Notatki - Lombok - Adnotacje cz.2

Spis treści

Adnotacje - cz.2

@ToString

Od tego momentu nie musimy już generować tej metody w Intellij. Wystarczy dodać coś takiego:

```
@Getter
@Setter
@AllArgsConstructor(staticName = "of")
@ToString
public class Dog {
    // kod klasy
}
```

Jeżeli teraz użyjemy pluginu, żeby zrobić **Delombok**, zobaczymy taką metodę:

```
public class Dog {
    // kod klasy

    public String toString() {
        return "Dog(name=" + this.getName() +
            ", age=" + this.getAge() +
            ", owner=" + this.getOwner() +
            ")";
    }
}
```

Możemy również jawnie podać, które pola mają zostać wyłączone z metody `toString()`.

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

    @ToString.Exclude
    private String name;
```

```
    private Integer age;

}
```

Jeżeli teraz użyjemy pluginu, żeby uruchomić **Delombok**, zobaczymy taką metodę:

```
public class Dog {
    // kod klasy

    public String toString() {
        return "Dog(age=" + this.getAge() + ")";
    }
}
```

Możemy również zastosować adnotację `@ToString(onlyExplicitlyIncluded = true)`, a wraz z nią adnotację `@ToString.Include`. Przykład:

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

    @ToString.Include
    private String name;

    private Integer age;

}
```

Przy wykorzystaniu **Delombok**, dostaniemy taki kod:

```
public class Dog {
    // kod klasy

    public String toString() {
        return "Dog(name=" + this.getName() + ")";
    }
}
```

Sama adnotacja `@ToString` pozwala stosować więcej parametrów niż tutaj poruszone. W tym celu zapraszam do poczytania dokumentacji @ToString ☺.

@EqualsAndHashCode

Od teraz nie musimy już również generować metod `equals()` oraz `hashCode()` z poziomu Intellij. `Lombok` pomoże nam również w tym. W tym celu należy dodać adnotację `@EqualsAndHashCode`.

```
@Getter
@Setter
@AllArgsConstructor(staticName = "of")
```

```
@ToString
@EqualsAndHashCode
public class Dog {

    @ToString.Exclude
    private String name;

    private Integer age;

}
```

Przy wykorzystaniu **Delombok**, zobaczymy teraz coś takiego:

```
public class Dog {
    // kod klasy

    public boolean equals(final Object o) {
        if (o == this) return true;
        if (!(o instanceof Dog)) return false;
        final Dog other = (Dog) o;
        if (!other.canEqual((Object) this)) return false;
        final Object this$name = this.getName();
        final Object other$name = other.getName();
        if (this$name == null ? other$name != null : !this$name.equals(other$name)) return false;
        final Object this$age = this.getAge();
        final Object other$age = other.getAge();
        if (this$age == null ? other$age != null : !this$age.equals(other$age)) return false;
        return true;
    }

    protected boolean canEqual(final Object other) {
        return other instanceof Dog;
    }

    public int hashCode() {
        final int PRIME = 59;
        int result = 1;
        final Object $name = this.getName();
        result = result * PRIME + ($name == null ? 43 : $name.hashCode());
        final Object $age = this.getAge();
        result = result * PRIME + ($age == null ? 43 : $age.hashCode());
        return result;
    }
}
```

Mamy również do dyspozycji `@EqualsAndHashCode.Include` oraz `@EqualsAndHashCode.Exclude`. Jeżeli stosujemy `@EqualsAndHashCode.Include` należy wykorzystać adnotację `@EqualsAndHashCode(onlyExplicitlyIncluded = true)` analogicznie jak w `@ToString`.

Tak samo jak w przypadku adnotacji `@ToString`, mamy możliwość określenia, czy mamy wołać metodę `super.equals()` oraz `super.hashCode()` jeżeli dziedziczymy z jakiegoś obiektu. Z racji rzadkości używania tej kombinacji nie poruszam jej tutaj. Jeżeli będzie to potrzebne, zapraszam do przeczytania dokumentacji @EqualsAndHashCode ☺.

@With

Lombok pozwala nam w bardzo szybki sposób dodać metody `withX()`, gdzie `X` jest nazwą pola, do którego generujemy daną metodę. Metoda ta pozwala nam zmienić wartość pola (jak setter), ale jednocześnie możemy takie metody chainować. Należy jednak pamiętać, że w przypadku Lomboka, każda metoda `with()`, zwraca kopię poprzedniego obiektu ze zmienionym tym jednym polem, wrócimy jeszcze do tego.

Jeżeli realizowałeś/realizowałaś z nami bootcamp, pamiętasz pewnie projekt kalkulatora kredytu hipotecznego. Pamiętasz, że pisaliśmy tam ręcznie metody **with**, przykładowo `withName()`? `@With` jest podobne do tamtych metod.

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

    @ToString.Include
    private String name;

    private Integer age;

}
```

Dzięki temu dostajemy teraz podobne zachowanie. Piszę podobne, gdyż w projekcie o kredycie hipotecznym, metoda `with()` była zdefiniowana w ten sposób:

```
public class Dog {
    public InputData withType(MortgageType type) {
        this.rateType = type;
        return this;
    }
}
```

Widać tutaj, że `with()` był wzbogaconym setterem. Po wywołaniu `with()`, operowaliśmy dalej na tym samym obiekcie, gdyż zwracaliśmy `this`. Inaczej mówiąc, operowaliśmy na obiekcie mutowalnym, czyli zmieniającym stan. Przy stosowaniu adnotacji `@With` tworzymy kopię poprzedniego obiektu z nową wartością. Nie mutujemy obiektu, tylko tworzymy nowy.

Żeby pokazać różnicę, zdefiniujmy metodę `with()` w klasie `Dog` w ten sposób:

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

    private String name;

    public Dog withName(String name) {
        this.name = name;
        return this;
```

```
    }

}
```

I użyjemy tej metody:

```
public class DogRunner {

    public static void main(String[] args) {
        Dog dog = Dog.of("Burek");
        Dog dog2 = dog.withName("Romek");
        System.out.println(dog);
        System.out.println(dog2);
    }
}
```

To na ekranie wydrukuje się dwukrotnie:

```
Dog(name=Romek)
Dog(name=Romek)
```

Jeżeli natomiast zastosujemy adnotację `@With`, to przy tym samym wywołaniu, na ekranie wydrukuje się:

```
Dog(name=Burek)
Dog(name=Romek)
```

Czyli w poprzednim przypadku operowaliśmy cały czas na tym samym obiekcie w pamięci. W przypadku adnotacji `@With`, najpierw tworzymy nowy obiekt będący kopią obiektu oryginalnego i dopiero tę kopię modyfikujemy.

Dokumentacja adnotacji @With