AI AI Oh!

Due Tuesday, January 28 at 8 a.m.

CSE 1325 - Spring 2025 - Homework #2 - Revision 0

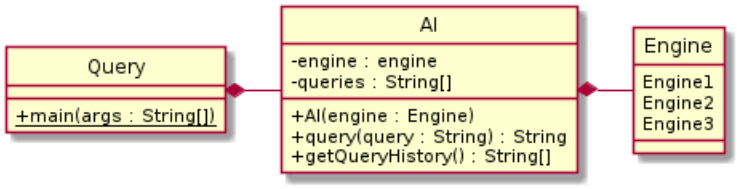
Assignment Overview

We've discussed the concept of *encapsulation* and how it is implemented using classes in Java with several examples. Now it's time for you to apply that knowledge by *encapsulating* information about your favorite AI engines. No, they may NOT write the code for you! :D

Important: You do NOT need to know anything AT ALL about AI to complete this assignment - I'll walk you through everything step by step in a lot of detail. If you work in the software industry, you'll likely write *many* programs in fields in which you are a novice - nothing artificial about it!

Full Credit

Consider this class diagram.



You may ONLY add the above public members to your classes. No additional public members are permitted. Don't duplicate any class responsibilities in your main method!

In your GitHub-managed working directory **cse1325/P02/full_credit**, write enum `Engine` that enumerates at least 3 **REAL** AI engines that you find via your favorite search engine. (Don't worry, you will NOT need to interact with these engines in any way until the Extreme Bonus.) **You may NOT use the ones shown in the example below.** Find your own!

Then write class `AI` with 2 *private* fields, 1 *public* constructor, and 2 *public* methods as shown in the UML class diagram.

- The constructor sets the `engine` to the corresponding parameter and allocates a 5-element array of `String` to `queries`.
- Method `query` "pushes" the `query` parameter onto the array as if it were a first-in first-out stack. So you'll need to copy the strings to the adjacent index first, and then add the `query` parameter at index 0. The string at index 4 will, of course, be lost. Then return a fixed response of your choosing (or be a little more creative if you like - just not an empty string!).
- Method `getQueryHistory` simply returns the private `queries` array.

Finally, write your main method in class `Query`.

- Obtain a `Scanner` so you can read queries from the user.
- If no arguments are provided, list the available AI engines (the enumerations in `Engine`) and exit.
- If one or more arguments are provided:
    - Convert the first argument from `String` to an `Engine` enumeration (enums have a `valueOf` method for this), then print out the enumeration (NOT the argument!).
    - Instance a new `AI` object using the `Engine` object in the previous bullet as the parameter.
    - Loop, accepting questions (a newline-terminated string) from the user and printing the result of calling your `AI` object's `query` method with each question. Exit the loop when Control+d (Linux and Mac) or Control+z (Windows) is pressed.
    - Finally, iterate over the AI's query history with a for-each loop, printing each question recorded as reported by `AI.getQueryHistory()`. Note that `AI.queries` is **private**. Use the method to obtain it, per encapsulation rules. You'll only get the most recent 5 questions.

You MUST provide a working build.xml file so that the `ant` tool works. A working build.xml is provided at `cse1325-prof/P01/full_credit/build.xml`, in any of the lecture code, or use the one provided at the end in the Hints section. Options R Us!

You MUST include at least 3 substantial commits of your evolving Full Credit work. A "one and done" commit, even with trivial additional commits just to get to 3, will incur *substantial* grade penalties.

Practice good version control!

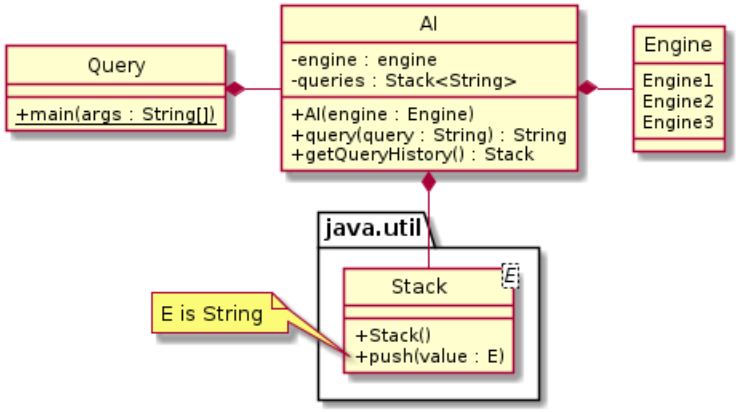
When compiled and run, your output should look something (but NOT exactly) like the screenshot on the next page. As long as you provide the required information, feel free to make the output look as nice and as "you" as you like. You do NOT need to provide screenshots, however, we'll run it ourselves.

```
ricegf@antares:~/dev/202501/P02/full_credit$ java Query
Available search engines are Komo Phind Brave
ricegf@antares:~/dev/202501/P02/full_credit$ java Query Phind
Using Engine Phind
Ask Phind anything:

Your most recent 5 queries were:
  null
  null
  null
  null
  null
ricegf@antares:~/dev/202501/P02/full_credit$ java Query Phind
Using Engine Phind
Ask Phind anything: What's the price of Kumquats in Kuwait?
That's a puzzler!

Ask Phind anything: Why did Alexander the Great cry?
That's a puzzler!

Ask Phind anything: How do we do structured programming in Java?
That's a puzzler!

Ask Phind anything: What good conspiracy theory should I start?
That's a puzzler!

Ask Phind anything: Should toilet paper hang over or under the roll?
That's a puzzler!

Ask Phind anything: What is my Wi-Fi password?
That's a puzzler!

Ask Phind anything: What is a male ladybug called?
That's a puzzler!

Ask Phind anything: Do you know any jokes?
That's a puzzler!

Ask Phind anything:

Your most recent 5 queries were:
  Do you know any jokes?
  What is a male ladybug called?
  What is my Wi-Fi password?
  Should toilet paper hang over or under the roll?
  What good conspiracy theory should I start?
ricegf@antares:~/dev/202501/P02/full_credit$
```

Bonus

Geesh, writing all that stack code is a real pain. Doesn't Java know how a stack works?

Yes, of course - in the `Stack` *class*!



Copy your `P02/full_credit` directory to `P02/bonus`, and in the latter directory modify AI.java to use a `Stack` object.

- You'll probably want to import the class `java.util.Stack`.
- Replace the `String[]` type with the `Stack<String>` type. The data type that will be stored in the stack is specified in angle brackets. Odd-looking, isn't it? Don't worry, you'll get used to it rather quickly!
- Replace `= new String[5]` with `= new Stack<>()` to construct your stack. Java already knows what type will be stored on the stack, because you told it when you declared the field.
- How do you push a value on the stack? Would you believe a `push(String)` method? Replace all the code for moving strings around in the array with a simple `queries.push(query)`. Easy peasy!
- Finally, instead of returning an array of strings, return the stack.

The other files won't change.

Wait - how is that possible? *We changed the return type from method getQueryHistory()!* See if you can figure out why the same code in main works for the array version in the Full Credit assignment *and* the `Stack` version in the Bonus assignment.

Include at least 3 substantial commits of your evolving Bonus work!

Compiling and running your program may now look something like the screenshot on the next page (but NOT exactly like - create DIFFERENT engines!). Notice that we're no longer limited to a history of 5 questions - ALL of our questions are stored in history. Credit the magic of classes!

```
ricegf@antares:~/dev/202501/P02/bonus$ java Query
Available search engines are Komo Phind Brave
ricegf@antares:~/dev/202501/P02/bonus$ java Query Brave
Using Engine Brave
Ask Brave anything:

Your most recent queries were:
ricegf@antares:~/dev/202501/P02/bonus$ java Query Brave
Using Engine Brave
Ask Brave anything: What's the price of Kumquats in Kuwait?
That's a puzzler!

Ask Brave anything: Why did Alexander the Great cry?
That's a puzzler!

Ask Brave anything: How do we do structured programming in Java?
That's a puzzler!

Ask Brave anything: What good conspiracy theory should I start?
That's a puzzler!

Ask Brave anything: Should toilet paper hang over or under the roll?
That's a puzzler!

Ask Brave anything: What is my Wi-Fi password?
That's a puzzler!

Ask Brave anything: What is a male ladybug called?
That's a puzzler!

Ask Brave anything: Do you know any jokes?
That's a puzzler!

Ask Brave anything:

Your most recent queries were:
  What's the price of Kumquats in Kuwait?
  Why did Alexander the Great cry?
  How do we do structured programming in Java?
  What good conspiracy theory should I start?
  Should toilet paper hang over or under the roll?
  What is my Wi-Fi password?
  What is a male ladybug called?
  Do you know any jokes?
ricegf@antares:~/dev/202501/P02/bonus$
```

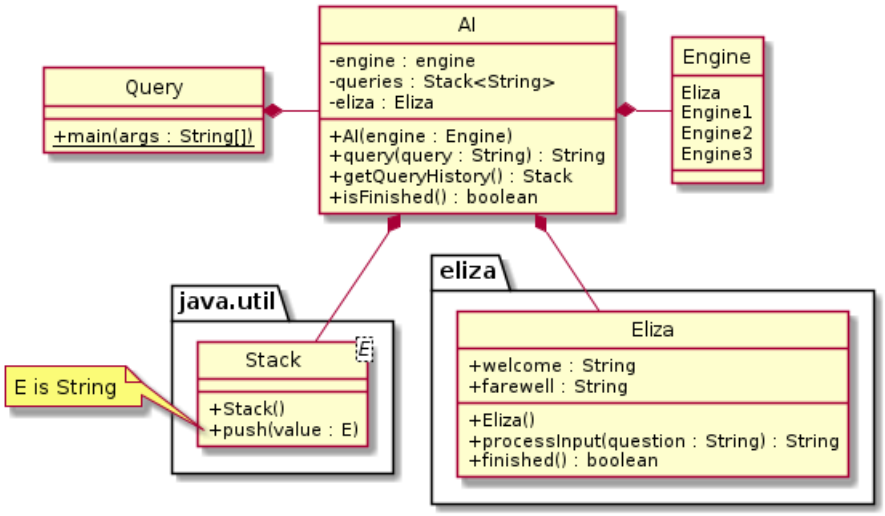
Extreme Bonus

You probably saw where this was going, We need to actually interact with a real state-of-the-art AI engine!

Well, it was state-of-the-art in 1966. You can't make this stuff up.

Feel free to use a modern engine if you like, but that turns out to be more complicated than I thought you'd want to try. So I dug up and polished up (a little) an ancient chatbot named Eliza, first written in the parentheses-happy Lisp language in 1966 and converted to Java (for a web applet!) in 2003. You'll find the slightly-polished code at `cse1325-prof/P02/baseline/eliza`, but do NOT try to learn Java from it!

All it needs is a good user interface. **Wait - you already wrote that!** Let's add Eliza to your list of AI engines - and this time, we can *talk* to her! (I obviously omitted a LOT of methods and classes in the `eliza` package in the UML class diagram below. But all you need to know about is the `Eliza` class as shown.)



Copy your `P02/bonus` directory to `P02/extreme_bonus`. Then copy the `cse1325-prof/P02/baseline/eliza` directory to `P02/extreme_bonus/eliza`. It is *critical* that Eliza's code be in the `eliza` subdirectory, and that it ALWAYS be built from the `extreme_bonus` directory. Those are Java's rules, and we don't argue with them. (I'll explain them later.)

OK, now add an `Eliza` value to your `Engine` enum per the class diagram. When you select the Eliza engine as an argument to your program, the magic will happen!

But first, update your `AI` class to *create* the magic.

- Import `eliza.Eliza`, the AI engine, into AI.java.
- Add an `eliza` field of type `Eliza` as shown in the class diagram.
- In the `AI` constructor, if and ONLY if the `engine` parameter is `Engine.Eliza`, instance a new `Eliza` object (no parameters are needed for the constructor) and assign it to the `eliza` field. If you like, you may also print its first welcoming response which will be returned using `eliza.processInput(eliza.welcome)`.
- In the `AI.query` method, if the `eliza` field is NOT null, return the result of passing the `query` parameter to `eliza.processInput(query)`. Otherwise, return the same fixed string as you did in the Bonus level for the other "engines".

That's it! You should now be able to carry on a conversation with the brilliant (well, in 1966 terms) Eliza AI!

You'll quickly notice that when you bid Eliza fairwell, she likewise bids you adeiu. However, your `Query.main` method just keeps on going. How can you close the program when the conversation ends?

Notice that class `Eliza` has a `finished()` method. This becomes true when Eliza has finished the conversation. You'll need to add an `isFinished()` method in class `AI` (as shown in the class diagram) to report it back to main. If `eliza` is null, always return false. But if `eliza` is NOT null, return `eliza.finished()` instead. Sounds like a job for a ternary, doesn't it?

Then, in `Query.main`, simply loop until `ai.isFinished()` is true or `scanner.hasNextLine()` is false!

Your output may look something like the next page. Or maybe a lot better! :)

Just before the class started, I also ran across another repository of Eliza in Java, also based on Charles Hayden's original translation from Lisp but possibly better than my quick-and-dirty polishing. I would have started here had I found it first. I did NOT look at or test it, though - the suggested solution will use the version from prof-rice.

https://github.com/codeanticode/eliza

Since we're not grading this one, feel free to ignore version control and lose all your code tomorrow.

It's painful, but nothing teaches good version control like **pain**! :D

```
ricegf@antares:~/dev/202501/P02/extreme_bonus$ java Query
Available search engines are Komo Phind Brave Eliza
ricegf@antares:~/dev/202501/P02/extreme_bonus$ java Query Eliza
Using Engine Eliza
How do you do.  Please state your problem.
>> I am a bit tired today.
Is it because you are a bit tired today  that you came to me ?

>> I did, in hopes you have a suggestion.
You say you did  ?

>>  I am here with a problem, hoping you have an answer.
How long have you been here with a problem  ?

>> I just wrote the program minutes ago!
Can you elaborate on that ?

>> I teach CSE1325, and this is an Extreme Bonus problem.
Do you say you teach cse1325  for some special reason ?

>> It's fun and I'm retired. :)
Do you believe it is normal to be retired  ?

>> It is now!
I'm not sure I understand you fully.

>> My wife says exactly the same thing!
In what way ?

>> Kindly, of course.
Please go on.

>> Goodbye
Goodbye.  It was nice talking to you.



Your most recent queries were:
  I am a bit tired today.
  I did, in hopes you have a suggestion.
   I am here with a problem, hoping you have an answer.
  I just wrote the program minutes ago!
  I teach CSE1325, and this is an Extreme Bonus problem.
  It's fun and I'm retired. :)
  It is now!
  My wife says exactly the same thing!
  Kindly, of course.
  Goodbye
ricegf@antares:~/dev/202501/P02/extreme_bonus$
```

Hints

Here's a working build.xml file. We'll keep using this one until Lecture 08.

You **must** include one in your repository for **each** assignment level - full_credit, bonus, and (please?) extreme_bonus!

Do NOT put it at P02, and (if you do the Extreme Bonus) do NOT put it in the eliza subdirectory. **Always build and run from the full_credit, bonus, or extreme_bonus subdirectory**.

```
<?xml version="1.0"?>
<project name="CSE1325" default="build">

 <target name="build" description="Compile source tree java files">
   <javac includeantruntime="false" debug="true" failonerror="true">
     <src path="."/>
   </javac>
 </target>

 <target name="clean" description="Clean output files">
   <delete>
     <fileset dir=".">
       <include name="**/*.class"/>
     </fileset>
   </delete>
 </target>
</project>
```