IJP Assignment 1: Chat bots

November 1, 2010

Introduction

This assignment is due at 16:00 on Friday 22nd October 2010.

The aim of this practical exercise is to write a simple "chat bot" in Java. During the course of the exercise you will:

- gain some programming experience by working on a more extended problem than you have tackled previously;
- practise integrating your code with core Java APIs and third-party libraries;
- work with a commonly used computational linguists' resource (WordNet);
- drive your friends mad with nonsensical automatically generated language.

Chat bots

A chat bot (or "chat 'bot", short for "chat robot") is a program which has been designed to converse with people, or indeed other chat bots. Since Weizenbaum's famous ELIZA, many, many chat bots have been created. Some were designed as competitors for the Loebner prize, others were created to demonstrate new language technologies, while yet others were created to exploit commercial interests in improving online sales. None of them are very convincing. However, creating a chat bot is good fun, and is an interesting learning exercise. In addition, the code you write in this practical exercise will be used in Assignment 2. You can choose to give your chat bot any personality and favourite conversational topics that you want.

Overview

By the end of this exercise you will have written a chat bot in Java. The chat bot has the following features:

- It can generate sentences to start conversations with another chat bot.
- It can process input from another chat bot and generate a response in the form of a sentence.

Marking Scheme

Marks will be awarded according to the following scheme:

Task 1: Generating sentences	50%
Task 2: Replying to input	50%

For each task, marks will be allocated for: **clear and concise documentation**; **programming style**; and **evidence of testing**. See later for details.

Preparation

You will find it helpful to refer to chapter five of "Objects First with Java: A Practical Introduction using BlueJ". You will also need to download the code on which you will base your work from the IJP course web page. The description of WordNet and the online glossary at `http://wordnet.princeton.edu/man/wngloss.7WN.html` will also be useful.

Sanity Warning: *Don't lose sleep over this exercise! If you are having difficulty, ask your lab demonstrator or lecturer for help.*

Setting up BlueJ

If you haven't already done so, make a directory called `IJPAssignments` under your home directory. Download the BlueJ skeleton project archive for this assignment from the IJP course web page into this newly created `IJPAssignments` directory. Extract the archive. Open the IJPAssignment1 project in BlueJ. You must tell BlueJ to use an archived Java package (`jwnl.jar`) for this project. To do this, go to the **Tools** menu, choose **Preferences** and then go to the **Libraries** tab. Click on **Add**, and browse to `IJPAssignment1/lib/jwnl.jar`. Restart BlueJ.

Extra step if you are not working on a DICE machine (e.g. at home): if you want to work on a non-DICE machine, you will need to change the configuration of the WordNet package. First, you need to get a copy of WordNet 1.7.1 (and it MUST be this version) from `http://wordnet.princeton.edu/oldversions`. Install it on your PC. The file `IPJAssignment1/jwnl/file_properties.xml` specifies where BlueJ can find the data files for WordNet. You should change the line:

```
<param name="dictionary_path"
    value="/group/teaching/ijp/WordNet-1.7.1/dict"/>
```

to refer to the place where you installed WordNet on your computer. For example:

```
<param name="dictionary_path" value="D:/WordNet1.7.1/dict"/>
```

Getting Started

Open the IJPAssignment1 project in BlueJ. You will find that there is a package structure to help you. Packages are collections of classes with related functionality. For this assignment, all the code is contained within a package called `ijp.assignment1`. Expand this package in BlueJ. You will see that it contains three sub-packages: `langen`; `chat` and `utils`. The `langen` package contains classes which handle language generation tasks. The `chat` package contains classes associated with the chat bots. The `utils` package contains helper code which will be useful to you when you modify the `langen` classes. You will not need to modify the `utils` package, although you may find it useful to use BlueJ to try out some of the methods in `WordNetWrapper`.

Open the `chat` package and look at the interface for the `ChatBot` class. You will implement the methods for starting conversations, and for replying to a remark made by another chat bot. These methods will be used by the `Discussion` class to enable a "dialogue" between two `ChatBot` objects.

The implementation of the `ChatBot` methods rely heavily on code you will write in the `langen` package. To reduce the complexity of this exercise, you will use a template approach to language generation. Open the `langen` package in BlueJ. There is an interface called `LanguageGenerator` which specifies methods for generating sentences and replies, and an implementation of that interface called `TemplateLanguageGenerator`. This design makes it easy to swap in different sorts of language technology by implementing the methods in `LanguageGenerator`. For example, if we wanted to use an ATN (Augmented Transition Network — you don't need to know about these) approach instead of a template approach to generation, we could replace `TemplateLanguageGenerator` with another implementation of `LanguageGenerator`, which would provide the same functionality but using augmented transition networks instead of templates. The code in the `chat` package would not be affected.

Task 1: Sentence generation

The first task is to implement a sentence generator by completing the `TemplateLanguageGenerator` method `generateSentence()`. The `SentenceTemplate` class will store the data for your generation templates. `SentenceTemplate`s have a field indicating what type of utterance the template generates, e.g. question or reply. You may add to the types if you wish. `SentenceTemplate`s also have a `String` field which stores the template as

a string in a special format. These templates contain either canned text or instructions to generate text to fill gaps in the template. The canned text and the instructions are separated by "%" strings. A simple example String representing a template is shown below:

```
"Did you know that %NOUN% is %ADJECTIVE%?"
```

The canned text "Did you know that" is followed by an instruction to generate a noun, followed by the canned text "is" followed by an instruction to generate an adjective. An example sentence which could be generated by this template is:

```
"Did you know that beauty is transitory?"
```

Activities

Remember the marking scheme: throughout these activities, as well as basic correctness, we will be looking for **clear and concise documentation**, good **programming style** and **evidence of testing**. See later for details.

1. Write **five** example templates of your own. Write a method which populates the `TemplateLanguageGenerator.sentenceList` data structure with these templates. [*10 marks*]

   Hint: *Remember, the more ambitious you make your templates, the harder it will be to write code which expands the templates in the next step!*

2. The templates you have written are a shorthand way of specifying how to generate sentences. You also need to write some code which expands these template strings into strings representing the full sentence. To do this, implement the `expandTemplate` method to give it the functionality described its Javadoc comment. You will find the methods in `utils.WordNetWrapper` useful if you want to generate nouns, verbs or adjectives at random. A `WordNetWrapper` object is large — consider how many such objects your program is using and if it needs that many. [*25 marks*]

3. Now you are ready to implement the method `generateSentence`. Make it pick a sentence from your `sentenceList` at random, and use your `expandTemplate` method to generate a sentence from that template. Test the sentence generator in BlueJ. [*7 marks*]

4. Use your sentence generation code to implement the `startConversation` method in `ChatBot`. Implement the other methods and constructor in `ChatBot`, apart from `reply`, which you will tackle in the next task. [*8 marks*]

Task 2: Responding to input

The second task is to make your chat bot (slightly) more responsive to input. This input could come from another chat bot or a human user. This will require implementation of the `generateReply` method in `TemplateLanguageGenerator`.

Activities

Remember the marking scheme: throughout these activities, as well as basic correctness, we will be looking for **clear and concise documentation**, good **programming style** and **evidence of testing**. See later for details.

5. Implement simple keyword matching so that the chat bot responds to certain words within remarks with a segment of canned text. This should be implemented in the method `keyWordMatch`. [*10 marks*]

Another approach to responding to input is generate replies using templates, rather than relying completely on canned text. To keep it simple, the chat bot will generate a reply which is (vaguely) related to the input through reference to related concepts.

6. Write a method called `findKeyWord` which picks a word from the input string which will be used to generate the reply. Write a method called `expandTemplateTopic` which expands the templates you specified in Task 1 using words related to the keyword from the input selected by `findKeyWord`. You can use the methods in `WordNetWrapper` to help you find words which are related to each other. For example, in response to the example template above, you could use to respond with a sentence which contains a synonym of the first noun in the input. [*25 marks*]

7. Now implement `generateReply` using the methods you implemented in this task. It should pick randomly whether to use canned text or a generated reply. Test this method using BlueJ. [*5 marks*]

   Hint: *You will find that this approach to language understanding is somewhat limited! Don't spend too long trying to make your chat bot talk sense!*

8. Finish implementing the methods in `ChatBot`. Test your chat bot using the `Discussion` class in the `chat` package. Make it converse for **ten** conversational turns (five from you, and five from the chat bot). Devise a set of tests to discover the capabilities (and/or limitations) of your chat bot. Save each chat output to a text file called `TestN.txt`, where

`N` is the number of the test. Insert some text at the top of each test file explaining why you chose to do that test, and any improvements you would like to make to your code based on the results of your test. Create a folder called TestResults in the directory `IJPAssignments/IJPAssignment1/`. Copy your test files into the `TestResults/` folder. [*10 marks*]

Testing and Documentation

Clear and concise documentation means proper Javadoc-style commenting of classes and methods (for users of the class), and adequate inline commenting of code (purpose of each attribute, private method, branch points, etc).

"Evidence of testing" means unit tests (Section 6.3 and 6.4 of the book). You are not expected to write full unit tests for all the classes and methods — that would be a big job. But you should write at least one test for each class, and test one class quite thoroughly; you should make it clear that you understand the concept of unit testing and are able to apply it.

JUnit supports testing of public methods. However `expandTemplate`, `keyWordMatch`, and `findKeyWord` are private — the public method which uses them is `generateReply`, which uses one of them at random. So you cannot effectively test these three by simply writing a JUnit test for `generateReply`.

One way around this is to write additional public methods (in the same class) which are wrappers for these three. For example:

```
public String keyWordMatchWrapper(String remark) {
    return keyWordMatch(remark);
}
```

Then write a unit test for the public method `keyWordMatchWrapper`, which will effectively be a test for private method `keyWordMatch`. There are better ways, but they involve advanced techniques not covered until later in this course.

Also, there is no need to write tests for the classes in `Utils` (`WordNetWrapper`, etc.): you are using these classes, not authoring them.

In Activity 8, the tests here are not unit tests. In these tests you start the conversation off with some question or statement, and see where the `Discussion` class takes the conversation over the course of ten turns. You want to start with a variety of different types of questions or statements to show for what sort of starting points your chat bot is effective and ineffective (for example, if you start off with "Why?", what will your chat bot do?).

Plagiarism

Informatics takes plagiarism seriously, and has automatic plagiarism detection systems for code submissions. Penalties apply, and Head of School and Head of College will be notified if plagiarism is detected.

A useful method of learning is discussing and sharing ideas with other students. However, you **MUST NOT** present other students' work as your own:

- Don't share code with other students.
- Acknowledge ideas and help you've had from other people.

You must take steps to ensure that your work is not copied by other students:

- Don't give people your code.
- Make sure your directories are read protected.
- Don't leave print outs of your work lying around.

If you are not sure exactly what is meant by plagiarism, see: `http://www.inf.ed.ac.uk/teaching/years/msc/courseguide10.html#plag`

Electronic Submission

If you followed the directions above, your work (code and test results) for this assignment should be in the folder `IJPAssignment1/` contained in the directory `/home/sXXXXXXX/IJPAssignments/` (where `XXXXXXX` is your matriculation number). You will submit the folder `IJPAssignment1/`.

Go to the folder `/home/sXXXXXXX/IJPAssignments/` by issuing the following command at a UNIX shell prompt:

```
cd /home/sXXXXXXX/IJPAssignments/
```

Now, create an archive of the `IJPAssignment1` folder (and all of its subfolders) by typing

```
tar cjvf IJPAssignment1-complete.tar.bz2 IJPAssignment1/
```

You will use the `submit` command to send us this archive. At the UNIX shell prompt, type

```
submit msc ijp 1 IJPAssignment1-complete.tar.bz2
```

You will be asked to confirm that you wish to submit your work. Once you say yes, you should see a message that your submission was successful. Ask a lab demonstrator or lecturer for help if you foresee having trouble with the submission process.

You can submit your work more than once. Each time you do so, *the previous version will be overwritten.* **Only the final submission will be marked.**