From Factory and Registry to Builder Using Lambda Expressions

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR
@JosePaumard https://github.com/JosePaumard

Agenda

Let us go deeper in the design patterns

How to use the previous principles

To design a Factory, a Registry

And a Builder

What Is a Factory?

A Factory is an object able to create other objects

It can be modeled by a Supplier

Demo

Let us create a factory with a Supplier

A Supplier can a be factory

More functionalities can be added

Using default methods

It can be made a Singleton too

Creating Registries Using Builders

A registry can also build other objects

```java
public Shape buildShape(String shape) {

   switch(shape) {
      case "square"   : return new Square();
      case "triangle": return new Triangle();
      case "circle"   : return new Circle();
      default:
         throw new IllegalArgumentException("Unknown shape " + shape);
   }
}
```

This is an easy to understand pattern, and easy to implement

Problem: you need to know the shapes at compile time…

What about making it dynamic?

Adding elements dynamically to a registry

Can be achieved with a Builder Pattern:

1) add elements to the registry

2) build the registry and seal it

There are many examples of this pattern in the JDK: Stream.Builder

```java
Stream.Builder<String> builder = Stream.builder();
builder.add("one");
builder.add("two");
builder.add("three");

Stream<String> stream = builder.build();
stream.forEach(System.out::println);
```

Example of the StreamBuilder

First, create a Stream.Builder object

Then add elements to the builder

Then build the stream

There are several problems here:

- two phases
- the builder has to know the factory

In fact the factory needs the builder

But not the contrary

```java
public class Builder<T> {

    public void add(String label, Supplier<T> supplier) {
        ...
    }
}
```

The builder can be made independent of the factory

```
public class Registry<T> {

    public T createFactory(String label) {}

    public static <T> Registry<T> build(Builder<T> builder) {}
}
```

This is the registry

That can be created using a factory method

Taking the builder as a parameter

Demo

Let us implement this builder

And build our registry of factories

Dynamically!

The factory / builder / registry elements

Can be modeled with functional interfaces

Implemented using lambdas

Module Wrap Up

What did you learn?

How to implement complex patterns

Using lambda expressions

It brings security, robustness

And performances!