Records & their constructors

May 8, 2020 • Sebastian Teumert • Java

A look at the constructors of records in Java 14, and how one can leverage the formal parameter list of records to already enable libraries like Jackson to work with records (and in this example, deserialize JSON to records).

Records will be re-previewed in Java 15 without any changes as part of [JEP 384] and are expected to become a regular feature of the language in Java 16.

Records can have regular constructors, just like classes

Records can have regular constructor. The constructor without formal parameter list, which is a new feature in records and helps combat boilerplate by leveraging auto-initialization, but they can also have regular constructors with a formal parameter list, just like classes.

A simple record might be given as:

```
record FooBar(String foo, List<String> bars) { }
```

And then we can add sanity checks – in this case just using the constructor without formal parameter list.

```
record FooBar(String foo, List<String> bars) {
    public FooBar {
        if (foo == null || foo.isBlank())
              throw new IllegalArgumentException("foo can not be null or
blank");
    }
}
```

Without a formal parameter list, values are assigned after the constructor has run. So one can initialize fields differently and **doesn't** need to assign them via `this.x = x`.

```
record FooBar(String foo, List<String> bars) {
    public FooBar {
        if (foo == null || foo.isBlank())
            throw new IllegalArgumentException("foo can not be null or
blank");
        if(bars == null)
            bars = new ArrayList<>();
    }
}
```

`bars` **will** be assigned correctly and a call to `bars()` will return the newly created list.

```
System.out.println(new FooBar("FooBar", null));
// prints FooBar[foo=FooBar, bars=[]]
```

However, assigning `bars` inside the constructor to `this.bars` via `this.bars = bars;` is *heavily discouraged*, in fact so much so that it is a point of active discussion on the amber-spec-experts mailing list to remove access to fields via `this.x` in the canonical constructor altogether (both reads & writes) [amber-spec-experts].

However, you can also use a constructor with a formal parameter list. When doing so, auto-initialization does *not* work and values *must* be assigned in the constructor (since the fields are final):

```
record FooBar(String foo, List<String> bars) {
    public FooBar (String foo, List<String> bars) {
        if (foo == null || foo.isBlank())
            throw new IllegalArgumentException("foo can not be null or
blank");
        if(bars == null)
            bars = new ArrayList<>();
        this.foo = foo;
        this.bars = bars;
    }
}
```

Why would you ever want to do that? Because you can put annotations on them. Currently, libraries like Jackson do not support records, but work to make this happen is underway [3]. Using formal parameter lists for the constructor lets us put annotations on them and get around that limitation:

```
record FooBar(String foo, List<String> bars) {
    public FooBar(
            @JsonProperty String foo,
            @JsonProperty List<String> bars) {
        if (foo == null || foo.isBlank())
            throw new IllegalArgumentException("foo can not be null or
blank");
        if(bars == null)
            bars = new ArrayList<>();
        this.foo = foo;
        this.bars = bars;
    }
}
```

Yes, this actually works right now, in Java 14, with Jackson 2.11.0.

```
public record FooBar(String foo, List<String> bars) {
    public FooBar(@JsonProperty("foo") String foo,
         @JsonProperty("bars") List<String> bars) {
        if (foo == null || foo.isBlank())
            throw new IllegalArgumentException("foo can not be null or
blank");
        if (bars == null)
            bars = new ArrayList<>();
        this.foo = foo;
        this.bars = bars;
    }

    public static void main(String[] args) throws JsonProcessingException {
        System.out.println(new ObjectMapper()
            .readValue("{\"foo\" : \"foo\", \"bars\": []}", FooBar.class));
        // prints out FooBar[foo=foo, bars=[]]
    }
}
```