class code as follows:

- a. On Coding Rooms, you will find an assignment entitled “**ASG #2: VirtualAgent**”. Carefully read the code that is provided. You will notice that the **protected** method **toTitleCase()** has a mostly empty method body. Add code to complete this method as explained in its Javadoc.
- b. Apart from the 1 change above, do **not** change **anything** else in the **VirtualAgent** class.

2. Create a new Java class for your chatbot:

- a. In Coding Rooms, create a file for a new class that will inherit from **VirtualAgent** and contain the Java code for your custom chatbot. Give your class an appropriate name.
- b. Include a **parameterized constructor** for your chatbot class. You do **not** need a no-argument constructor. Remember to use the “**this**” keyword to deal with instance variable vs parameter variable name ambiguity. Your constructor should also demonstrate the effective reuse of mutator methods.
- c. Include (at least) 2 **properly encapsulated** instance variables in your class. Include an accessor and mutator for each. Mutator methods should demonstrate effective parameter sanity checking.

- d. Implement the required **respondToUserQuery()** method. This is the “brain” of your chatbot. Your **main()** will pass String input from the keyboard into this method. This method should **not** print anything or read user input from the keyboard.

Your method code should analyse the user input String and **return** an appropriate response String to **main()**. Use the inherited **toLowerCase()** method and 4-5 different methods from the String class in your solution. For an extra challenge, try using the **matches()** method with regular expressions.

Some humble advice: start small with a few simple phrases and keep adding until you can carry on an interesting conversation with your chatbot.

- e. Include a **toString()** method that accesses the **toString()** method in the **VirtualAgent** class as part of the String it generates.
- f. You may include additional helper (i.e., **private**) methods in your chatbot class that you can call from **respondToUserQuery()** so it doesn't become too long (rule of thumb: if a method is longer than 1 printed page, it can probably be broken down into smaller, more focused methods).
- g. For all of the code you write, strive to minimize the scope of variables and methods when possible.

3. Complete the **main()** method in the **AgentTest** class:

- a. Your **main()** method (in the **AgentTest** class) should drive the main logic for your program. Before instantiating your chatbot, you might want to ask the user for some personal information (e.g., their name) so that you can instantiate your chatbot object with this information.

After instantiating your chatbot object, your code should execute a “chat loop” that will: (1) get input from the user at the keyboard, (2) call **respondToUserQuery()**, and (3) display the response String.

- b. When the user inputs “bye” the chatbot should terminate.
- c. Try to keep your **main()** function as minimal as possible. Most of your logic should be within your chatbot class.

4. Follow all 10 style rules from Unit 1, and all 8 style rules from Unit 2.

5. Include complete Javadoc comments for all of the code you write.

6. Reflect on your work. Complete the reflection questions and rubric below.

7. Demonstrate your work. Once you have met all of the requirements above, give Mr. Rao a demo of your program and discuss your self-assessment.

Reflecting on Your Assignment

- A. What aspects of the project were the most challenging for you, and why?

- B. If you did not meet all the requirements, which ones do you need to improve on?





- C. What part of your project are you most proud of?

Reflecting on Your Assignment

Name: _____

Topic: _____

Considering the requirements, how would you rate your project? (✓ Check one)

Criteria	 1 = Poor	 2 = Fair	 3 = Good	 4 = Great
Technical Requirements 1a, b, c 2a, b, c, d, e, f, g 3a, b	Improvement needed on 5 (or more) technical requirements.	Improvement is needed on 3 or 4 technical requirements.	All code requirements are met but need to improve on 1 or 2 of them.	All code requirements are met! Exemplary Java code.
Coding Style Rules 18 Style Rules + Javadocs	Poor style, many style rules ignored and/or missing Javadoc comments.	Significant room to improve on coding style and/or Javadocs.	Good style, but some room to improve on the coding style or JavaDocs.	Perfect style, Javadocs, and great attention to detail!
Application Complexity	Trivial chatbot or buggy. More effort is needed.	Pretty simple conversation. More effort was possible. 10+ responses	Decent effort, the project meets requirements but could have been more challenging. 20+ responses	"Intelligent" conversation, strong interest and effort demonstrated. Used some regular expressions. 30+ responses.

