Programmer Thinking = Loops

Dr. William L Honig, Loyola University Chicago

Some thoughts to get you successfully started programming with loops. Programmers learn "patterns" or a standard way of doing something, especially with control structures. Some programming organizations require approval to do anything but use a pattern in a loop!

A. For Loop

The for loop is extremely flexible and can do many things; therefore, it is dangerous. To avoid bugs keep this in mind:

> Studies have shown that the most acceptable forms of for loops (to avoid off-by-one errors) are these:
>
> ```
> for ( int i = 0; i <= maxIndex; i++ )                          //for loop 1
> for ( int i = 0; i < sizeof(array or string or something); i++ ) //for loop 2
> for ( int i = maxIndex; i >= 0; i-- )                          //reverses loop 1
> ```
>
> A suitable expression will usually be used to calculate maxIndex or sizeof. You need a really good reason to write anything different!

Do not use the for loop for weird constructs such as

```
for (a = a / 2; count < ITERATIONS; System.out.println(xnew)) { . . .}
```

Make such a loop into a while loop. That way, the sequence of instructions is much clearer.

```
a = a / 2;
while (count < ITERATIONS) // OK
{  . . .
   System.out.println(xnew);
}
```

B. For (each) Loop

When possible, it's even better to use the for each version of the for loop to iterate through a collection of things. You do not need to worry about how many and how to index the elements of the collection. You only need to know the type of the collection's contents. The formal name of this sort of loop in Java is "the enhanced for" statement.

The general syntax of the for each loop is (note the use of :)

```
for ( Thing aThing : a collection of Things ) { . . . }
```

The aThing variable will have a new value (one of the Things in the collection of things) each time through the loop and the loop will run exactly the correct number of times (once for each Thing in the collection).

For example:

```
String[] poem;
. . .
for( String line : poem ){
        System.out.println(line);
}
```

Arrays work well with this form of for statement. So do all collections in general. To be formal, the expression on the right of the colon must evaluate to either an array or something that is an Iterable. See the Java documentation for the full details if you are really interested.
http://docs.oracle.com/javase/specs/jls/se8/html/jls-14.html#jls-14.14.2

About the only limitation on this approach to looping is that you cannot refer to the thing you are working on this time through the loop with a number. So if you really needed to know that the current thing is the 1st or 2nd or 25th thing, then use the older form of for loop.

C. Sentinel Loops

Often a loop will run for some unknown number of times until something happens. Generally, do not use a for loop for this case; for loops are better when you know or can calculate at run time the number of times something will need to loop.

The best approach is the sentinel loop using while. Think of the sentinel as a toll road gate that will let you into the loop until you run out of money.

```
value = get something;
while (value != sentinel) {
    process value or do other stuff;
    value = get something;
```

Here's an example getting values from the user until the user enters ## to indicate they are finished.

```
Scanner keyboard;
String line;
keyboard = new Scanner(System.in);

System.out.println("Enter a line of a poem (or ## when finished) ");
line = keyboard.nextLine( );

while (! line.equals("##")) {
    // TO DO add line to poem
    System.out.println("Enter a line of a poem (or ## when finished) ");
    line = keyboard.nextLine( );
}
```

Sentinel loops are often used to get user input but can also be used to get things from another program, process a collection, or anything where you keep doing it until something is found, occurs, fails, or succeeds.

Note that the pattern requires you to repeat the code to get the value both before the loop begins and at the end of the loop. This duplication may seem wasteful but studies have shown it is the very best way to avoid bugs!

Version 2.0 (October 2016) to remove one for pattern, minor text changes.
Programming Thinking: Loops, for Introduction to Object Oriented Programming, 2016