Quiz Yourself: Comparing Loop Constructs (Intermediate)

Also in This Issue

QUIZ

Quiz Yourself: Comparing Loop Constructs (Intermediate)

Which loop construct would you use when looping on manual user input? Subtleties abound...

by Simon Roberts and Mikalai Zaikin

August 26, 2019

If you have worked on our quiz questions in the past, you know none of them is easy. They model the more difficult questions from certification examinations. The levels marked “intermediate” and “advanced” refer to the exams, rather than the questions. Although in almost all cases, “advanced” questions will be harder. We write questions for the certification exams, and we intend that the same rules apply. Take words at their face value and trust that the questions are not intended to deceive you, but straightforwardly test your knowledge of the ins and outs of the language.

The objective is to compare loop constructs.

Imagine that you are writing an interactive console application that does the following:

1. Prints a prompt
2. Reads a command as input
3. If the input was `Q`, exits; otherwise, executes the specified command and restarts at step 1

Which code construction would be the best option to implement the scenario? Choose one.

A. A `for` loop with a `break` statement
B. An enhanced `for` loop
C. A `while` loop with a `continue` statement
D. A `do`/`while` loop

Answer. This question asks you to make a value judgment and that’s often frowned upon in a multiple-choice exam. However, in this case, the judgment is fairly standard, and by the time the entire explanation has been presented, we’re inclined to think you’ll agree.

Two key observations can be made here that will guide selection of an appropriate loop. One is that steps 1 and 2 must be executed at least once; that is, the code is not expected to exit until after *at least one* command has been read. Another observation—closely related to the first—is that it’s not possible to make the decision to exit the program until after a command has been read.

Let’s consider how these requirements would affect your implementation if you try to use a `while` loop. The first observation is that a `while` loop performs its test right at the entry to the loop; that test is performed before the body of the loop is executed at all. Therefore, if the loop is used to control the program’s life and shutdown, one possibility would be to print a prompt and read a command before the loop starts, as in this pseudocode:

```
Issue prompt
Read command
While (command is not "Q")
  Execute the command
  ...
```

Now, that approach might not look like a problem, but you still need to issue subsequent prompts and read the subsequent commands. That must be done every time through the loop, so the relevant code must be inside the loop. That means the code must be in two places at once, which is an indirect way of saying that it must be duplicated. The code would look like this:

```
Issue prompt
Read command
While (command is not "Q")
  Execute the command
  Issue prompt // Duplicate
  Read command // Duplicate
End-while
```

Duplication of code is ugly and error-prone during maintenance. An alternative approach would be to set the command to some dummy value before the loop, and make sure that executing the dummy command does nothing. This would avoid duplication but it's still additional complexity, and it might be difficult to understand, because it uses special values to effectively "hack" its own behavior.

This problem must surely be a strike against the use of a `while` loop, but consider the other possibilities before you make your value judgment.

Next, let's consider the standard `for` loop. A `for` loop is really only a `while` loop with some decorations, and the decorations address two scenarios common in loop constructions. One is the initialization (and perhaps declaration) of some variables that will be used in the loop. The other is updating variables that change through the iterations. However, at its heart, a `for` loop is just a `while` loop with extras. It's possible that one of those extras might be useful in this scenario, but you still have the fundamental problem that you'll need to duplicate the prompting and input-reading code.

Next, let's consider the enhanced `for` loop. An enhanced `for` loop provides a cleaner means of processing every element from an `Iterable` object. (The target `Iterable` is commonly, but not necessarily, provided by a collection such as a `List`.) In the specified problem, items must be read from the console as input, and termination of the loop occurs when the input value is `Q`. By contrast, the enhanced `for` loop takes items from an `Iterable` and terminates when no more items are available. Although it's possible to create an `Iterable` object that prompts the user, reads a line of text, returns a `String` if the user's input wasn't `Q`, and terminates the iteration if the user's input was `Q`, this is convoluted and rather indirect for this scenario. There doesn't seem to be much to recommend the enhanced `for` loop in this situation, so let's consider the final option.

The remaining option is the `do`/`while` loop. The test that determines whether to iterate this loop is placed at the end of the `do`/`while` construction, which has consequences. Any `do`/`while` loop always executes at least once, because execution has to pass through the body of the loop to reach the test. Another perspective on this is to note that the body of the loop executes *before* the test is performed. This is much more appropriate for the proposed scenario. The prompt can be issued, and the command input can be read, in the body of the loop. The result, without any duplicated code, ensures that the prompt and input happen before the test, and the resulting structure will be clean and simple.

At this point, it's probably clear that option D fits the scenario perfectly and, as such, it's a much better choice than the others. Therefore, you can finally say that option D is correct and options A, B, and C are incorrect.

Here are a few side notes. Option A mentions using a `break` statement and option C mentions using a `continue`. A structure using `for` and a `break` could be pressed into service, but neither of these suggestions results in an elegant solution. Option A might be coded along these lines:

```
for (;;) { // effectively an infinite loop
    // Issue prompt
    String input = // read user input
    if (input.equals("Q"))
        break; // jump out of loop
    executeCommand(input);
}
```

However, this is still ugly compared to the use of a `do`/`while` loop. Similarly, the use of a `continue` in a `while` loop does not offer any obvious route to a clean solution.

The correct option is D .

Also in This Issue

Know for Sure with Property-Based Testing
Arquillian: Easy Jakarta EE Testing
Unit Test Your Architecture with ArchUnit
The New Java Magazine
For the Fun of It: Writing Your Own Text Editor, Part 1
Quiz Yourself: Using Collectors (Advanced)
Quiz Yourself: Threads and Executors (Advanced)
Quiz Yourself: Wrapper Classes (Intermediate)
Book Review: Core Java, 11th Ed. Volumes 1 and 2

Simon Roberts

Simon Roberts joined Sun Microsystems in time to teach Sun's first Java classes in the UK. He created the Sun Certified Java Programmer and Sun Certified Java Developer exams. He wrote several Java certification guides and is currently a freelance educator who publishes recorded and live video training through Pearson InformIT (available direct and through the O'Reilly Safari Books Online service). He remains involved with Oracle's Java certification projects.

Mikalai Zaikin

Mikalai Zaikin is a lead Java developer at IBA IT Park in Minsk, Belarus. During his career, he has helped Oracle with development of Java certification exams, and he has been a technical reviewer of several Java certification books, including three editions of the famous *Sun Certified Programmer for Java* study guides by Kathy Sierra and Bert Bates.

Share this Page