Look Who's Talking *(Etude 5)*

Benchmarking report

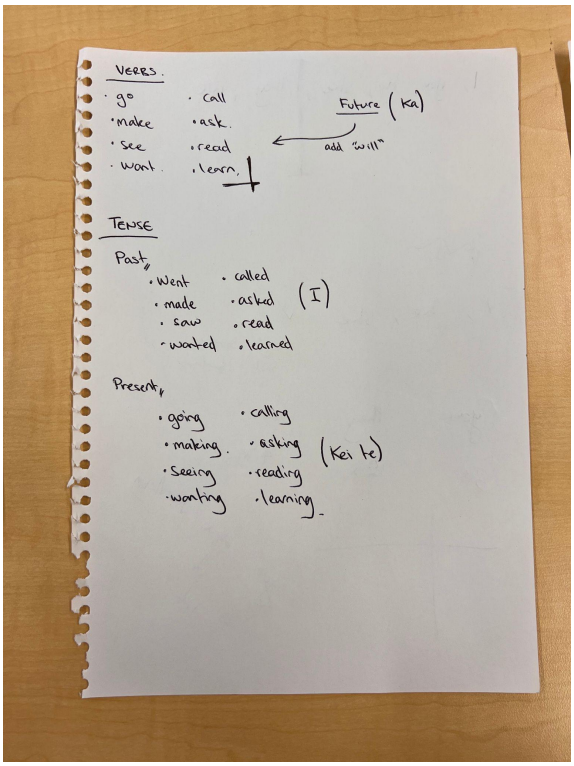
Jay Lee - 3976743
Hongyu Huang - 3567710

Purpose

The purpose of this etude was to create a program that takes an English sentence such as *"We (3 excl) are going"* and translates it into Māori (*"Kei te haere mātou"*). Typically in English sentences, pronouns can be very ambiguous whereas in Te Reo Māori, the number of people involved and the inclusivity of the pronoun is specified. Furthermore, verbs in English differ depending on what tense you are speaking in. But in Te Reo Māori, a verb will not change; rather, the 'sentence starter' differs depending on the tense. An example of this is how "*haere*" is the verb for "*to go*" and "*kei te*" is the sentence starter for the present tense. Therefore, "going" in the present tense will translate to "*kei te haere*".

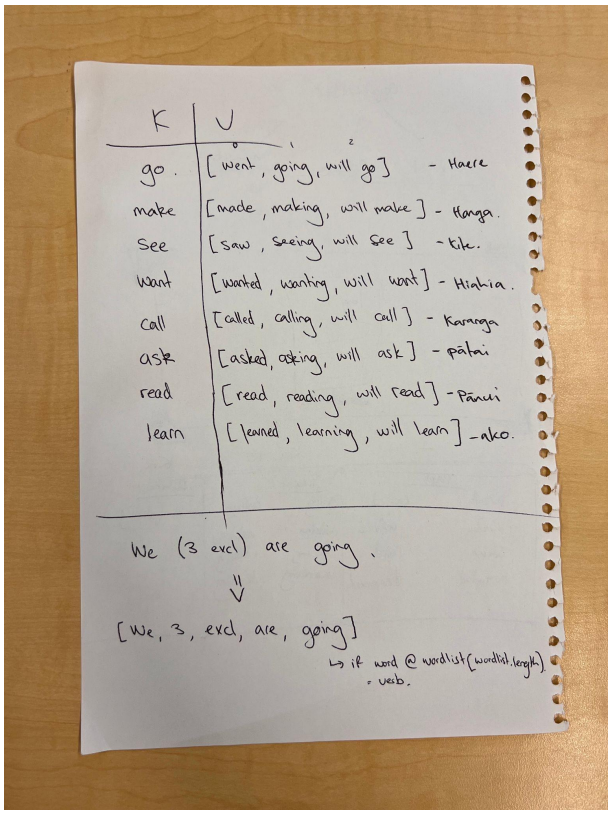
Methodology

When it came to developing this program, we made sure to really understand the problem at hand and make a well-thought-out plan to help in our approach. This involved a lot of pen and paper scribbling.



Pen and Paper Planning (PPP)

The first hiccup in our plan was how to handle the wide range of "use cases" in which there are 3 tenses for 8 different verbs we are trying to translate. Handling each of these individually would lead to inflated if, else statements which would be hard to follow. This problem acknowledgement led to the idea of a hashmap being used for storing the English verb as the key and having the three tense variations of the verb being stored in a String array which would be the value pairing to the key (see attached image).



However, we soon realised this would not be efficient as we had no way of including the Te Reo Māori translation of the verb. This lead us to change our idea so we'd have the Te Reo Māori translation of the verb as the key in the hashmap while still keeping the value pairing (String array) the same.

Development

User Input

After extensive discussion and planning, we began developing our program. For receiving user input, we used the Scanner class and passed the system input into a String variable. From here, we needed to split the input into individual words and insert them into an array. However, in order to do this effectively, we had to find a way of handling user input that has parenthesis (for when user input specified the number of participants and the inclusivity). In order to deal with this, we used regexes (regular expressions) to remove the parenthesis from the input and replace them with spaces which then allowed us to split the input into these spaces and insert each word into a String array.

Verbs and Tense

With the user input now split and inserted into an array, we began our implementation of the first translations (the verb and tense). Since the HashMap we initialised earlier contained all the information we required (the Te Reo Māori translation of the verb and the associated English translations in the different tenses), we decided to loop through the HashMap entries. For each HashMap entry, we compared the strings in the value String arrays to the last string in the user input array (the English verb) and depending on which string matched, we were able to determine the tense (since index 0 of the String array in the Hashmap held the past tense English translation, index 1 held the present and index 2 held the future). From this, we appended the correct Te Reo Māori sentence starter to the output we would be printing. From here, we further looped through each string array in the hashmap and if we found a match between the string and the user input, we appended the key value from the hashmap to the output (which was the Te Reo Māori translation of the verb).

Pronouns

Since we know that in the English sentence structure, the pronoun is always first, we did comparisons with the word in the first index of the user input array and depending on the match (I, he, she, you, they, we) We branched into different if/else statements to handle the different numbers and inclusiveness of the pronouns. Based on the table provided, we could see that “I” has one Te Reo Māori translation, “you” has three, “he”/”she” has one, “we” has four and ”they” has two. Since “I” and “he”/”she” only had one, they were the easiest to implement.

	Includes both speaker and listener	Excludes the listener(s)	Excludes the speaker	Neither the speaker nor listener(s)
One person		**au/ahau** I,me	**koe** you	**ia** he, she, him, her
Two people	**tāua** we, us (you and I)	**māua** we, us (but not you)	**kōrua** you two	**rāua** they, them
Three or more people	**tātou** we, us (including you)	**mātou** we, us, (but not you)	**koutou** you	**rātou** they, them

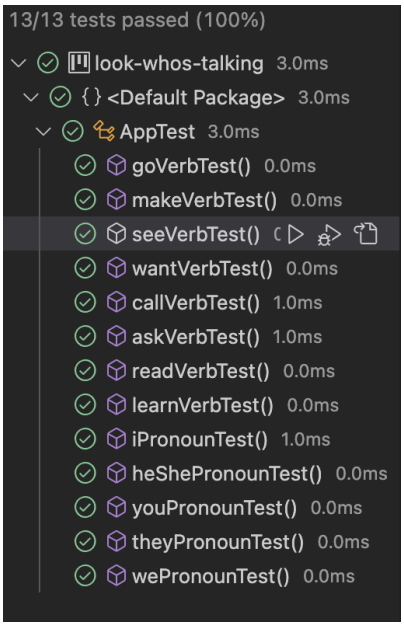
However, “we”, “you” and “they” required further if/else conditions to account for the number of people and the inclusiveness. Depending on which condition was satisfied, the appropriate Te Reo Māori translation was appended to the output.

The Output

Throughout the whole translation process, we kept track of some class boolean variables. In each function (verbAndTenseCheck and pronounCheck), whenever anything was appended onto the StringBuilder variable “Output”, the appropriate boolean variable was checked to true (in the verbAndTenseCheck function, the boolean variable “verb” was changed and “pronoun” for the other function). At the end of the program, we implemented a printOutput function which, depending on the boolean variables, would either replace the whole StringBuilder variable with “INVALID” or just print the output variable as is. If either the verb or pronoun boolean variable was set to false (meaning the user input was invalid, incorrect or could not be translated), the program would print “INVALID”.

Testing

Initially in development, we used .txt files that we ran alongside our program to test that the output was translated as expected. In these .txt files, we used different variations of pronouns and different tenses. However, further along the development process when the program was coming together; we began using JUnit for testing purposes.

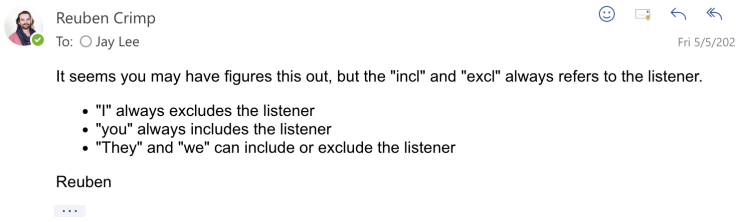


Since our program was split into two major functions (verbsAndTense and Pronouns) and didn't return boolean variables but rather appended to the output StringBuilder variable, we made custom functions in our AppTest.java class which would take the expected results and the emulated user input as variables and then call functions with the emulated user input. From here, it would then do an assertEquals onto the expected result with the outcome from the function call to pass the test.

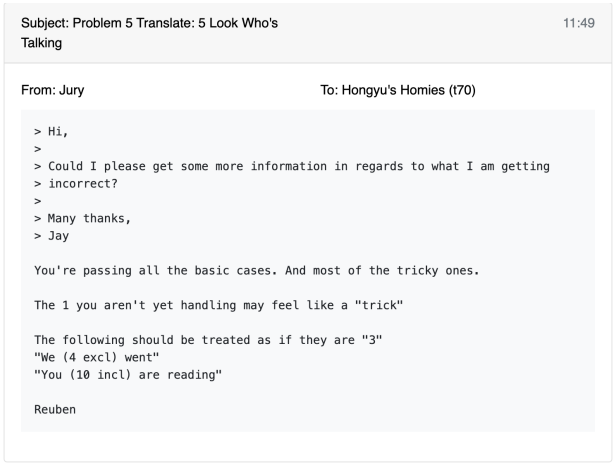
With these custom functions, we were then able to implement JUnit tests for every single possible combination of verbs, tenses and pronouns with the different user inputs.

Reflection

During the development process, there were some issues we came across that definitely could've been prevented with better planning and discussion.



The first problem was that we had trouble understanding the inclusivity of the pronouns and the understanding that we ultimately settled on, was actually the wrong one. After getting further clarification from the teaching staff, we came to the understanding that the "excl" and "incl" in the user input would always be referring to the listener. This issue could have been resolved far sooner if we had taken longer to discuss and plan the initial spec of the project and maybe included some teaching staff in our discussion.



The second problem we encountered was misreading the number of people involved in the pronouns. Our initial implementation only accounted for the user specifying 2 or 3 participants in the conversation. However, some clarification from AutoJudge stated we needed to account for more than 3 participants which was actually shown in the spec. Due to this, we

had to change our pronounCheck function slightly to parse the String (indicating the number of participants) into an integer so we could do a comparison and change the translation if the number of participants was greater than or equal to 3.