# KIT101 Programming Fundamentals

# PP Task 5.2 Collection of Strings

## Overview

**Purpose:** Gain practical experience writing or adapting code to manage data in arrays.

**Task:** Implement a menu-driven application to manage a collection of words, combining existing code (from the notes) with code you write yourself.

**Learning Outcomes:** 1 👁 2 🔀 3 ✏

**Time:** Aim to complete this task before the start of Week 6.

**Resources:**

- Introductory Programming Notes:
  - 08 Making Decisions
  - 09 Repeating Actions with Loops
  - 10 Managing Collections with Arrays
- Online Mini-lectures:
  - Arrays:
    - Introduction and key syntax
    - Tracing array code
    - Writing methods to work with arrays

## Submission Details

Upload the following to the MyLO submission folder for this task:

- Your **WordManager.java** source file
- A **screenshot** showing that your WordManager program runs

## Assessment Criteria

A ▩ Completed submission will:

- Include completed implementations of `add`, `printList` and `averageLength`

- Show that you are working with multiple String values in an array
- Follow the unit's coding style for layout, variable names (use of case), and commenting (including your name at the top)
- Compile and run, with the screenshot showing that it works

## Instructions

In this task you will write a small program to maintain a collection of words. The program will allow the user to add words, display the list of words, and calculate and display the average length of the words in the collection. (Several common operations, like deletion, are not supported.)

You will be able to make use of samples provided in the notes and accompanying the lectures to solve parts of the problem, but you will need to modify some of these too. If you make use of code from the notes (which is likely) then include a short acknowledgement in the program's header comment, such as "The methods *methodName* and *methodName2* include source code provided in the notes on MyLO" (the wording is up to you).

1. Download the **5.2PP Collection of Strings Starter Code**. This has some of the structure and necessary imports already included. Note that it has two methods already defined, in addition to main(), but that they are empty or almost empty.

2. The structure of the main program should be the following:

```
Program: WordManager

Constants:
int CAPACITY, maximum number of words, initialised to 20

Variables:
Scanner sc, to read user input, initialised when declared
String[] words, collection of words, initialised to hold CAPACITY elements
int count, number of elements in words, initialised to 0
int choice, user's menu selection
String aWord, new word given by user

Steps:
1    Display "Word Manager"
2    Do
3    |   Display a menu "1. Add a word
     |                   2. Display words
     |                   3. Display average word length
     |                   4. Quit"
4    |   Prompt user to "Enter option: "
```

```
5     |   Assign choice value of next int from sc (the Scanner)
6     |   Switch based on choice
6-1   |   |   In the case that choice is 1:
      |   |       Prompt for and read in next word from user, storing in aWord
      |   |       Assign count result of calling add with words, count, aWord
6-2   |   |   In the case that choice is 2:
      |   |       Call printList with words, count
6-3   |   |   In the case that choice is 3:
      |   |       Display "Average word length: " + result of calling averageLength with words, count
7     While choice is not 4
```

3. Begin by implementing main():
   - Declare the variables and constant. Make sure that the array and count are initialised.
   - Create the do-while loop, then the code to display the menu and get the user's choice. **Make sure you can run the program and end it by choosing Quit.**
   - Create the `switch` and for each of the three cases (1–3) add a statement to display which option was selected (you will replace this later), followed by `break`;

4. Test that the menu works: each option should indicate that it was selected, and 4 should quit.

## Implementing the add word functionality

3. You can see that the implementation of `add()` is currently empty. Implement the body of the method, referring to the notes as needed. It should behave as follows:
   - if `count` is less than the length of the array then it adds the given word to position `count` in the array (this will be the next empty place) and returns a value of `count` + 1;
   - otherwise it displays a message saying the collection is full, and returns the value of `count` unchanged. (This is slightly different to the implementation provided in the notes.)

4. Add code to read a word from the user, store it in `aWord`, and then call `add()` with the `words` array, value of `count` and the word to add (`aWord`). (In a larger program this would be done in a separate method, but to keep this program a little simpler you will write some of the necessary code inside the relevant `case` of your `switch` statement.) Make each line of code line up with the first one, as in

```
case some-value: statement1;
                 statement2;
                 break;
```

and remember to end with **`break`**.

> **Note:** Make sure you update `count` with the result of calling `add()` so that, if the user enters too many words, you are not unnecessasrily incrementing `count`. This is a common pattern in languages like Java, where a method that modifies a collection also returns a value to indicate if or how it changed that collection.

5. Compile the program and fix any errors. The next stage will allow you to more easily test that add is working.

## Implementing list printing functionality

6. You can see that the implementation of `printList()` is currently empty. Using the notes and sample code on MyLO as a reference, complete its implementation so that it prints the first `count` elements of the array as a comma-separated list of values. So, if `words` currently contained 4 values and those values were `"it's"`, `"just"`, `"an"` and `"example"`, then `printList` would display:

   ```
   it's, just, an, example
   ```

   and move to the next line.

   > **Tip:** You'll need some additional code to handle the ", " separator to make sure you don't add it after the final element.

7. Revise the relevant **`case`** in main() so that it calls printList with the array of `words` and `count`.

8. Compile and run the program. Enter a few words then display the list. Enter some more words, up to and beyond the array's capacity, and then display it again. Discuss any runtime errors with your tutor.

## Implementing average word length functionality

9. Add a new method called **averageLength** according to the following, high-level specification:

```
Method: double averageLength(String[] words, int count)

Returns:
double, the average length of the first count words in the array words

Parameters:
String[] word, the list of words
```

```
int count, the number of filled positions in the array

Variables: For you to decide

Steps:
1. Calculate the sum of each word's length
2. Calculate and return the average word length (that sum divided by count)
```

**Tip:** This method is *very* similar to one in the code accompanying the notes and online lectures on arrays. Ask your tutor for a suggestion if you are really stuck.

10. Revise the relevant case in main() so that it calls averageLength with the array of `words` and `count` and displays the result as part of a formatted message.

11. Compile and test the program with a variety of words in the array (work out for yourself what the average length should be to confirm your program is running corretly.

12. **Take a screenshot** of a sample run of your program (try to fit in the entire run and to use every menu option), then submit both the screenshot and source code to the task's assignment submission folder on MyLO.