










KIT101 Programming Fundamentals

PP Task 4.2 Repetition, repetition, repetition

Overview


- Purpose:** Learn how to use conditional and iterative control flow statements in a program.
- Task:** Use the instructions on the following pages to complete the implementation of a typing practice program, and trace some small examples of iterative control flow. Submit to MyLO when complete.
- Learning Outcomes:** 1  2  3 
- Time:** Aim to complete this task before the start of Week 5.
- Resources:**
- Introductory Programming Notes:
 - 08 Making Decisions
 - 09 Repeating Actions with Loops
 - Appendix: Code Tracing Problems
 - Online Mini-lectures:
 - Making Decisions:
 -  Boolean expressions & comparing values
 -  Two-way branching with if and if-else
 -  Multi-way branching with switch
 - Repeating Actions with Loops:
 -  Loop components, while and do-while
 -  The for-loop and loop debugging tips
 - Code Tracing:  Selection & Repetition

Submission Details

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

- **Tracing tables** for the code samples at the start. This can be *either* photos/scans of hand-written tables (recommended) or a Word/PDF document
- Your **TypingTutor.java** source file (no need for program screenshots in this task)

Assessment Criteria

A  Completed submission will:

- Include valid tracing tables for the [code samples](#) in the warm up exercise, showing the execution order and output produced
- Include completed implementations of the `printHeading` and `runTutorial` methods
- Demonstrate the use of `for` and `while` loops, as appropriate to the task
- Require the user to practice a different word than in the starter code
- Follow the unit's coding style for layout, variable names (use of case), and commenting (including your name at the top)

Instructions

Warm up exercise: Trace code involving repetition

1. Create tracing tables for the following code samples. The tracing tables should illustrate the execution path each program took, both by the sequence of row numbers in the table and the output messages produced. Remember to include your name and the name of the task in the document(s) you submit.

Repetition Sample 1:

```
1  int counter = 5;
2  System.out.println("Message 0");
3  while (counter < 8) {
4      System.out.println("Message 1");
5      counter = counter + 1;
6  }
7  System.out.println("Message 2");
```

Repetition Sample 2:

```
1  int counter = 9;
2  System.out.println("Message 0");
3  while (counter < 8) {
4      System.out.println("Message 1");
5      counter = counter + 1;
```

```
6   }
7   System.out.println("Message 2");
```

Repetition Sample 3:

```
1   int counter = 10;
2   System.out.println("Message 0");
3   while (counter < 12) {
4       counter = counter + 2;
5       System.out.println("Message 1");
6       counter = counter - 1;
7   }
8   System.out.println("Message 2");
```

Implement a program involving repetition and selection

In the next, larger part of this task you will complete the implementation of a typing tutor program, which gives a user practice typing a word as quickly and as accurately as they can. In this program the word and number of required repetitions are fixed, but may be modified by changing the values of constants declared in the source code.

2. Download and extract the **4.2PP Repetition Starter Code** from MyLO.
3. Open **TypingTutor.java** in DrJava and read through the code. Although there's a lot to take in the program can be summarised as follows:

Declare:

- constants for the *word* to practice and *required number* of repetitions;
- variables to record the *start* and *end* time of a practice, and *total time in seconds*; and
- a variable to record the number of *attempts*

Display the program's title using the **printHeading()** method

Give instructions to the user

Perform the test by calling **runTutorial()** method

Report on the outcome of the test
Display a goodbye message using the `printHeading()` method

4. If you compile and run the program now you'll find it doesn't do much. No heading is actually displayed and the user never gets to practice typing anything (although they *do* appear to be very fast at doing nothing). You need to provide the implementations of `printHeading()` and `runTutorial()`.

Implement `printHeading`

5. Read the header comment for `printHeading()`, which formats its String-valued parameter to be all upper case with an underline produced by tildes (~, near the Esc key) and a trailing blank line. To illustrate its expected behaviour, if it were called with the String "sOMe HeAdiNg" then it should display:

```
SOME HEADING
~~~~~
```

(Note the blank line after the heading, which must appear in your program's output.)

6. Implement `printHeading` according to the following algorithm:

Method: `printHeading(String heading)`

Parameters:

String heading, *the heading to format and display*

Steps:

- 1 Display *heading* in upper case
- 2 For *i* is 0 to *heading.length()* - 1 (i.e., while *i* < *heading.length()*)
- 2-1 | Display a single ~, without ending the line
- 3 End the line
- 4 Display a blank line

Tip: Strings have a `toUpperCase()` method that returns a copy of the String in capital letters.

Tip: There is a variant of `System.out.println()` that takes no arguments and merely prints a newline, which ends the current line of text or produces a blank line if it was already at the start of a new line of output.

7. Compile and test the program. Are you getting the opening and ending headings now?

Implement `runTutorial`

8. The `runTutorial` method continuously prompts the user to type in a particular word until they have correctly typed it the required number of times. If they never get it right then the method will never end (and that's OK).

To illustrate its expected behaviour here is a sample interaction with the user if `runTutorial` were called with a `Scanner` object to read keyboard input, word equal to "Fred" and required equal to 3 (user input is **highlighted**):

```
Type 'Fred': Fred
Correct!
Type 'Fred': fred
Try again
Type 'Fred': Fred
Correct!
Type 'Fred': derf
Try again
Type 'Fred': Fred
Correct!
```

9. There's currently a minimal implementation of the method, which declares one of the required variables and returns its value at the end. Complete the implementation of `runTutorial` according to the following algorithm:

Method: `int runTutorial(Scanner sc, String word, int required)`

Returns:

`int`, *the total number of attempts made by the user*

Parameters:

`Scanner in`, *to read user's input*

`String word`, *the word the user must type correctly*

`int required`, *the number of times they must type it correctly*

Variables:

```
int attempts, the total number of attempts made by the user, initially 0
int correct, the number of correct entries so far, initially 0
String attempt, the current word entered by the user
```

Steps:

```
1      While correct < mustSolve do
1-1    |   Prompt user with "Type '" + word + "': "
1-2    |   Assign attempt value of next String from in (the Scanner)
1-3    |   Increment attempts (add 1 to value of attempts)
1-4    |   If attempt is equal to word then
1-4-1  |   |   Display "Correct!"
1-4-2  |   |   Increment correct (add 1 to value of correct)
1-5    |   Else
1-5-1  |   |   Display "Try again"
2      Return attempts
```

Tip: While testing your implementation you may wish to insert additional `println()`s to report the current values of `attempts` and `correct` (but if you do then remove these before submitting). You may also wish to decrease the required number of correct responses, which is defined as a constant in `main()`.

10. When you think it's working, **modify the word the user has to type**, which is defined in `main()`.

Without showing them your code (i.e., drag the interactions pane up to hide it) have another student try your program.

11. Ensure you've completed the [code tracing exercises](#) from the start and then upload the code traces and your completed source file to the submission folder for this task.
