Spring - Beans vs Lombok vs Records

Spis treści

Spring vs Lombok

Argumentem za stosowaniem wstrzykiwania poprzez pole, korzystając z adnotacji `@Autowired`, jest ograniczenie boilerplate code'u, jaki wymuszają na nas inne rodzaje wstrzykiwania. A jak wiadomo, uproszczony kod, to jest to, co lubimy. Czytelność kodu w znacznym stopniu poprawia jakość pracy dewelopera, co przekłada się na jakość aplikacji, a to lubimy jeszcze bardziej. Jednak zalecanym rodzajem wstrzykiwania jest to poprzez konstruktor, a nie poprzez pole.

W jaki sposób możemy sobie to uprościć? Jeżeli Springa uzupełnimy o bibliotekę **Lombok**, będziemy mieli możliwość wygenerowania potrzebnych nam konstruktorów, stosując jedynie adnotacje. I tak dzięki Lombokowi wstrzykiwanie poprzez konstruktor staje się atrakcyjniejsze. Jakie adnotacje się przydadzą? `@AllArgsConstructor` i `@RequiredArgsConstructor`. Musisz pamiętać, że od wersji Springa **4.3**, w przypadku kiedy Bean posiada tylko jeden konstruktor, nie ma potrzeby oznaczania go adnotacją `@Autowired`. Jest to robione automatycznie przez samego Springa. Skoro jest jeden, to Spring wie, że to własne tego ma użyć, do utworzenia obiektu.

```
package pl.zajavka;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import lombok.RequiredArgsConstructor;

@Component
@RequiredArgsConstructor
class ExampleBean {

    private final InjectedBean injectedBean;

    // ... nie musimy pisać konstruktora ręcznie, bo Lombok nam go wygeneruje

}
```

Co natomiast zrobić, kiedy będziemy mieli kilka konstruktorów? Wystarczy rozszerzyć Lombokową adnotacje o parametr *onConstructor*, który zadba o dodatnie adnotacji `@Autowired` do wygenerowanego konstruktora, tak jak w przykładzie poniżej.

Przykład użycia onConstructor:

```
package pl.zajavka;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import lombok.RequiredArgsConstructor;

@Component
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
class ExampleBean {

    private final InjectedBean injectedBean;

    // ...
}
```

Spring vs Java Records

Skoro omówiliśmy już zestawienie Springa z Lombokiem, może pojawić Ci się pytanie: *A co z rekordami?* Java przecież wprowadziła rekordy, żeby programista mógł wyraźnie oznaczyć, które klasy mają być rozumiane jako przechowywacze danych. I tutaj pojawia się pewna filozoficzna kwestia.

Które klasy powinniśmy dodawać do Spring contextu, a które z nich powinny być zarządzane przez nas ręcznie?

Patrząc od strony technicznej, Spring może zarządzać wszystkimi rodzajami obiektów: obiektami, które przechowują dane oraz obiektami, które służą do realizowania logiki biznesowej. W praktyce natomiast został wypracowany taki sposób pracy ze Springiem, że jest on używany do zarządzania obiektami realizującymi jakąś logikę biznesową (czyli np. serwisy), natomiast obiekty, które przechowują dane, są zarządzane przez programistę ręcznie, czyli obiekty takie są tworzone ręcznie, przy wykorzystaniu np. konstruktora, albo wzorca Builder.

Na tej podstawie można wywnioskować, że rekordy nie będą zarządzane przez Spring context, gdyż służą one do modelowania danych. Przy wykorzystaniu rekordów nie piszemy klas realizujących logikę biznesową.

Podsumowanie

Spring jest narzędziem, którego celem jest ułatwienie tworzenia oprogramowania. Zbiera to, co jest powtarzalne w tworzeniu systemów. Dzieli na moduły i pozwala na dowolne ich użycie i mieszanie w zależności od konkretnych potrzeb. Ułatwia konfiguracje, bo dostarcza jej domyślną wersję, ale nadal pozwala ją modyfikować. Spring upraszcza także kluczowy element, jakim jest konfiguracja klas będących serwisami. W pełni wspiera je w całym cyklu ich życia. Daje programiście szkielet, do którego może wpinać moduły tworząc zaawansowaną aplikację spełniającą wymogi biznesowe.