Use Precise Java Method Parameters

javacodegeeks.com/2017/06/use-precise-java-method-parameters.html

Learn how to pick the right method parameter types and get more robust and shorter code in your Java applications.

We Java developers generally have a bad habit of using method parameters without thinking of what is actually needed and just picking whatever we are used to, what we have available or whatever first comes into mind. Consider the following representative example:

```
01                String poem(Map<Integer, String> numberToWord)
    private static {
02          return new StringBuilder()
03                      "There can be only
                .append("                    )
04              .append(numberToWord.get(1))
05                      " of
                .append(you.\n"     )
06                      "Harts are better of when there are
                .append("                                    )
07              .append(numberToWord.get(2))
08                      " of them
                .append(together.\n"          )
09                      "These
                .append("       )
10              .append(numberToWord.get(3))
11                      " red roses are a symbol of my love to
                .append(you.\n"                                      )
12              .toString();
13      }
```

When we use the method above, we provide a Map that translates from a number to a String. We might, for example, provide the following map:

```
1  Map<Integer, String> englishMap
   =                                 new HashMap<>();
```

```
2          englishMap.put(1,"one");
```

```
3          englishMap.put(2,"two");
```

```
4          englishMap.put(3,"three");
```

When we call our poem method with the englishMap then the method will produce the following output:

```
1  There can be only one of
   you.
```

```
2  Harts are better of when there are two of them
   together.
```

```
3  These three red roses are a symbol of my love to
   you.
```

That sounds good. Now suppose that your significant other is a computer nerd and you want to spice up your poem and make an impression, then this is the way to go:

```
1  Map<Integer, String> nerdMap
   =                              new HashMap<>();
```

```
2          nerdMap.put(1,"1");
```

```
3          nerdMap.put(2,"10");
```

```
4          nerdMap.put(3,"11");
```

If we now submit our nerdMap to the poem method, it will produce the following poem:

```
1  There can be
   only             1 of you.
```

```
2  Harts are better of when there
   are                                10 of them together.
```

```
3         red roses are a symbol of my love to
   These11 you.
```

As with all poems, it is difficult to judge which poem is more romantic that the other but I certainly have my own personal view.

The Problems

There are several problems with the solution above:

First of all, as an outside caller, we cannot be sure that the poem method does not change the Map we provide. After all, we provide a Map and there is nothing preventing a receiver to do whatever possible with the map, even clearing the entire map altogether. This can of course be avoided by wrapping the Map using the Collections.unmodifiableMap() method or provide a copy of an existing map whereby the copy is later discarded.

Secondly, we are tied to use a Map when we only need something that translates from an integer to String. This might create unnecessary code in some cases. Think back of our nerdMap, where the values in the map could easily be computed using the Integer::toBinaryString instead of mapping them manually.

The Solution

We should strive to provide precisely what is needed in any given situation and not more. In our example we should modify the poem method to take a function that goes from an integer to a String. How this function is implemented on the caller side is of less importance, it can be a map or a function, or code or something else. Here is how it should be done in the first place:

```
01                String poem(IntFunction<String> numberToWord)
    private static {
02          return new StringBuilder()
03                      "There can be only
                .append("                    )
04              .append(numberToWord.apply(1))
05                      " of
                .append(you.\n"     )
06                      "Harts are better of when there are
                .append("                                    )
07              .append(numberToWord.apply(2))
08                      " of them
                .append(together.\n"          )
09                      "These
                .append("       )
10              .append(numberToWord.apply(3))
```

```
11                      " red roses are a symbol of my love to
                .append(you.\n"                                    )
12              .toString();
13      }
```

If we want to use the poem method with a Map, we simply call it like this:

```
1
2       System.out.println(poem(englishMap::get));
1
```

If we want to compute the values like we did for the the nerd poem then we can do it even simpler:

```
1  System.out.println(poem(Integer::toBinaryString));
```

Heck, we can even produce a poem to a significant other suffering from a dual personality disorder like this:

```
1  System.out.println(
2          poem(
3              no -> englishMap.getOrDefault(no   , Integer.toString(no
               +                                 1+                      1))
4          )
5      );
```

This will produce the following poem:

```
1  There can be only two of
   you.
2  Harts are better of when there are three of them
   together.
3        red roses are a symbol of my love to
   These4 you.
```

Be careful with your method parameters!

Reference: Use Precise Java Method Parameters from our JCG partner Per Minborg at the Minborg's Java Pot blog.