The record: migrate to immutability

May 7th, 2025 Johan Hutting / Devoxx UK

do your thing

The Java record, a short introduction

New language feature as of Java 16: *classes that act as transparent carriers for immutable data*

This is not syntactic sugar, the bytecode is different from a “normal” class!

You will get for free:

- A canonical constructor
- `toString()`
- `hashCode()`
- `equals()`

Take for example a Box record/class:

```
record Box(String label, int width, int height, int depth) { }
Box box = new Box( label: "Example",  width: 210,  height: 297,  depth: 100);
```

Use cases for records

What records offer:

- Reduce cognitive load transporting data in a PoJo between components
- Memory/performance benefits on high volume systems
- Getting rid of Lombok

You will... run into having to fix side-effects of existing code. But that can be a good thing!

Do not use records when you:

- Need to hide fields
- Require mutable fields

“Mutable” records

Records only offer *shallow* immutability, meaning only the supplied fields are immutable!

Meaning: if a field is an Object with mutable content, that can still be changed.

Take for example a List field that contains an ArrayList:

```
record Box(String label, int width, int height, int depth, List<Item> contents) { }
```

```
Box box = new Box( label: "Example",  width: 210,  height: 297,  depth: 100, new ArrayList<>());
```

Defensive constructors

You can override the original constructor (please don't!) or extend the constructor:

```
record DefensiveBox(String label, int width, int height, int depth, List<Item> contents) {
    DefensiveBox {  no usages
        // the values are already assigned in the normal constructor!
        // however, they are still treated as variables at this point.
        if (label == null || label.isBlank()) { // validate business logic
            label = "None";
        }

        contents = List.copyOf(contents); // immutable copy
    }
}
```

Validate only constructors

A better option: throw exceptions on “illegal” input values

```
record ValidatedBox(String label, int width, int height, int depth, List<Item> contents) {
    ValidatedBox { // the values are already assigned in the normal constructor!  no usages
        try {
            contents.addAll(List.of());
            throw new IllegalArgumentException("Contents should be an immutable list");
        } catch (UnsupportedOperationException uoe) { } // expected behaviour

        if (label == null || label.isBlank()) { // validate business logic
            throw new IllegalArgumentException("The box requires a filled label");
        }
    }
}
```

Copying records

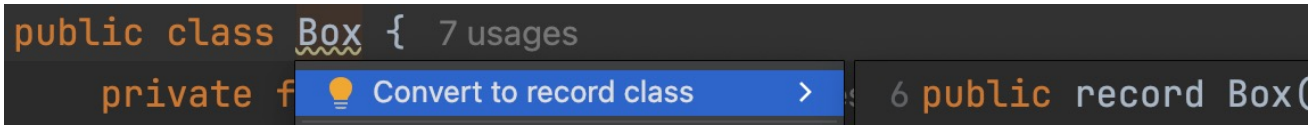
Copy with small changes: use the `withValue` pattern, otherwise stick to Builders.

```
public BoxWithBuilder withLabel(String label) {  no usages  new *
    return new BoxWithBuilder(label, width, height, depth, contents);
}

public static class Builder {  7 usages  jhutting *
    private String label;  3 usages
    private int width, height, depth;  3 usages
    private List<Item> contents = new ArrayList<>();  5 usages

    public Builder(BoxWithBuilder original) {  no usages  jhutting
        this.label = original.label;
        this.width = original.width;
        this.height = original.height;
        this.depth = original.depth;
        contents.addAll(original.contents);
    }
    public Builder addContent(Item item) {...}
    public Builder removeContent(Item item) {...}
    public Builder replaceContent(List<Item> items) {...}
    public Builder setLabel(String newLabel) {...}
    public Builder setHeight(int newHeight) {...}
    public Builder setWidth(int newWidth) {...}
    public Builder setDepth(int newDepth) {...}
    public BoxWithBuilder build() { return new BoxWithBuilder(label, width, height, depth, List.copyOf(contents)); }
}
```

Migrating

The most common approach will be manual migrations: you will also need to adjust code for side-effects that occurred over time.

If you use Intellij and your class matches the requirements:



If you're using Lombok's @Value there is an OpenRewrite recipe available called `LombokValueToRecord` that will work for many cases.
It will only migrate the @Value annotation, the other ones such as @Slf4j and @Builder remain.
Method references such as Box::getHeight also will have to be migrated manually.

Should you migrate?

It depends / YMMV

(use them where they add value)

Take a deeper dive?

I wrote two blog articles on records:
https://codeadventures.littlebluefrog.nl/
posts/06-immutability-in-records/
posts/07-copying-records

do your thing

Thank you!

Feedback is appreciated

@JohanHutting.bsky.social

www.ing.com