CT2109 - Object-Oriented Programming: Data Structures & Algorithms

Assignment 01

Andrew Hayes

Student ID: 21321503

2BCT

University of Galway

January 20, 2023

1 Problem Statement with Analysis & Design Notes

At its most basic, this assignment will require an array of all the letters of the alphabet, and two methods: one to loop through the array in alphabetical order, and the other to loop it in counter-alphabetical order.

There will be a choice presented to the user whether to loop through the alphabet *forwards* (alphabetical order) or *backwards* (counter-alphabetical) order. To choose between these two options there will be a case statement. A `char` will be scanned in from the user. If the `char` is equal to `'f'`, then the `forwards()` method will be executed, and the time taken will be printed out. If the `char` is equal to `'b'`, then the `backwards()` method will be executed, and the time taken will be printed out. The `default` of the case statement will be to print an error message and exit with code `1`. While the use of two separate methods for the alphabetical & counter-alphabetical versions is not the absolute most efficient option in terms of lines of code, the use of separate functions enhances the readability of the code and allows for a cleaner implementation, at the cost of repeating only two or three lines of code. It also reduces the need for nested if statements, which are both inefficient and often difficult to read.

The `forwards()` & `backwards()` methods will be essentially the same. They will both return the `long` data type, which will be the time taken in seconds. To calculate the time taken in seconds, before the "game" begins, the start time will be recorded with `System.currentTimeMillis()`. The end time will be recorded in a similar manner at the end, and the total will be calculated from subtracting one from the other. This total will then be divided by 1000 (to convert from milliseconds to seconds) and then returned, where it will then be printed out.

The game itself will work in a similar manner in both methods: the array of the letters of the alphabet will be looped through using a `while` loop A character will be scanned in from the user. If the character is correct, a success message will be printed and the index variable incremented or decremented as appropriate for the method. Otherwise, the loop will be executed again on the same character, effectively ignoring the input, until the appropriate character is entered. If the index is at the last character of either sequence (alphabetical or counter-alphabetical), a special message will be printed out if the answer is correct.

The only real difference between the two methods will be the way in which they loop through the alphabet array. `forwards()` will loop through the array in alphabetical order, starting at index `0` and incrementing the index for each correct answer, with the loop condition `while (i < 26)`. `backwards()` will loop the array in counter-alphabetical order, starting at index `25` and decrementing the index for each correct answer, with the loop condition `while (i >= 0)`.

2 Code

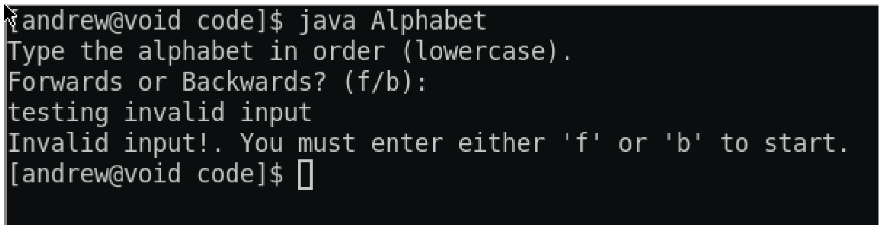
```java
import java.util.*;
import java.io.*;

public class Alphabet {
    // initialising a String containing all the letters of the alphabet and
    turning it into a character array (to save typing)
    public static String alphabetString = "abcdefghijklmnopqrstuvwxyz";
    public static char alphabet[] = alphabetString.toCharArray();

    // declaring a Scanner object to use to scan in the user input
    public static Scanner scanner = new Scanner(System.in);

    public static void main(String args[]) {

        System.out.println("Type the alphabet in order (lowercase).");

        // prompting the user to enter 'f' or 'b' and scanning in the input (only
     scanning the 0th character of input)
        System.out.println("Forwards or Backwards? (f/b): ");
        char fb = scanner.next().charAt(0);

        // switch statement to determine forward or backwards, or exit
        switch(fb) {
                case 'f':
                    System.out.println("Time taken: " + forwards() + "s");
                    break;

                case 'b':
                    System.out.println("Time taken: " + backwards() + "s");
                    break;

                default:
                    System.out.println("Invalid input!. You must enter either 'f' or 'b' to start.");
                    System.exit(1);

        }
    }

    // method for going through the alphabet forwards
    private static long forwards() {
        // getting the time when the user started
        long start = System.currentTimeMillis();

        // looping for each of the 26 letters
        int i = 0;
        while (i < 26) {
            // checking if the scanned in character was the correct one
            // if the it was not, then i will not be incremented, and it will loop again on the same letter.
            if (scanner.next().charAt(0) == alphabet[i]) {
                if (i == 25) {  // checking if it's the last letter, and if so, printing a special message
                    System.out.println("Correct! Well done!");
                    i++;
                }
                else {          // otherwise, prompting the user to enter the next letter
                    System.out.println("[" + alphabet[i] + ": Correct! Now type '" + alphabet[++i] + "']"); // i gets incremented
                }
            }

        }

        // calculating the total time taken
        long total = System.currentTimeMillis() - start;

        // returning the total time taken in seconds (by dividing by 1000)
        return total / 1000;
    }

    // method for going through the alphabet backwards
    private static long backwards() {
        // getting the time when the user started
        long start = System.currentTimeMillis();

        // looping for each of the 26 letters
        int i = 25;
        while (i >= 0) {
            // checking if the scanned in character was the correct one
            // if the it was not, then i will not be decremented, and it will loop again on the same letter.
            if (scanner.next().charAt(0) == alphabet[i]) {
                if (i == 0) {  // checking if it's the last letter, and if so, printing a special message
                    System.out.println("Correct! Well done!");
                    i--;
                }
                else {           // otherwise, prompting the user to enter the next letter
                    System.out.println("[" + alphabet[i] + ": Correct! Now type '" + alphabet[--i] + "']"); // i gets decremented
                }
            }

        }

        // calculating the total time taken
        long total = System.currentTimeMillis() - start;

        // returning the total time taken in seconds (by dividing by 1000)
        return total / 1000;
    }

}
```

Listing 1: Alphabet.java

3 Testing

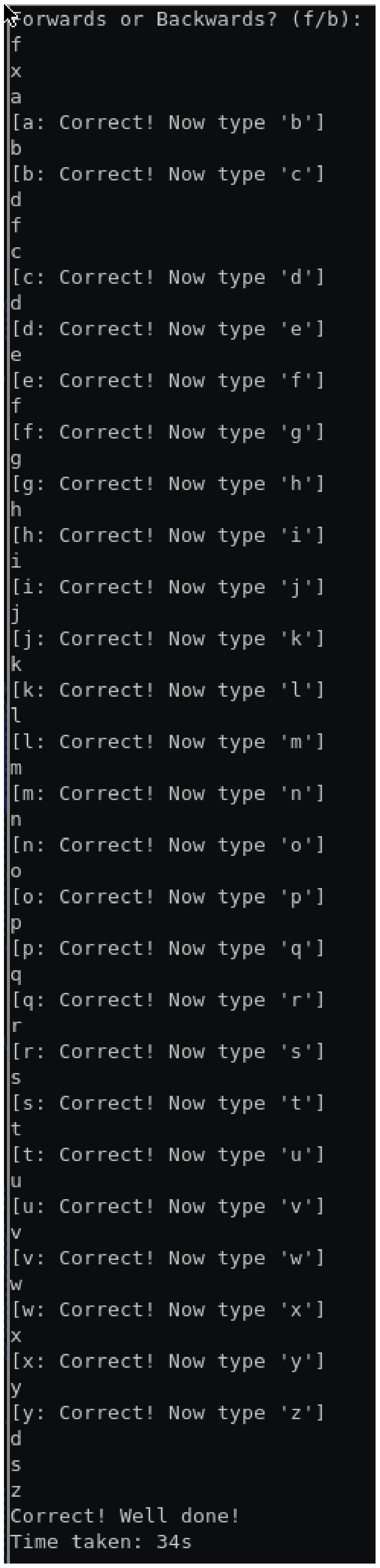
There are two main scenarios that must be tested for both the `forwards()` & `backwards()` methods: correct input & incorrect input. Of these two main scenarios, we should test them at the first character of the sequence, the last character of the sequence, and some of the middle characters. The main reason for this is that the first & last characters are in a sense special, particularly the last one, so what may work for the "middle" characters may not work for the others.

Furthermore, the case statement to choose between forwards & backwards should be tested with input `f`, `b`, and some incorrect, third option.



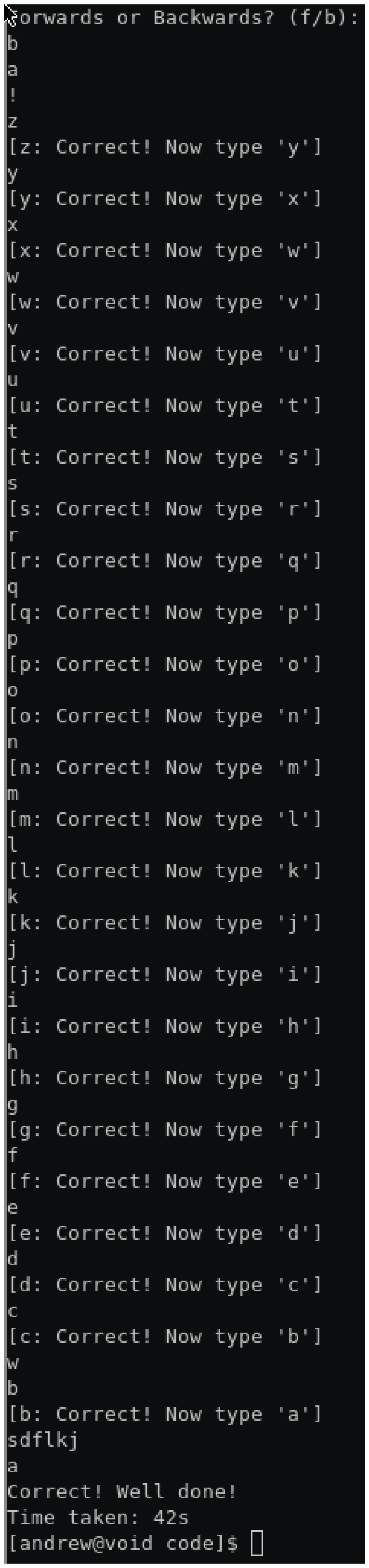
```
[andrew@void code]$ java Alphabet
Type the alphabet in order (lowercase).
Forwards or Backwards? (f/b):
testing invalid input
Invalid input!. You must enter either 'f' or 'b' to start.
[andrew@void code]$ 
```

Figure 1: Testing invalid input to the case statement



```
Forwards or Backwards? (f/b):
f
x
a
[a: Correct! Now type 'b']
b
[b: Correct! Now type 'c']
d
f
c
[c: Correct! Now type 'd']
d
[d: Correct! Now type 'e']
e
[e: Correct! Now type 'f']
f
[f: Correct! Now type 'g']
g
[g: Correct! Now type 'h']
h
[h: Correct! Now type 'i']
i
[i: Correct! Now type 'j']
j
[j: Correct! Now type 'k']
k
[k: Correct! Now type 'l']
l
[l: Correct! Now type 'm']
m
[m: Correct! Now type 'n']
n
[n: Correct! Now type 'o']
o
[o: Correct! Now type 'p']
p
[p: Correct! Now type 'q']
q
[q: Correct! Now type 'r']
r
[r: Correct! Now type 's']
s
[s: Correct! Now type 't']
t
[t: Correct! Now type 'u']
u
[u: Correct! Now type 'v']
v
[v: Correct! Now type 'w']
w
[w: Correct! Now type 'x']
x
[x: Correct! Now type 'y']
y
[y: Correct! Now type 'z']
d
s
z
Correct! Well done!
Time taken: 34s
```

Figure 2: Testing forwards with valid & invalid, first, middle, & last input



```
Forwards or Backwards? (f/b):
b
a
!
z
[z: Correct! Now type 'y']
y
[y: Correct! Now type 'x']
x
[x: Correct! Now type 'w']
w
[w: Correct! Now type 'v']
v
[v: Correct! Now type 'u']
u
[u: Correct! Now type 't']
t
[t: Correct! Now type 's']
s
[s: Correct! Now type 'r']
r
[r: Correct! Now type 'q']
q
[q: Correct! Now type 'p']
p
[p: Correct! Now type 'o']
o
[o: Correct! Now type 'n']
n
[n: Correct! Now type 'm']
m
[m: Correct! Now type 'l']
l
[l: Correct! Now type 'k']
k
[k: Correct! Now type 'j']
j
[j: Correct! Now type 'i']
i
[i: Correct! Now type 'h']
h
[h: Correct! Now type 'g']
g
[g: Correct! Now type 'f']
f
[f: Correct! Now type 'e']
e
[e: Correct! Now type 'd']
d
[d: Correct! Now type 'c']
c
[c: Correct! Now type 'b']
w
b
[b: Correct! Now type 'a']
sdflkj
a
Correct! Well done!
Time taken: 42s
[andrew@void code]$ 
```

Figure 3: Testing backwards with valid & invalid, first, middle, & last input