7

czwartek, 10 kwietnia 2025 11:24

nieznane atrybuty powinny być ignorowane i nie powodować błędu,
puste atrybuty powinny być pomijane przy generowaniu JSON,
atrybuty powinny być generowane zawsze w tej samej kolejności tj. ProductID, ProductName, ProductPrice, DateOfProduction, DateOfExpiry,

```
@JsonInclude(JsonInclude.Include.NON_NULL)  31 usages
@JsonPropertyOrder({"ProductID", "ProductName", "ProductPrice", "DateOfProduction", "DateOfExpiry"})
@JsonIgnoreProperties(ignoreUnknown = true)
```

pola klasy efs.task.reflection.json.ProductDTO powinny być mapowane do następujących elementów JSON:
ProductDTO.id naProductID w formacie JSON,
ProductDTO.name naProductName w formacie JSON,
ProductDTO.price naProductPrice w formacie JSON,
ProductDTO.expiryDate naDateOfExpiry w formacie JSON w następującym formacie yyyy-MM-dd,
ProductDTO.productionDate naDateOfProduction w formacie JSON w następującym formacie yyyy-MM-dd@HH:mm:ss,

```
@JsonProperty("ProductID")  3 usages
private Long id;
@JsonProperty("ProductName")  3 usages

private String name;
@JsonProperty("ProductPrice")  3 usages
private BigDecimal price;
@JsonProperty("DateOfExpiry")  3 usages
@JsonFormat(pattern="yyyy-MM-dd")
private Date expiryDate;
@JsonProperty("DateOfProduction")  3 usages
@JsonFormat(pattern = "yyyy-MM-dd@HH:mm:ss")
private Date productionDate;
```

Utwórz w oddzielnych plikach w pakiecie efs.task.reflection.annotations,

Publiczną adnotację czasu wykonania o nazwie NotNull którą będzie można zastosować na polach klasy oraz parametrach metod. Adnotacja powinna zwierać opcjonalny parametr o nazwie value z wartością domyślną n/a.
Publiczną adnotację czasu kompilacji o nazwie BuilderProperty którą będzie można zastosować na metodach i konstruktorach. Adnotacja powinna zwierać wymagany parametr o nazwie name typu String.
Oznacz adnotacją @NotNull pole name oraz description w klasie Villager,
Oznacz adnotacją @NotNull parametry name oraz description publicznego konstruktora klasy Villager,
Oznacz adnotacją @BuilderProperty z parametrem name = init publiczny konstruktor klasy Villager,
Oznacz adnotacją @BuilderProperty z parametrem name = health metodę setHealth w klasie Villager,

```java
@NotNull  2 usages
  private final String description;

  private Integer age;  2 usages

  int health;  2 usages
  private Villager() { this(HIDDEN_VILLAGER_NAME, HIDDEN_VILLAGER_DESC); }

  private Villager(Integer age , String name) {  no usages
    this(name, HIDDEN_VILLAGER_DESC);
    this.age = age;
  }

  private Villager(String name, Integer age) {  no usages
    this(name, HIDDEN_VILLAGER_DESC);
    this.age = age;
  }

@BuilderProperty(name="init")
public Villager(@NotNull String name, @NotNull String description) {
    this.name = name;
    this.description = description;
}

@BuilderProperty(name="health")  no usages
```

Zaimplementować zawartość metod klasy efs.task.reflection.ClassInspector zgodnie z komentarzami zamieszczonymi w pliku

```java
   */
  public static Collection<String> getAnnotatedFields(final Class<?> type,  1 usage
      final Class<? extends Annotation> annotation) {
      Collection<String> anno = new HashSet<>();
    //TODO usuń zawartość tej metody i umieść tutaj swoje rozwiązanie
      for (Field field : type.getDeclaredFields()) {
          if (field.isAnnotationPresent(annotation)) {
              anno.add(field.getName());
          }
      }
    return anno;
  }
```

Set żeby nie było duplikatow

type.getDeclaredFields(): Ta metoda zwraca wszystkie pola zadeklarowane w klasie type,
sprawdzmay sobie kazda i jeżeli pole jest adnotowane to dodajemy ti kolekcji

```java
    */
    public static Collection<String> getAllDeclaredMethods(final Class<?> type) {  1 usage
        //TODO usuń zawartość tej metody i umieść tutaj swoje rozwiązanie
        Collection<String> anno = new HashSet<>();
        for (Method method : type.getDeclaredMethods()) {
            anno.add(method.getName());
        }

        for(Class<?> iface : type.getInterfaces()) {
            for(Method method : iface.getDeclaredMethods()) {
                anno.add(method.getName());
            }
        }
        return anno;
    }
```

Znowu set
Dodajemy sobue nazwy metod do kolekcji, a potem w druiej petli spradzmy wszystkie interfesy implementowane przez dana klase i tez dodajemy do kolekcji, czyli zwraca kolekcję wszystkich nazw metod, które zostały zadeklarowane w klasie type i we wszystkich interfejsach

```java
    public static <T> T createInstance(final Class<T> type, final Object... args) throws Exception {  4 usages
        Constructor<?>[] constructors = type.getDeclaredConstructors();

        for (Constructor<?> constructor : constructors) {
            Class<?>[] parameterTypes = constructor.getParameterTypes();

            if (parameterTypes.length != args.length)
                continue;

            boolean match = true;
            for (int i = 0; i < parameterTypes.length; i++) {
                if (args[i] != null && !parameterTypes[i].isAssignableFrom(args[i].getClass())) {
                    match = false;
                    break;
                }
            }

            if (match) {
                constructor.setAccessible(true);
                return (T) constructor.newInstance(args);
            }
        }

        throw new NoSuchMethodException("Nie znaleziono pasującego konstruktora dla klasy: " + type.getName());
    }
```

Najpierw pobieramy wszystkie konstruktory
Constructor<?>[] constructors = type.getDeclaredConstructors();
Iterujemy przez nie w petli
Sprawdzamy czy liczba parametrow się zadza
if (parameterTypes.length != args.length)
  continue;, jeżeli nie to pomijamy i przecodzimy do nastepnwo

boolean match = true;
for (int i = 0; i < parameterTypes.length; i++) {
  if (args[i] != null && !parameterTypes[i].isAssignableFrom(args[i].getClass())) {
    match = false;

```
            break;
        }
    }
```
potem tutaj spraddzam czy parametry sa dopasowane, jeżeli mamy match to uzywamy
constructor.setAccessible(true) żeby umozliwic dostep do prywatncy konstruktorow
constructor.newInstance(args) tworzy noowa instancje
I na kooncu mamy oblsue wyjatkow

Podsumowujac
Metoda createInstance próbuje znaleźć odpowiedni konstruktor w klasie type, który pasuje do przekazanych argumentów (args). Jeśli taki konstruktor zostanie znaleziony, utworzy instancję obiektu i zwróci ją. Jeśli żaden konstruktor nie pasuje, rzuci wyjątek NoSuchMethodException.

Testy: