



Project Lombok
Cause We Hate Boilerplate!

Agenda

- O mnie
- Boilerplate – definicja
- Boilerplate w Javie
- Project Lombok
- Przegląd możliwości
- Jak działa Lombok ?
- Kontrowersje
- Pytania i odpowiedzi



O mnie

Łukasz Żmudziński

- Java Developer (4+ lata)
- JavaScript Developer (2+ lata)
- Entuzjasta lekkich metodyk wytwarzania oprogramowania
- Stawiam na *jakość* tworzonego oprogramowania
- W wolnym czasie podróżuje



Boilerplate

Boilerplate to kod:

- powtarzający się w wielu miejscach, który przeważnie wnosi niewiele do logiki Twojego systemu,
- który zaciemnia logikę sposób działania systemu,
- który trzeba napisać by zrealizować daną funkcjonalność,
- który można generować używając narzędzi,
- który będziesz musiał przeczytać za pół roku i nie będziesz do końca pewny czy siedzi w nim chochlik,
- za który nie chciałby płacić Twój klient (ale zapłaci 😊),
- specyficzny dla technologii.



Boilerplate w skrócie

Kod, który trzeba:

- » **PISAĆ**
- » **CZYTAĆ**
- » **TESTOWAĆ**
- » **UTRZYMYWAĆ**



Boilerplate w skrócie

Zwrot wartości dla klienta:

0.01 %



Boilerplate w Javie

?



Boilerplate w Javie

DEMO



Lombok – adnotacje

- @Getter / @Setter
- @ToString
- @EqualsAndHashCode
- @NoArgsConstructor, @RequiredConstructor, @AllArgsConstructor
- @Data
- @NonNull
- @Cleanup
- @Synchronized
- @SneakyThrows
- ...





@Getter / @Setter

```
public class Person {  
  
    @Getter @Setter private String firstName;  
    @Getter @Setter (AccessLevel.PROTECTED) private boolean employed = true;  
  
}
```

@Getter / @Setter

```
public class Person {  
    private String firstName;  
    private boolean employed = true;  
  
    public String getFirstName() {  
        return this.firstName;  
    }  
    public void setFirstName(final String firstName) {  
        this.firstName = firstName;  
    }  
)  
    public boolean isEmployed() {  
        return this.employed;  
    }  
    protected void setEmployed(final boolean employed) {  
        this.employed = employed;  
    }  
}
```



@ToString

```
@ToString
```

```
public class Vehicle {  
    private String name;  
}
```

```
@ToString(callSuper=true, exclude="producer")
```


```
public class Car extends Vehicle {  
    private String producer;  
}
```

@ToString

```
public class Vehicle {  
    private String name;  
    @java.lang.Override  
    public java.lang.String toString() {  
        return "Vehicle(name=" + this.name + ")";  
    }  
}  
  
public class Car extends Vehicle {  
    private String producer;  
    @java.lang.Override  
    public java.lang.String toString() {  
        return "Car(super=" + super.toString() + ")";  
    }  
}
```


@ToString

```
public class Vehicle {  
    private String name;  
    @java.lang.Override  
    public java.lang.String toString() {  
        return super.toString() + " " + name + ")";  
    }  
}  
  
public class Car {  
    private String name;  
    @java.lang.Override  
    public java.lang.String toString() {  
        return "Car(super=" + super.toString() + ")";  
    }  
}
```



The tooltip displays the following options for the @ToString annotation:

- callSuper : boolean - ToString
- doNotUseGetters : boolean - ToString
- exclude : String[] - ToString
- includeFieldNames : boolean - ToString
- of : String[] - ToString

Press 'Ctrl+Space' to show Template Proposals

@NonNull



```
public class Person {  
  
    @NonNull @Getter @Setter  
    private String firstName;  
  
}
```

@NonNull

```
public class Person {  
    @NonNull  
    private String firstName;  
  
    @NonNull  
    @java.lang.SuppressWarnings("all")  
    public String getFirstName() {  
        return this.firstName;  
    }  
  
    @java.lang.SuppressWarnings("all")  
    public void setFirstName(@NonNull final String firstName) {  
        if (firstName == null) throw new java.lang.NullPointerException("firstName");  
        this.firstName = firstName;  
    }  
}
```

@EqualsAndHashCode



```
@EqualsAndHashCode
public class Person {

    private String firstName;
    private @NotNull String lastName;

}
```

@EqualsAndHashCode

```
public class Person {
    private String firstName;
    @NonNull
    private String lastName;

    @java.lang.Override
    public boolean equals(final java.lang.Object o) {
        if (o == this) return true;
        if (o == null) return false;
        if (o.getClass() != this.getClass()) return false;
        final Person other = (Person)o;
        if (this.firstName == null ? other.firstName != null :
            !this.firstName.equals(other.firstName)) return false;
        if (this.lastName == null ? other.lastName != null :
            !this.lastName.equals(other.lastName)) return false;
        return true;
    }

    @java.lang.Override
    public int hashCode() {
        final int PRIME = 31;
        int result = 1;
        result = result * PRIME + (this.firstName == null ? 0 : this.firstName.hashCode());
        result = result * PRIME + (this.lastName == null ? 0 : this.lastName.hashCode());
        return result;
    }
}
```




@Data

@EqualsAndHashCode + @Getter + @Setter + @ToString

@Data



```
@Data(staticConstructor="newInstance")
public class Person {

    private String firstName;
    private @NonNull String lastName;
}
```



@Data

```
public class Person {  
    private String firstName;  
    @NonNull  
    private String lastName;  
  
    private Person(@NonNull final String lastName) {  
        if (lastName == null) throw new  
            java.lang.NullPointerException("lastName");  
        this.lastName = lastName;  
    }  
  
    public static Person newInstance(@NonNull final String lastName) {  
        return new Person(lastName);  
    }  
}
```



@Synchronized

```
public class Hashtable {  
  
    /**  
     * The total number of entries in the hash table.  
     */  
    private transient int count;  
  
    public synchronized boolean isEmpty() {  
        return count == 0;  
    }  
  
    @Synchronized  
    public boolean isEmptyPlusPlus() {  
        return count == 0;  
    }  
  
}
```

@Synchronized

```
public class Hashtable {  
    /**  
     * The total number of entries in the hash table.  
     */  
    private transient int count;  
  
    public synchronized boolean isEmpty() {  
        return count == 0;  
    }  
  
    public boolean isEmptyPlusPlus() {  
        synchronized (this.$lock) {  
            return count == 0;  
        }  
    }  
    @java.lang.SuppressWarnings("all")  
    private final java.lang.Object $lock = new java.lang.Object[0];  
}
```




@Cleanup

```
public void testCleanUp() throws IOException {  
    @Cleanup  
    ByteArrayOutputStream baos = new ByteArrayOutputStream();  
    baos.write(new byte[] { 'Y', 'e', 's' });  
    System.out.println(baos.toString());  
}
```

@Cleanup

```
public void testCleanUp() throws IOException {  
    ByteArrayOutputStream baos = new ByteArrayOutputStream();  
    try {  
        baos.write(new byte[]{'Y', 'e', 's'});  
        System.out.println(baos.toString());  
    } finally {  
        baos.close();  
    }  
}
```

@SneakyThrows

```
public void testCleanUp() /*throws IOException*/{  
    @Cleanup  
    ByteArrayOutputStream baos = new ByteArrayOutputStream();  
    baos.write(new byte[] { 'Y', 'e', 's' });  
    System.out.println(baos.toString());  
}
```

 **Unhandled exception type IOException**

2 quick fixes available:

 [Add throws declaration](#)

 [Surround with try/catch](#)

Press 'F2' for focus

@SneakyThrows

```
public void testCleanUp() {  
    try {  
        @Cleanup  
        ByteArrayOutputStream baos = new ByteArrayOutputStream();  
        baos.write(new byte[] { 'Y', 'e', 's' });  
        System.out.println(baos.toString());  
    } catch (IOException e) {  
        // this should never happen !  
    }  
}
```



@SneakyThrows

@SneakyThrows

```
public void testCleanUp() {  
    @Cleanup  
    ByteArrayOutputStream baos = new ByteArrayOutputStream();  
    baos.write(new byte[] { 'Y', 'e', 's' });  
    System.out.println(baos.toString());  
}
```



@SneakyThrows

```
public void testCleanUp() {  
    try {  
        ByteArrayOutputStream baos = new ByteArrayOutputStream();  
        try {  
            baos.write(new byte[]{'Y', 'e', 's'});  
            System.out.println(baos.toString());  
        } finally {  
            baos.close();  
        }  
    } catch (final java.lang.Throwable $ex) {  
        throw lombok.Lombok.sneakyThrow($ex);  
    }  
}
```



@SneakyThrows

```
@SneakyThrows (FileNotFoundException.class)
Public void testCleanUp() {
    @Cleanup
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    baos.write(new byte[] { 'Y', 'e', 's' });
    System.out.println(baos.toString());
}
```

 Unhandled exception type IOException

2 quick fixes available:



[Add throws declaration](#)

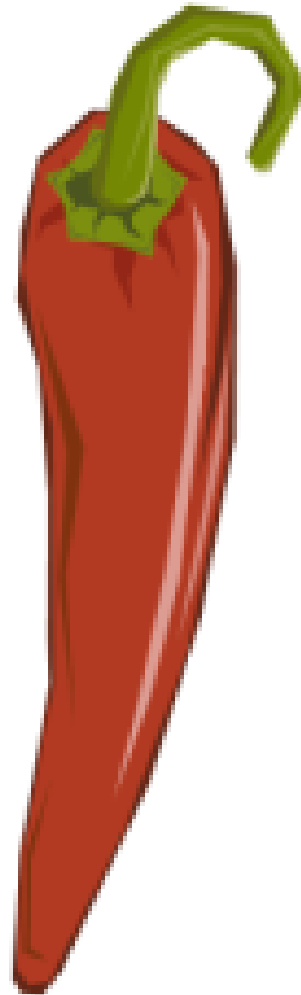


[Surround with try/catch](#)

Press 'F2' for focus

Jak działa Lombok ?

Wersja dla Managerów



Wersja dla Hakerów

Jak działa Lombok ?



- Używa* API procesorów adnotacji (JSR 269)
- Modyfikuje tylko jedną rzecz w kompilatorze Java (javac) drzewo AST (czysty kod źródłowy w postaci drzewa) dodając rzeczy od siebie zanim javac zacznie właściwą kompilację.
- W Eclipse czy Netbeans uruchamiany jest jako Java Agent (dodając do kodu parsera odwołanie do AST ze zmianami Lomboka)
- Zatem wszystkie funkcje, które działają na AST (kompilacja, sprawdzanie składni, uzupełnianie składni) dostają zmodyfikowane AST

Jak działa Lombok ?



Reinier Zwitterloot:

...,I'm the author of project lombok.

It's a **total hack**. Using **non-public API**. Presumptuous casting (knowing that an annotation processor running in javac will get an instance of JavacAnnotationProcessor, which is the internal implementation of AnnotationProcessor (an interface), which so happens to have a couple of extra methods that are used to get at the live AST).

On eclipse, it's arguably worse (and yet more robust) - a **java agent** is used to **inject code into the eclipse grammar and parser class**, which is of course entirely non-public API and totally off limits.

If you could do what lombok does with standard API, I would have done it that way, but you can't. Still, for what its worth, I developed the **eclipse plugin** for eclipse v3.5 running on **java 1.6**, and without making any changes it worked on eclipse v3.4 running on java 1.5 as well, so it's not completely fragile.”...

Project Lombok - inne funkcje

- @Logger (Morbok),
- @Fluent,
- Delombok,
- Przeciążanie Operatorów czyli $a+b*c \Rightarrow a.add(b.multiply(c))$

TODO

- @Builder
- @Delegate
- @Mixin

A za dwa tygodnie Devoxx 2010 ...



Lombok – wspierane narzędzia

- ✓ Java 1.6
- ✓ Ant
- ✓ Maven
- ✓ Eclipse IDE
- ✓ NetBeans IDE
- ✓ JDeveloper IDE

✗ IntelliJ IDEA

✗ Java 1.5



Kontrowersje

- Adnotacje powinny przenosić „META” informacje. Nie powinno być sytuacji, w której usunięcie adnotacji sprawi, że kod przestanie się kompilować.



Lombok - plany

Obecna wersja: **0.9.3**

Przyszłość: **1.0.0**

- Wsparcie dla 3 głównych IDEs: Eclipse, NetBeans IDE, IntelliJ IDEA.
- Wsparcie dla wszystkich automatycznych refaktoryzacji
- Wtyczki dla 3 IDE oferujące np. zamianę istniejącego gettera na @Getter (a w przyszłości jeszcze preview takiej transformacji)
- Stabilne API do pisania swoich transformacji



Pytania ?



Zasoby

- <http://projectlombok.org/>
- <https://github.com/rzwitserloot/lombok>
- <https://github.com/rzwitserloot/lombok.ast>
- <https://github.com/rzwitserloot/lombok.patcher>
- <https://github.com/lukasz-zmudzinski/JavaCamp5>





FIN

