Podstawy Java

Adnotacje

O adnotacjach

Adnotacje to dodatkowe informacje o programie, które niekoniecznie znajdują się w programie. Adnotacje nie mają bezpośredniego wpływu na operacje które adnotują. Adnotacje mają następujące zastosowanie:

- Są informacją dla kompilatora,
- Niektóre narzędzia IDE mogą przetwarzać adnotacje i generować kod,
- Niektóre adnotacje mogą być używane i przetwarzane podczas pracy programu.

Adnotacje - przykład użycia

Stworzymy adnotację, która będzie informacją dodatkową w kodzie, którą przetworzymy podczas pracy programu. Stworzymy następującą adnotację:

```
@Target(value = ElementType.FIELD)
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MinMaxValue {
    int minValue();
    int maxValue() default 10;
}
```

Deklaracja adnotacji posiada przedrostek **@interface**. Wewnątrz adnotacji możemy deklarować dwa pola, mogące zwracać domyślne (**default**) wartości. Zamysłem za stworzeniem tego typu adnotacji jest użycie jej w programie jako adnotacji dla zmiennych, których zakres ma być ograniczony wartościami od **minValue** do **maxValue**. Ponieważ chcemy, aby adnotacja była użyta wyłącznie jako adnotacja pól, konieczne jest dodanie adnotacji **@Target**:

```
@Target(value = ElementType.FIELD)
```

Ponieważ będę chciał używać wartości tej adnotacji w kodzie, muszę użyć adnotacji:

```
@Retention(value = RetentionPolicy.RUNTIME)
```

Adnotacje - przykład użycia

Do użycia adnotacji posłużę się klasą TeddyBear:

```
public class TeddyBear {
    @MinMaxValue(minValue = 10)
    private int age;

    public TeddyBear(int age) {
        setAge(age);
    }

    public int getAge() {
        return age;
    }

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

Nad polem jedynego pola klasy dodałem adnotację **@MinMaxValue** której zadaniem jest przechowywanie informacji o zakresie minimalnym i maksymalnym tego pola. Do pola **minValue** które nie miało domyślnej wartości konieczne było przypisanie wartości w miejscu użycia. Ponieważ chciałbym aby walidacja tej wartości przebiegła za każdym razem po zmianie wieku obiektu klasy **TeddyBear**, dokonam tej weryfikacji w setterze tego pola.

```
public void setAge(int age) {
    this.age = age;
    try {
        if(!TeddyBearAgeValidator.validate(this)){
            throw new IllegalArgumentException("Niepoprawna wartość wieku!");
        }
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    }
}
```

Adnotacje - przykład użycia

Jedynym brakującym elementem jest walidator, czyli mechanizm weryfikacji wartości pola **age** misia, oraz wartości dopisanych do adnotacji tego pola:

```
public class TeddyBearAgeValidator {
    public static boolean validate(Object o) throws IllegalAccessException {
        // z obiektu pobieram jego pola
        Field[] fields = o.getClass().getDeclaredFields();
        // dla każdego pola robie sprawdzenie czy posiada moja adnotacje
        for (Field field : fields) {
            MinMaxValue annotation = field.getAnnotation(MinMaxValue.class);
            if (annotation != null) {
                // dla adnotacji weryfikuje czy istnieje (jeśli nie istnieje: annotation == null)

                field.setAccessible(true); // upubliczniam metodę
                int age = (int) field.get(o); // pobieram wartość pola field z obiektu o
                if (age < annotation.minValue() || age > annotation.maxValue()) { // weryfikuje wartość z min/max z adnotacji
                    return false; // wartość wykracza poza zakres
                }
                field.setAccessible(false); // uprywatniam metodę
            }
        }
        return true;
    }
}
```

Adnotacje - przykład użycia

Przykład użycia:

```
TeddyBear tb = new TeddyBear(5);
System.out.println(tb.getAge());
```

Zwróci poprawny wynik i nie wystąpi błąd podczas wykonania programu. Natomiast zmiana wartości parametru konstruktora obiektu **tb** na wartość przekraczającą 10 (z domyślnej wartości **maxValue** adnotacji **MinMaxValue**) spowoduje błąd podczas wykonania:

```
Exception in thread "main" java.lang.IllegalArgumentException: Niepoprawna wartość wieku!
    at com.sda.exercises.TeddyBear.setAge(TeddyBear.java:19)
    at com.sda.exercises.TeddyBear.<init>(TeddyBear.java:8)
    at com.sda.exercises.Main.main(Main.java:8)
```

Adnotacje - zadanie

Treść:

Stwórz aplikację która ma adnotację "MaxLength" która mówi o maksymalnej długości znaku w podanym String'u (polu). Stwórz dowolną klasę z polem typu String i spróbuj użyć swojej adnotacji. Następnie stwórz klasę która zajmuje się weryfikacją wartości podczas pracy programu (weryfikacja dodana do settera pola adnotowanego).

Adnotacje - przykład użycia adnotacji z Projektu Lombok

Innym typem adnotacji są te, które generują kod programu. Przykładem adnotacji tego typu są te, które należą do biblioteki Javy z projektu Lombok, który pozwala generować dodatkowy kod z kilku przydatnych adnotacji. Do użycia biblioteki konieczne jest dopisanie do **dependencies** naszego **mavenowego** projektu nowych zależności. W pliku **pom.xml** powinien się znaleźć następujący wpis:

```
<dependencies>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.16.20</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Do niego konieczne jest również instalowanie dodatkowego pluginu i uruchomienia (w IntelliJ) przetwarzania adnotacji. Przejdź do ustawień przez:

File ->

Settings ->

Build, Execution, Deployment ->

Annotation Processors ->

Zaznacz pole '**Enable Annotation Processing**'

Adnotacje - przykład użycia adnotacji z Projektu Lombok

Następnie zainstaluj plugin:

File ->

Settings ->

Plugins ->

W pole wyszukiwania wpisz **‘Lombok’** ->

Kliknij **‘Search in repositories’** ->

Lombok Plugin ->

Install ->

Zresetuj IntelliJ

Teraz kiedy uruchomisz kod użyjemy adnotacji z nowej biblioteki.

Adnotacje - przykład użycia adnotacji z Projektu Lombok

Do użycia tej biblioteki posłużę się klasą **Student**:

```
public class Student {
    private String name = "dżordż";
    private int age;
}
```

Zwróć uwagę na brak **getterów** i **setterów** dla obu pól klasy. Ponadto, nie mamy również konstruktora z parametrami, co czyni oba pola klasy kompletnie niedostępnymi na zewnątrz tej klasy. Jeśli instalacja pluginu Lombok powiodła się poprawnie, możliwe stanie się dodanie adnotacji np. **@Data**, **@Getter**, **@Setter** .

```
@Data
public class Student {
    private String name = "dżordż";
    private int age;
}
```

@Setter – powoduje wygenerowanie Setterów do pola/pól.

@Getter – powoduje wygenerowanie Getterów do pola/pól.

@ToString– powoduje wygenerowanie metody toString do klasy.

@EqualsAndHashCode – generuje metodę Equals oraz HashCode dla klasy.

@RequiredArgsConstructor – generuje metodę konstruktora z parametrami które muszą zawierać.

@Data - powoduje wygenerowanie metod **@Getter**, **@ToString**, **@EqualsAndHashCode**, oraz **@Setter** na wszystkich polach które nie są **final**, oraz **@RequiredArgsConstructor**.

Adnotacje - przykład użycia adnotacji z Projektu Lombok

Dzięki dodaniu **@Data** do deklaracji klasy możliwe jest użycie metod: