Comparison of Looping Structures in Java

We've been using the "enhanced for-loop" (sometimes called the "for-each" loop) to cycle through each item in a LinkedList. The for-each loop is used to access each successive value in a collection of values, and in general can be used to iterate over anything that implements the `Iterable<E>` interface. Here's a for-each loop that calculates the sum of the integers in a LinkedList of integers:

```
LinkedList<Integer> alon = new LinkedList<Integer>();

// some code to fill in alon with Integers

Integer result = 0;

for (Integer element: alon){
    result = result + element;
}
```

Java provides two other mechanisms for looping, the for-loop and the while-loop.

For loop

Here is the basic structure of a for-loop:

```
for (initializer; boolean expression; increment-statement){
  statements in for-block
}
```

where:

- initializer: initialize variables that will be used in the for-block
- boolean expression: determines if looping should continue
- increment-statement: a statement executed at the end of each execution of the for-block
- for-block: statements to execute every time the boolean expression is true

The for-loop executes by following these steps:

1. execute the initializer
2. evaluate the boolean expression; if it evaluates to
   (a) true, then
      i. execute the for-block
      ii. execute the increment-statement
      iii. go back to step 2
   (b) false, then jump to the first statement after the for-block

Rewriting our code for summing the numbers in an LinkedList using a for-loop gives us this:

```
Integer result = 0;

for (int index = 0; index < alon.size(); index = index + 1){
  result = result + alon.get(index);
}
```

Notice we're using the LinkedList method `get()` that allows access to a list element by its position in the list.

While loops

The other looping structure in Java is the while-loop:

```
while (boolean expression){
  statements in while-block
}
```

The while-loop works as follows:

1. Evaluate the boolean expression
2. If the expression is true, execute the statments in the while-block and go back to step 1
3. If the expression is false, jump to the first statement after the while block

So, our loop to sum the items in an LinkedList becomes:

```
Integer result = 0;
int index = 0;

while(index < alon.size()){
  result = result + alon.get(index);
  index = index + 1;
}
```

The while loop is essentially the same as the for-loop, except that the initializer statement and the increment-statement are not built in to the syntax of the while-loop, as they are in the for-loop. Instead, the initializer (in the example above, the statement `int index = 0;`) and the increment-statement (`index = index + 1;`) are coded as separate statements.

When should I use a particular looping structure?

The enhanced for-loop is the loop of choice in most situations, because it is easier to read and more self-documenting than the other two loops. But there are some situations when the enhanced-for cannot be used:

- The enhanced-for loop is meant to access elements only, not to change them. Do not use the enhanced-for to increment each element in a LinkedList, for example.
- The enhanced-for loop is meant for single element access only. If you want to compare successive elements, for example, use one of the other looping structures.
- The enhanced-for loop iterates over the structure in the "forward" direction, by single steps. If you want to process a structure from end to beginning, or process just every other element, you should not use the enhanced-for.