May 11, 2016 | Vojtech Ruzicka

Break the Java Generics Naming Convention?

April 30, 2016 by

Choosing descriptive, intent-revealing names is one of the key principles of writing clean code, which is easy to read and understand and usually requires little to no comments. According to the official convention, generic type parameter names are supposed to be just one capital letter. Should you break this convention to make your names more descriptive?

Java Generics Naming Convention

The official Sun/Oracle Java naming convention for generic type parameters is:

> By convention, type parameter names are single, uppercase letters. This stands in sharp contrast to the variable naming conventions that you already know about, and with good reason: Without this convention, it would be difficult to tell the difference between a type variable and an ordinary class or interface name.
> The most commonly used type parameter names are:
>
> - E – Element (used extensively by the Java Collections Framework)
> - K – Key
> - N – Number
> - T – Type
> - V – Value
> - S,U,V etc. – 2nd, 3rd, 4th types

There is not much information than can be contained in just one letter. Even with just one type parameter it might be sometimes hard to quickly tell what exactly it represents. When there are more parameters, it is usually much worse. And the only reason stated why to follow that convention (and have cryptic type parameter names) is to be able to distinguish regular classes and generic types. So what are the options to make your type parameter names more readable?

Option 1. Follow the convention, explain in JavaDoc

Let's say you don't want to break the convention, but still want more information about what the S, U or V actually means. Your only option is basically to document it using comments, preferably JavaDoc. However, I prefer to contain as much information as possible in method, variable and class names and document only special cases or tricky parts (as long as it's not publicly exposed API). Usually you can contain everything you need in identifier names and when you really need to write some comments, it is a red flag. That usually means that the code is too complicated, method has too many responsibilities, you are using a hacky solution etc. In that case it is time to refactor. There is a whole chapter on this topic in Clean Code book, which summarizes this topic nicely. I suggest avoiding this option unless you really want to stick to the original convention. If so, at least make sure the class/method has minimal number of type parameters. It will be much easier to read. Also, a big number of those parameters might be a sign of Single Responsibility Principle violation.

Option 2. Descriptive name with a prefix or suffix

If you prefer more descriptive names, but still want to be able to instantly recognize generic types, this option is for you. After choosing a nice descriptive name, certain suffix (or prefix) is added to that name. So instead of K,V you have KeyT and ValueT. Or KeyType and ValueType. In the recent Google Java Style Guide, it is possible to use either only letter convention or descriptive name with T appended such as ListenerT. I personally like Type appendix more than T, since it is not much longer and it reads better. However, I would recommend sticking to the Google Style Guide as it is widely recognized convention and can be seen as a more up-to-date version of the old Sun naming convention. There are also some suggestions like $ prefix to type parameters ($Listener) by Tim Boudreau, but I don't find that instantly recognizable as a generic type and would rather choose the Google Style Guide.

Option 3. Same naming conventions as regular classes

If the sole reason for having special convention for generic types is to be able to tell them apart from regular types, I think that reason is now longer valid. If you are using a modern IDE such as IntelliJ Idea, the generic types should be marked with a different color. It is easy to recognize them and you can follow the usual class naming conventions – eg. use Key and Value instead of K and V. I think this approach is the best, because it uses descriptive names with known convention without adding any extra unnecessary garbage as appending Type suffix.

```
public class Box<Content extends Serializable> {

    private Content content;

    public Content getContent() {
        return content;
    }

    public void setContent(Content content) {
        this.content = content;
    }
}
```

IntelliJ Idea 2016.1 – type parameter has a different color

Conclusion

While I think that sticking to official conventions is very important, there are rare occasions, where it is better to go with a different one. In my opinion generics type naming convention is one of them. Having clear and descriptive names is way more important than following the old convention. There are various approaches described in this post, which improve readability of the types. Is does not matter that much which one you choose, but more important is when you choose it, the whole team should stick to it and follow it as a new convention. If you have any other interesting approaches how to solve the generics naming problem, I will be glad to see the in the comments bellow.

Filed Under: Java
