

[New Guide] Download the 2018 Guide to IoT: Harnessing Device Data

[Download Guide](#) ▶

Type-safe Data Views Using Abstract Document Pattern

by Emil Forslund  MVB · Jan. 19, 16 · Java Zone · Opinion

Heads up...this article is old!

Technology moves quickly and this article was published 2 years ago. Some or all of its contents may be outdated.

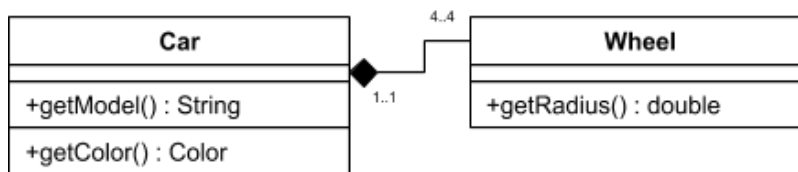
Take 60 minutes to understand the Power of the Actor Model with "Designing Reactive Systems: The Role Of Actors In Distributed Architecture". Brought to you in partnership with Lightbend.



How do you organize your objects? In this article I will introduce a pattern for organizing so called noun-classes in your system in a untyped way and then expose typed views of your data using traits. This makes it possible to

get the flexibility of an untyped language like JavaScript in a typed language like Java, with only a small sacrifice.

Every configuration the user does in your UI, every selection in a form need to be stored someplace accessible from your application. It needs to be stored in a format that can be operated on. The school-book example of this would be to define classes for every noun in your system, with getters and setters for the fields that they contain. The somewhat more serious way of doing the school-book model would be to define enterprise beans for every noun and process them using annotations. It might look something like this:



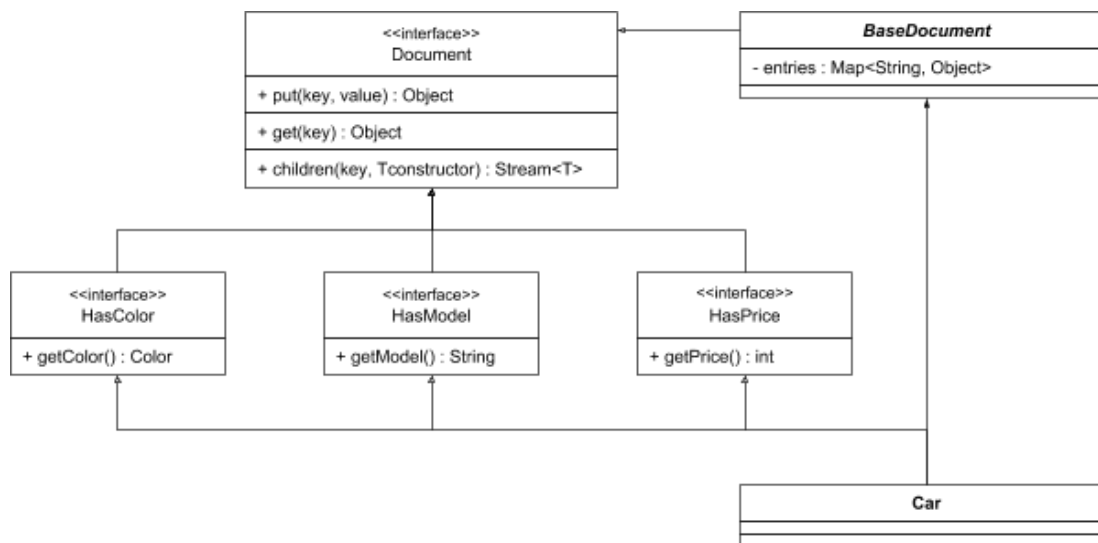
There are limitations to these static models. As your system evolves, you will need to add more fields, change the relations between components and maybe create additional implementations for different purposes. You know the story. Suddenly, static components for every noun isn't as fun anymore. So then you start looking at other developers. How do they solve this? In untyped languages like JavaScript, you can get around this by using Maps. Information about a component can be stored as key-value pairs. If one subsystem need to store an additional field, it can do that, without defining the field beforehand.

```

1  var myCar = {model: "Tesla", color: "Black"};
2  myCar.price = 80000; // A new field is defined on-the-fly
  
```

It accelerates development, but at the same time comes with a great cost. You lose type-safety! The nightmare of every true Java developer. It is also more difficult to test and maintain as you have no structure for using the component. In a recent refactor we did at Speedment, we faced these issues of static versus dynamic design and came up with a solution called the *Abstract Document Pattern*.

Abstract Document Pattern



A Document in this model is similar to a Map in JavaScript. It contains a number of key-value pairs where the type of the value is unspecified. On top of this *un-typed* abstract document is a number of Traits, micro-classes that express a specific property of a class. The traits have *typed* methods for retrieving the specific value they represent. The noun classes are simply a union of different traits on top of an abstract base implementation of the original document interface. This can be done since a class can inherit from multiple interfaces.

Implementation

Implementation

Let's look at the source for some these components:

Document.java

```
1  public interface Document {
2      Object put(String key, Object value);
3
4      Object get(String key);
5
6      <T> Stream<T> children(
7          String key,
8          Function<Map<String, Object>, T> constructor
9      );
10 }
```

BaseDocument.java

```
1  public abstract class BaseDocument implements Document {
2
3      private final Map<String, Object> entries;
4
5      protected BaseDocument(Map<String, Object> entries) {
6          this.entries = requireNonNull(entries);
7      }
8
9      @Override
10     public final Object put(String key, Object value) {
11         return entries.put(key, value);
12     }
13
14     @Override
15     public final Object get(String key) {
16         return entries.get(key);
17     }
18
19     @Override
20     public final <T> Stream<T> children(
21         String key,
22         Function<Map<String, Object>, T> constructor) {
23
24         final List<Map<String, Object>> children =
25             (List<Map<String, Object>>) get(key);
26
27         return children == null
28             ? Stream.empty()
29             : children.stream().map(constructor);
30     }
31 }
```

HasPrice.java

```
1 public interface HasPrice extends Document {
2
3     final String PRICE = "price";
4
5     default OptionalInt getPrice() {
6         // Use method get() inherited from Document
7         final Number num = (Number) get(PRICE);
8         return num == null
9             ? OptionalInt.empty()
10            : OptionalInt.of(num.intValue());
11     }
12 }
```

Here we only expose the getter for price, but of course you could implement a setter in the same way. The values are always modifiable through the put()-method, but then you face the risk of setting a value to a different type than the getter expects.

Car.java

```
1 public final class Car extends BaseDocument
2     implements HasColor, HasModel, HasPrice {}
```

As you can see, the final noun class is minimal, but you can still access the color, model and price fields using typed getters. Adding a new value to a component is as easy as putting it into the map, but it is not exposed unless it is part of an interface. This model also works with hierarchical components. Let's take a look at how a HasWheels-trait would look.

HasWheels.java

```
1 public interface HasWheels extends Document {
2     final String WHEELS = "wheels";
3
4     Stream getWheels() {
5         return children(WHEELS, Wheel::new);
6     }
7 }
```

It is as easy as that! We take advantage of the fact that in Java 8 you can refer to the constructor of an object as a method reference. In this case, the constructor of the Wheel-class takes only one parameter, a Map<String, Object>. That means that we can refer to it as a Function<Map<String, Object>, Wheel>.

Conclusion

There are both advantages and of course disadvantages with this pattern. The document structure is easy to expand and build upon as your system grows. Different subsystems can expose different data through the trait-interfaces. The same map can be viewed as different types depending on which constructor was used to generate the view. Another advantage is that the whole object hierarchy exists in one single Map which means that it is easy to serialize and deserialize using existing libraries, for example Google's gson tool. If you want the data to be immutable, you can simply wrap the inner map in an `unmodifiableMap()` in the constructor and the whole

hierarchy will be secured.

One disadvantage is that it is less secure than a regular beans-structure. A component can be modified from multiple places through multiple interfaces which might make the code less testable. Therefore you should weigh the advantages against the disadvantages before implementing this pattern on a larger scale.

If you want to see a real-world example of the *Abstract Document Pattern* in action, take a look at the source code of the Speedment project where it manages all the metadata about the users' databases.

Learn how the Actor model provides a simple but powerful way to design and implement reactive applications that can distribute work across clusters of cores and servers. Brought to you in partnership with Lightbend.

Like This Article? Read More From DZone



Object Identity and Equality in Java



Knowing the New Java 8 Features: Streams




Base64 Encoding in Java 8



**Free DZone Refcard
Getting Started With Kotlin**

Topics: [JAVA](#) , [JAVA8](#) , [PATTERN](#)

Published at DZone with permission of Emil Forslund , DZone MVB. [See the original article here.](#) 
Opinions expressed by DZone contributors are their own.

Get the best of Java in your inbox.

Stay updated with DZone's bi-weekly Java Newsletter. [SEE AN EXAMPLE](#)

SUBSCRIBE

Java Partner Resources

Level up your code with a Pro IDE

JetBrains



Microservices for Java Developers: A Hands-On Introduction to Frameworks & Containers

Red Hat Developer Program



Build vs Buy a Data Quality Solution: Which is Best for You?

Melissa Data



Reactive Microsystems - The Evolution of Microservices at Scale

Lightbend



The Road to Jakarta EE

by David Delabassée  MVB · Apr 25, 18 · Java Zone · Opinion

Download *Microservices for Java Developers: A hands-on introduction to frameworks and containers*. Brought to you in partnership with Red Hat.

The Eclipse Foundation is making multiple announcements related to Jakarta EE that includes the unveiling of <https://jakarta.ee> and the Jakarta EE logo, the results of the developer survey, etc. It's probably a good time to reflect on how we got here...

End 2016, early 2017, I was jokingly using the following slide when I was discussing Java EE.

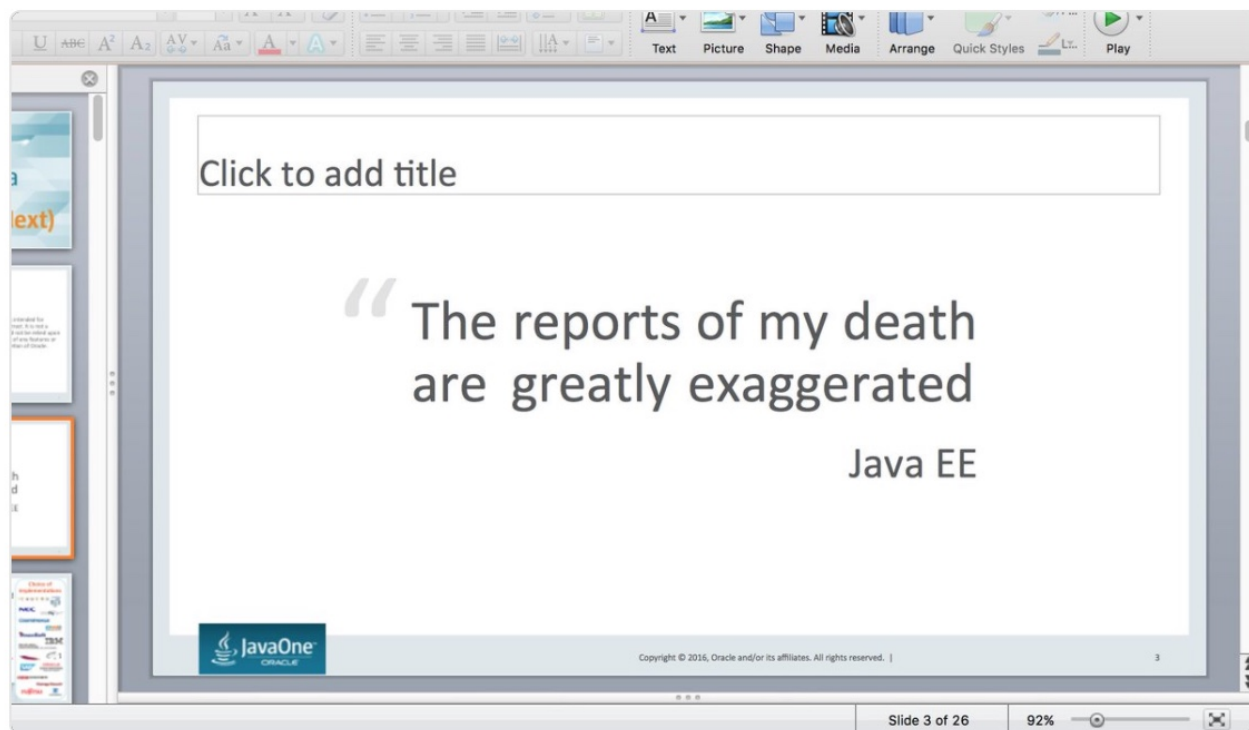


David Delabassée

@delabassée



Just stumbled upon a slide that I was using last year #EE4J



7:28 PM - 18 Oct 2017 from Republic of Serbia

19 Retweets 40 Likes



 Java EE Platform



19



40

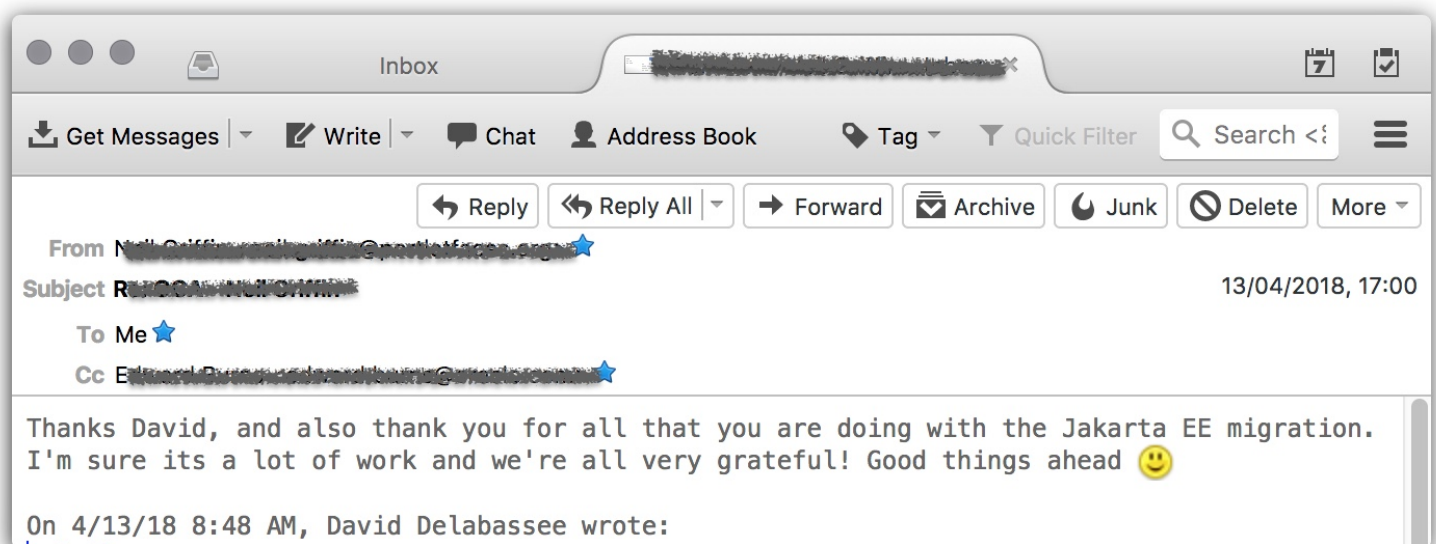


I am relatively confident that at that time no one would have predicted what would happen. You have to remember that during that period, things were not so simple for Java EE as questions were raised about the future of the platform. At that time, Oracle along with the different JCP experts were focused on finalizing Java EE 8 and its various APIs.

Relatively soon in the process we also started to think about the future of the platform, i.e. 'Java EE Next' as we informally called the post-Java EE 8 era. The Java EE 9 plans that were shared at JavaOne 2016 weren't particularly well received so we went back to the drawing board. It was clear that we had to do something radically different to get the Java EE ecosystem excited and engaged again.

To lift all the concerns that were raised on Java EE through the years, we thought that the platform should, from that point on, evolve in a more open fashion and at a more rapid pace. It was clear that the platform had to adopt an open source model including a well-established governance. Early in those reflections, Oracle along with some key players from the ecosystem, namely Red Hat and IBM, decided that the Eclipse Foundation would be the obvious venue to host this radical evolution. One of the many reasons is that the Eclipse is already hosting the MicroProfile.IO project, which is itself augmenting the Java EE platform with capabilities geared towards Microservices Architecture. Shortly after that, additional Java EE players such as Tomitribe, Payara and Fujitsu joined the initiative. And that's in a nutshell how EE4J Jakarta EE came to life.

Transitioning the development of the platform to the Eclipse Foundation is a huge undertaking. It involves many technical and non-technical aspects including complex legal issues that I won't cover here given IANAL! In addition, we are not talking about a small project; we are talking about a large collection of established projects that includes GlassFish, Jersey, Grizzly, Mojarra, Open MQ to name just a few. And that's not all, there are also all the activities related to the opening of the various TCKs. It is simply a huge effort and probably the largest project that the Eclipse Foundation has ever embarked on (see here for some background). This is one of the reasons why it was decided early on that Jakarta EE would use Java EE 8 as its baseline and that older versions of the platform would not be part of Jakarta EE; that approach was simply reasonable and pragmatic. All those efforts happen as we speak and while we would all prefer that works to be behind us so that we can all effectively focus on the key goal of Jakarta EE, i.e. evolving the platform, we still have to wait that everything is in place. On that note, I have recently received the following mail from a well-known community member.



We were discussing a matter unrelated to Jakarta EE and while I sincerely appreciate the gratitude from that person, I really need to stress something about the whole Jakarta EE effort. Some of us are clearly more visible in the community (ex. Dmitry Kornilov who represents Oracle in the PMC and me as an evangelist) but Jakarta EE is really a team effort on the Oracle side. There are many people who are working behind the (Oracle) scene to transition Java EE to the Eclipse Foundation. It is impossible to mention all my colleagues that are, closely or remotely, involved in this effort. The list is simply too long and I don't want to take the risk of omitting someone. I, however, want to publicly acknowledge the work of Ed Bratt, Will Lyons, and Bill Shanon who deserve a particular mention as they have been working tirelessly since the early days of this effort to make sure Jakarta EE happens! So thanks to you all!

You should also realize that usually when a project is open-sourced, all the related activities, including all the legal aspects, are happening upstream and it is only when everything is discussed, agreed and done that the project is made public. But early on, we have decided to be as transparent as possible, which is why we have announced our initial intent last summer. At that time, lots of things were not decided yet and that lead us to where we are today, i.e. in the early days of Jakarta EE including the creation of a new but already actively engaged open-source community. A lot of work still needs to happen to properly tackle the ultimate goal of Jakarta EE, i.e. evolve the platform towards an open-source and Java-based, Cloud Native foundation that will be relevant for the next decade. The Jakarta EE community is actively working towards that goal and today's announcement represent an important initial milestone!

Download Building Reactive Microservices in Java: Asynchronous and Event-Based Application Design. Brought to you in partnership with Red Hat.

Like This Article? Read More From DZone



How Java EE Can Get Its Groove Back



Opening Up Java EE: An Update




Java EE Has a New Name....



**Free DZone Refcard
Getting Started With Kotlin**

Topics: JAVA, JAKARTA EE, JAVA EE, ECLIPSE FOUNDATION

Published at DZone with permission of David Delabassee , DZone MVB. [See the original article here.](#) 
Opinions expressed by DZone contributors are their own.
