Java 8 のイディオム: 役に立つコーディングの慣例への賞賛

Java 8 の慣例がもたらす意外なメリット

Venkat Subramaniam
Founder
Agile Developer, Inc.

2017年 9月 21日

Java 8 で慣例としている関数合成の表記は、作成するコードの品質を高めるだけでなく、他の開発者たちとの関係にも良い影響を与える可能性があります。

このシリーズの他の記事を見る

関数型スタイルのプログラミングがもたらすメリットの 1 つは表現力ですが、皆さんが作成するコードに、それはどのような意味を持つのでしょうか？この記事では、命令型スタイルと関数型スタイルのサンプル・コードを比較して、表現力と簡潔さという 2 つの特性を明らかにし、これらの特性がコードの読みやすさをサポートする仕組みを確認します。また、それとは逆に、簡潔さを追及しすぎて他の開発者の助けにならないコードになる例を見ていきます。そして最後に、ドットを縦に整列させて関数合成を表記するという Java 8 の慣例を取り上げます。このような表記は Java 開発者にとって馴染みのないものかもしれませんが、その価値は、単純な例で納得できるはずです。

このシリーズについて
Java 始まって以来の最も大々的な更新となっている Java 8 には、どこから手を付けてよいのか戸惑ってしまうほど新しい機能が満載されています。このシリーズでは、教育者である著者の Venkat Subramaniam がイディオムを考慮した Java 8 手法を簡潔に紹介し、当たり前のように思ってきた Java の慣例を見直して、プログラムに新しい手法と構文を徐々に取り込んでいくよう読者を導きます。

意外な結果

Java 8 がリリースされてから約 1 年後に、私は自分の Web サイトに簡単なアンケートを掲載して、開発者に自由参加を促しました。このアンケートは、各参加者に命令型スタイルまたは関数型スタイルのいずれかのコード・スニペットを提示し、そのコードの動作を判断してもらうというものです。参加者が回答するまでの時間を測定し、2 つの異なるスタイルのサンプル・コードに対する結果を比較しました。48 時間公開して 1,100 人を超える参加者が回答したこのアンケートの結果から、意外なことがいくつかわかりました。

ほとんどの開発者は (僭越ながら、私を含め)、命令型スタイルのプログラミングに豊富な経験があります。関数型スタイルが登場してからしばらく経っていますが、関数型スタイルは、Java プ

ログラマーの大半にとって馴染みがありません。この点を踏まえると、命令型のコードを提示されたアンケート回答者の 82 パーセントが正しい動作を判断できたのは当然の結果です。それと同時に、関数型スタイルのコードを提示された回答者のうち、正しい動作を判断できたのは 75 パーセントにとどまっているのも説明がつきます。

けれども私が驚いた点は、2 つのプログラミング・スタイルの間での回答時間の差です。コードを理解するまでの平均時間は、命令型スタイルのほうが関数型スタイルより 30 秒長くなっていました。

気軽に実験してみてください

関数型スタイルのコードがうまく作成されていれば、命令型スタイルのコードに比べ、より表現力に優れ、より簡潔なコードになります。この点を、単純な例によって明らかにします。以下のサンプル・コードを見て行く前に、タイマーを用意してください。私が行ったアンケートの回答者と同じく、皆さんのタスクは提示されたコードの詳細を理解することです。それぞれのサンプル・コードについて、コードの動作を理解するまでにかかった時間を測ってください。

準備は整いましたか？タイマーをスタートしたら、以下のコードを読んで、コードから予想される動作を書き留めてください。

リスト 1. 命令型のサンプル・コード

```
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);
```

このコードを理解するまでに、どれくらいの時間がかかりましたか？思ったより時間がかかったとしても、驚かないでください。理解するまでに時間がかかるのは、皆さんの能力の問題ではなくコードの品質が悪いためですから。

今度は、Java 8 でサポートされている関数型スタイルによって作成された同様のサンプル・コードを読んで、その動作を回答してください。

リスト 2. 関数型のサンプル・コード

```
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(", ")));
```

このコードの場合、理解するまでに、どれくらいの時間がかかりましたか？当然のことながら、コードの目的はリスト 1 ですでにわかっているので、これは真の意味での実験にはなりません。これらのサンプル・コードを真の意味で比較したいとしたら、一人の同僚に一方のサンプル・コードを提示し、別の同僚にもう一方のサンプル・コードを提示して理解してもらうという実験を何組かのペアで行って、回答時間を比較するという方法をとってください。

関数型スタイルのコーディングが重要となる理由

Java 8 に馴染みがあるとしたら、リスト 2 のコードを理解するのに何の問題もなかったことでしょう。Java 8 に馴染みがないとしても、説明的なメソッド名のおかげで、すぐにその目的を理解できたはずです。リスト 2 のコードを短時間で理解できた理由は、リスト 1 より遥かに簡潔なコードになっているからです。

基本的に、このコードの内容は「指定された名前の集合のうち、4 文字の名前だけを選択して、それらの名前をコンマで結合する」ことです。

この例は作為的なものですが、コーディングにおける簡潔さと表現力の価値を明確に示しています。関数型スタイルのコードは、命令型スタイルのコードよりもこれらの特性が大いに優れていることがわかります。

理解しやすいコードの作成

関数型スタイルのコードは表現力に優れ、簡潔であることから、プログラムが短くなるとともに、読んで理解するのも簡単になります。別の例を見てください。

リスト 3. 命令型のサンプル・コード

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

上記のコードは、`numbers` という名前のリストを基に、3 より大きく 8 より小さい偶数の値を 2 倍にした値を合計し、その結果を出力します。このコードは 7 行からなりますが、1 行か 2 行減らすことも不可能ではありません。

上記と同じ内容のコードを、関数型スタイルを使用して Java 8 で作成すると、以下のようになります。

リスト 4. 上記のサンプル・コードを関数型スタイルで作成した場合

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

リスト 4 も同じく 7 行のコードとなっていますが、この場合、これ以上コードを縮小しても有益なことはありません。

関数型スタイルのコードは常に命令型スタイルのコードよりも短いとは限りません。コードの長さよりも重要なのは、表現力です。コードを簡潔にしても、読みにくくなるのであれば、何の助けにもなりません。

避けるべき過ち

関数型スタイルのコードは、命令型のコードよりも簡潔になるよう意図されますが、簡潔にすることが、コードを読みやすくする結果につながるとは限りません。以下の例を見てください。

リスト 5. 選択した名前を結合する関数型のコード

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

リスト 5 では、`filter`、`map` などの関数型の要素がコードの表現力を高めています。けれどもこのコードを見ると、簡潔というよりも、素っ気ないという印象を受けたかもしれません。

たった 2 行であるにも関わらず、このコードを読んで理解するにはかなりの努力が必要です。目を凝らして見なければ、ある関数呼び出しが終わって次の関数呼び出しが始まる場所がわかりません。このコードは極めて短いものの、ぶっきらぼうに作成されています。このような扱いにくいコードを作成する理由は 1 つしかありません。それは、コードを作成した開発者が、一緒に働く他の開発者全員を嫌っているに違いないということです。

素っ気ないコードではなく、簡潔なコードにすること

素っ気ないコードは見事なほどに短くなるかもしれませんが、短くても理解するのに苦労します。簡潔なコードも同じく短いとは言え、苦労なく読み進めて簡単に理解できます。

プログラミングの際に、私たちは表現力と読みやすさにどれだけの価値があるのか見失いがちです。Java 8 ではこの 2 つの特性を慣例によって高めるために、関数合成を作成する際はドットを縦に並べることを推奨しています。

残念ながら私が見たところ、Java 8 をよく知らないプログラマーは何度注意されてもこの慣例を無視する傾向があります。したがって、経験を積んだプログラマーが、コードのレビュー時にこの慣例が適用されるようにする必要があります。優れた Java 8 IDE ではこのような慣例を使用するためのショートカットを提供しているので、そのような Java 8 IDE を利用するという手もあります。

リスト 6 に、この整列表記の慣例に従ってリスト 5 のコードを作成し直した場合のコードを示します。

リスト 6. Java 8 での関数合成

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

上記に示されているコードはリスト 5 の素っ気ないコードと同じですが、ドットを縦に整列させてあり、複数の条件を単一の引数に結合したい気持ちを抑えてあります。その結果、各行がまとまりのある内容になっています。つまり、限定されて、フォーカスを絞ったそれぞれの行で、単一の明確なタスクに対応しています。

役に立つ慣例

Java 8 の整列表記の慣例はなくても困らないように思えますが、この慣例に従うと非常に大きなメリットをもたらす可能性があります。

- この慣例に従ったコードは読んで理解するにも、説明するにも簡単です。コードの各部分を詳細に調べる前に、コードの目標全体を迅速に把握できます。
- 要素の場所を確実かつ簡単に特定できることから、コードの変更にかかる時間が短縮されます。別の条件を含める場合や、既存の条件を削除または変更する場合に、該当する行を比較的簡単に見つけて変更を加えることができます。
- コードが簡単に保守できるようになっていれば、チームの他の開発者への心遣いが伝わります。他の開発者の助けになるコードを作成することで、コードの保守が容易になるだけでなく、チームの士気が大いに高まります。

まとめ

コードの各行を短く簡潔にすることは優れた慣例ですが、やり過ぎると、コードがぶっきらぼうで理解しにくいものになってしまう可能性があります。表現力を高めるには、そのコードを簡単に理解できるかどうかを自問自答してください。より読みやすいコードにするためには、Java 8 のドットを縦に並べるという表記を採用することをお勧めします。これらの単純な手法によって、簡潔で表現力に優れ、読んで理解しやすい関数型スタイルのコードを作成できるようになります。

関連トピック： Javaによる関数型プログラミング (オライリージャパン、2014年) Agile developer: Imperative vs. functional style survey IBM Code: Java journeys