Java 8 idioms: In praise of helpful coding

The surprising benefits of a Java 8 convention

Venkat Subramaniam

May 30, 2017

A Java 8 convention for function composition could improve not only your code, but your relationship with other developers.

Expressiveness is one of the benefits of functional-style programming, but what does that mean for your code? In this article we compare examples of imperative and functional-style code, identifying qualities of expressiveness and conciseness. You'll see how these qualities support readability, and you'll consider an opposite example: when the quest for conciseness leads to unhelpful code. Finally, we examine Java 8's convention of vertically aligning dots in function composition. While this convention may be unfamiliar to some Java developers, a simple example demonstrates its value.

About this series
Java 8 is the most significant update to the Java language since its inception—packed so full of new features that you might wonder where to start. In this series, author and educator Venkat Subramaniam offers an idiomatic approach to Java 8: short explorations that invite you to rethink the Java conventions you've come to take for granted, while gradually integrating new techniques and syntax into your programs.

A surprising outcome

About a year after Java 8 was released, I posted a brief, uncontrolled survey on my website, inviting developers to participate. Each participant was presented with a snippet of code in either the imperative or the functional style, then asked to determine the code's behavior. I measured how long each visitor took to produce a response, comparing results from the two different code samples. The survey was open for 48 hours, and during that time more than 1,100 people participated. The results contained a few surprises.

Most developers, including your humble author, are very experienced with the imperative style of programming. Although the functional style has been around for a long time, it is unfamiliar to most Java programmers. Knowing this, it's not surprising that 82 percent of survey respondents who received the imperative code were able to determine its correct behavior. At the same time, only 75 percent of respondents who received the functional-style code got it right.

What surprised me, however, was the time it took respondents to understand the two code samples. The median time to figure out the imperative code was 30 seconds longer than the median time for the functional-style code.

Try this experiment at home

Functional-style code is more expressive and concise than imperative-style code—*if it is written well*. A simple example demonstrates this point. Before looking at the code samples below, please prepare a timer. Like my survey respondents, your task will be to figure out the details of the code. You will measure the time required for each sample.

All set? Start the timer and read the code below, then write down the behavior you would expect from it.

Listing 1. Imperative code sample

```
List<String> names = Arrays.asList("Jack", "Jill", "Nate", "Kara", "Kim", "Jullie", "Paul", "Peter");

List<String> subList = new ArrayList<>();
for(String name : names) {
  if(name.length() == 4)
    subList.add(name);
}

StringBuilder namesOfLength4 = new StringBuilder();
for(int i = 0; i < subList.size() - 1; i++) {
  namesOfLength4.append(subList.get(i));
  namesOfLength4.append(", ");
}

if(subList.size() > 1)
  namesOfLength4.append(subList.get(subList.size() - 1));

System.out.println(namesOfLength4);
```

How much time did you need to figure out this code? Don't be surprised if it took longer than you would expect. The time isn't a reflection of your ability so much as the poor quality of the code.

Now consider an equivalent sample, written in the functional style supported by Java 8:

Listing 2. Functional code sample

```
List<String> names = Arrays.asList("Jack", "Jill", "Nate", "Kara", "Kim", "Jullie", "Paul", "Peter");

System.out.println(
  names.stream()
    .filter(name -> name.length() == 4)
    .collect(Collectors.joining(", ")));
```

How much time would it take you to figure out this code? Obviously, you've already determined the purpose from Listing 1, so it's not a true experiment. If you want to really compare the samples, ask several pairs of colleagues to figure out one code sample or the other, then compare their response times.

Why functional-style coding matters

If you're familiar with Java 8, you probably had no trouble figuring out the code in Listing 2. Even if you aren't familiar with Java 8, you might have been able to wing it, thanks to the descriptive method names. You were also able to make sense of the code quickly, because it is considerably more concise than Listing 1.

Essentially, the code says: *given a collection of names, select only names of length 4, then join them together with a comma.*

The example is contrived, but it does illustrate the value of conciseness and expressiveness in coding. We see these qualities far more in functional-style code than in imperative code.

Writing readable code

Functional-style code is expressive and concise, which leads to programs that are both shorter and easier to read. Consider another example:

Listing 3. Imperative code sample

```
int result = 0;
for(int e : numbers) {
  if(e > 3 && e % 2 == 0 && e < 8) {
    result += e * 2;
  }
}
System.out.println(result);
```

Given the list called `numbers`, this code performs a sum of double of even numbers greater than 3 and less than 8, then prints the results. The code consists of seven lines, which we could potentially reduce by one or two lines.

Now consider the same code written in Java 8, using the functional style:

Listing 4. A functional-style alternative

```
 System.out.println(
  numbers.stream()
    .filter(e -> e  > 3)
    .filter(e -> e % 2 == 0)
    .filter(e -> e < 8)
    .mapToInt(e -> e * 2)
    .sum());
```

Listing 4 is also seven lines, but in this case it would not be helpful to reduce the code any further.

Functional-style code isn't always shorter than imperative-style code. What's more important is that it is expressive. It isn't helpful for code to be concise but hard to read.

Don't make this mistake

Functional-style code is designed to be more concise than imperative code, but that does not ensure it will be more readable. Consider the following example:

Listing 5. Functional code to join select names

```
System.out.println(names.stream().filter(name -> name.startsWith("J")).filter(name -> name.length() > 3)
  .map(name -> name.toUpperCase()).collect(Collectors.joining(", ")));
```

In Listing 5, `filter`, `map`, and other functional elements increase the expressiveness of the code. But you might notice that this code feels more terse than concise.

While it's only two lines, this code still requires considerable effort to read and understand. Your eyes strain to see where one function call ends and the next one begins. The code is extremely brief, but it was written tersely. There is only one reason to write such unhelpful code: the developer must hate everyone they work with.

Make your code concise, not terse

Terse code may be impressively brief, but it is still hard to read. Concise code is also brief, but is pleasant to read and easy to understand.

In programming, we can easily lose sight of the value of expressiveness and readability. Java 8 encourages these qualities by convention, suggesting that we align dots vertically for function composition.

Unfortunately, I've observed that programmers new to Java 8 frequently ignore this convention, even after multiple reminders. More experienced programmers should enforce the convention during code reviews. A good Java 8 IDE can also help, by providing shortcuts to using conventions such as this one.

Listing 6 shows what happens when we rewrite the code from Listing 5 using the alignment convention:

Listing 6. Function composition in Java 8

```
System.out.println(
 names.stream()
      .filter(name -> name.startsWith("J"))
      .filter(name -> name.length() > 3)
      .map(name -> name.toUpperCase())
      .collect(Collectors.joining(", ")));
```

Here we see the same terse code from Listing 5, but the dots are aligned vertically, and we've resisted the urge to combine multiple conditions into a single argument. As a result, each line is cohesive: narrow and focused, with a single, clear task.

A helpful convention

While it might seem dispensable, following Java 8's alignment convention can be highly beneficial.

- Code that follows this convention is easier to read, understand, and explain. We can quickly grasp the overall objective before examining each part in detail.

- Elements are clear and easy to locate, supporting faster modification. If we wanted to include another condition, or remove or modify an existing one, it would be relatively simple to locate the line and make the change.
- The code is easier to maintain, which conveys that we care about other developers on our team. Writing helpful code can greatly influence team morale, in addition to making code easier to maintain.

Conclusion

It is good practice to keep each line of your code short and concise, but going overboard can lead to code that is terse and hard to read. To improve expressiveness, ask yourself if the code is easy to understand. To improve readability, employ Java 8's convention of lining up dots vertically. Using these simple techniques, you will create functional-style code that is concise, expressive, and readable.

Related topics

- Functional Programming in Java: The Pragmatic Bookshelf, 2014
- Agile developer: Imperative vs. functional style survey
- IBM Code: Java journeys