Notatki - Lombok - Adnotacje cz.1

Spis treści

Adnotacje - cz.1

Gettery i Settery

Biorąc na przykład klasy, w której mamy zdefiniowaną bardzo dużą ilość pól. Przykłady będziemy pokazywać na przykładzie piesków ☺.

Klasa Dog bez użycia Lombok

```
package pl.zajavka;

public class Dog {

    private final String name;

    private final Integer age;

    private final Owner owner;

    public Dog(final String name, final Integer age, final Owner owner) {
        this.name = name;
        this.age = age;
        this.owner = owner;
    }

    public String getName() {
        return name;
    }

    public Integer getAge() {
        return age;
    }

    public Owner getOwner() {
        return owner;
```

```
    }

}
```

Zwróć uwagę, że kod, który jest odpowiedzialny za konstruktor i gettery (nie ma setterów, bo pola są `final`) zajmuje więcej miejsca niż pola, które faktycznie realizują naszą "potrzebę biznesową". Utrudnia to jednocześnie czytelność tego kodu.

Wykorzystajmy zatem Lombok:

Klasa Dog z użyciem adnotacji @Getter

```
package pl.zajavka;

import lombok.Getter;

@Getter
public class Dog {

    private final String name;

    private final Integer age;

    private final Owner owner;

    public Dog(final String name, final Integer age, final Owner owner) {
        this.name = name;
        this.age = age;
        this.owner = owner;
    }

}
```

Widać, że kod skrócił się już w znaczny sposób. W ten sam sposób możemy określić settery (`@Setter`). W tym przypadku, nawet jeżeli pola są określone jako `final`, możemy dodać adnotację `@Setter` oprócz adnotacji `@Getter`. Natomiast gdy będziemy chcieli użyć taki setter, czyli zwyczajnie go wywołać, to zobaczymy, że nie mamy takiej metody dostępnej. Wynika to z tego, że pola są `final`, czyli nie możemy ponownie przypisać do nich wartości po utworzeniu obiektu.

Adnotacje `@Getter` i `@Setter` mogą zostać również napisane nad konkretnym polem. W przypadku poniżej określamy, że nie chcemy mieć gettera dla pola `name` i faktycznie go nie będzie. Osiągamy to poprzez dodanie dopisku `AccessLevel.NONE`. Ta informacja może być rozumiana jako modyfikator dostępu gettera, gdzie w tym przypadku możemy również napisać, że dane pole ma nie mieć gettera w ogóle. Możemy ustawić również inny **AccessLevel**, mówiący, że getter może być `PUBLIC`, `PROTECTED`, `PACKAGE`, lub `PRIVATE`. To samo ma zastosowanie dla adnotacji `@Setter`.

```
public class Dog {

    @Getter(AccessLevel.NONE)
    private final String name;

    //... pozostałe elementy klasy
}
```

Należy tutaj również dodać, że jeżeli będziemy dodawać lub usuwać pola z klasy to nie musimy nic więcej robić. gettery i settery są generowane na etapie kompilacji. Więc na etapie pisania możemy założyć, że nic nie musimy robić bo zostaną one wygenerowane automatycznie na podstawie określonej definicji/konfiguracji w momencie kompilacji. Nie musimy sami dodawać lub usuwać getterów lub setterów jak zmienimy coś w klasie. A normalnie byśmy musieli ☺.

Lombok dostarcza również dokumentację, w której możemy zobaczyć przykłady: getter i setter. W dokumentacji stwierdzenie `Vanilla Java` oznacza Javę bez użycia Lomboka.

@NonNull

Adnotacja `@NonNull` może być używana, jeżeli chcemy dodać sprawdzenie, czy wartość nie jest `null` i jeżeli jest to wyrzucić wyjątek **NullPointerException**. Adnotacja ta może być używana na polu, parametrze metody lub konstruktora. Zostanie wtedy automatycznie wygenerowane sprawdzenie, czy pole nie jest `null`.

Przykład z wykorzystaniem Lombok

```
package pl.zajavka;

import lombok.NonNull;

public class Person {

    private String name;

    public Person(@NonNull String name) {
        this.name = name;
    }
}
```

Przykład bez wykorzystania Lombok

```
package pl.zajavka;

import lombok.NonNull;

public class Person {

    private String name;

    public Person(@NonNull String name) {
        if (name == null) {
            throw new NullPointerException("name is marked @NonNull but is null");
        }
        this.name = name;
    }
}
```

Link do dokumentacji `Lombok` @NonNull.

Konstruktory

Kolejną wartością dodaną jest możliwość stosowania adnotacji generujących konstruktory. Każdy z tych wygenerowanych konstruktorów robi to co robi konstruktor - przypisuje wartości pól w obiekcie.

@NoArgsConstructor

Adnotacja `@NoArgsConstructor` dodaje konstruktor bezparametrowy, przykład poniżej:

```
@Getter
@Setter
@NoArgsConstructor
public class Dog {

    private String name;

    private Integer age;

    private Owner owner;

}
```

Zwróć uwagę, że pola nie są już `final`, dlaczego? Bo jeżeli zrobimy je teraz `final` to dostaniemy błąd kompilacji wynikający z tego, że pola `final` nie mają przypisanej wartości. Dzieje się tak bo konstruktor domyślny nie przypisuje wartości domyślnych polom. Jeżeli chcemy wymusić wartość domyślną na tych polach, należy zastosować `@NoArgsConstructor(force = true)`. Wtedy pola zostaną zainicjowane odpowiednio wartościami domyślnymi `0` / `false` / `null`.

Jeżeli natomiast zastosujemy taką kombinację:

Wykorzystanie adnotacji @NoArgsConstructor(force = true)

```
@Getter
@Setter
@NoArgsConstructor(force = true)
public class Dog {

    private final String name;

    private int age;

    private final Owner owner;

}
```

Spowoduje to wygenerowanie takiego kodu:

Analogiczny przykład kodu bez wykorzystania Lombok

```
public class Dog {

    private final String name;
```

```
    private int age;

    private final Owner owner;

    public Dog() {
        this.name = null;
        this.owner = null;
    }

    public String getName() {
        return this.name;
    }

    public Integer getAge() {
        return this.age;
    }

    public Owner getOwner() {
        return this.owner;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

Nie ma potrzeby przypisywania wartości `age` w konstruktorze, gdyż domyślnie jest to 0.

Link do dokumentacji `Lombok` @NoArgsConstructor.

@RequiredArgsConstructor

Adnotacja `@RequiredArgsConstructor` generuje konstruktor z polami klasy, które są albo `final`, albo oznaczone jako `@NonNull`. Do pól `@NonNull` dodawane jest również sprawdzenie czy pole nie jest `null`. Czyli konstruktor wyrzuci **NullPointerException** jeżeli przekażemy wartość jakiegoś pola jako `null`. Kolejność pól w konstruktorze zgadza się z kolejnością pól w klasie.

Wykorzystanie adnotacji @RequiredArgsConstructor

```
@Getter
@Setter
@RequiredArgsConstructor
public class Dog {

    @NonNull
    private String name;

    private Integer age;

    private final Owner owner;

}
```

Kod bez wykorzystania Lombok

```
public class Dog {
```

```
    @NonNull
    private String name;

    private Integer age;

    private final Owner owner;

    public Dog(@NonNull String name, Owner owner) {
        this.name = name;
        this.owner = owner;
    }

    public @NonNull String getName() {
        return this.name;
    }

    public Integer getAge() {
        return this.age;
    }

    public Owner getOwner() {
        return this.owner;
    }

    public void setName(@NonNull String name) {
        this.name = name;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

Zwróć uwagę, że adnotacja `@RequiredArgsConstructor` stworzyła konstruktor tylko dla parametrów oznaczonych jako `@NonNull` lub `final`. Pole `age` nie zostało wykorzystane.

Link do dokumentacji Lombok @RequiredArgsConstructor.

@AllArgsConstructor

Adnotacja `@AllArgsConstructor` generuje konstruktor z każdym polem w naszej klasie. Dla pól `@NonNull` dodawane jest sprawdzenie czy pole nie jest `null`. @AllArgsConstructor

Delombok

Jeżeli na którymś etapie zaczniesz zastanawiać się, co tak właściwie zostanie na końcu wygenerowane albo skąd są brane przykłady tego jak będzie wyglądał kod bez wykorzystania `Lombok`. Można to osiągnąć stosując `Lombok` plugin, który w zakładce `Refactor` (w paskach na górze ekranu), daje mi opcje *Delombok* > *All lombok annotations*.

staticName

Pamiętasz klasy `LocalDateTime` itp? A konkretnie sposób tworzenia tych obiektów? Wykorzystywana była metoda `of()`. `Lombok` daje nam możliwość dodania takiej metody przy wykorzystaniu adnotacji. Robi się

to w ten sposób:

Wykorzystanie Lombok i metody of()

```
@Getter
@Setter
@AllArgsConstructor(staticName = "of")
public class Dog {

    private String name;

    private Integer age;

    private final Owner owner;

}
```

Dzięki fragmentowi `staticName = "of"` możemy teraz napisać kod w ten sposób:

```
public class Example {
    public static void main(String[] args) {
        Dog dog = new Dog(); ①
        Dog dog2 = Dog.of("name", 12, null); ②
    }
}
```

① Ten konstruktor przestaje być dostępny, staje się on prywatny.

② Od teraz możemy ten obiekt tworzyć w ten sposób.