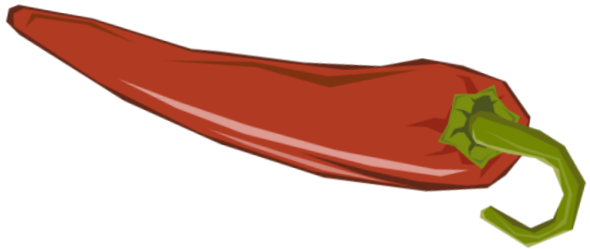
СБЕРБАНК ТЕХНОЛОГИИ

Lombok

СБЕРБАНК ТЕХНОЛОГИИ

Boilerplate code - это такой раздел кода, который должен быть включен во многие места с небольшими или никакими изменениями. Программист должен написать много кода для выполнения минимальной задачи.



СБЕРБАНК ТЕХНОЛОГИИ

Project Lombok - это java-библиотека, которая автоматически подключается к вашему редактору (с помощью плагинов) и автоматически генерирует boilerplate (шаблонный) код (при компиляции), сокращая количество лишнего кода, которого и так не мало в Java.

СБЕРБАНК ТЕХНОЛОГИИ

@NonNull

- Аннотируем поле, параметр, метод или локальную переменную
- lombok генерирует null-check –

```
if (param == null) {
    throw new NullPointerException ("param");
}
```

@Getter and @Setter:

- Аннотируем ими любое поле, и ломбок сгенерирует валидные java beans методы.
- Также можно поместить аннотации на класс. В этом случае, как будто вы аннотируете все нестатические поля этого класса с аннотацией.
- Можем выставить уровень доступа для методов или исключить вовсе
- Можем проставить методам аннотации
- Автоматически к методам копируется javadoc

@GETTER AND @SETTER

СБЕРБАНК ТЕХНОЛОГИИ

@Getter & @Setter

Setter.AnyAnnotation[] Getter.AnyAnnotation[]	onMethod	Любые аннотации, перечисленные здесь, помещаются в сгенерированный метод. *Синтаксис этой функции зависит от версии JDK.* @Setter (onMethod=@__({@AnnotationsGoHere})) @Getter (onMethod=@__({@AnnotationsGoHere}))
AccessLevel	value	Если вы хотите, чтобы Setter/Getter был непубличным, вы можете указать здесь альтернативный уровень доступа. Default: lombok.AccessLevel.PUBLIC

@Setter

Модификатор и тип	Элемент	Описание
Setter.AnyAnnotation[]	onParam	Любые аннотации, перечисленные здесь, помещаются в параметр сгенерированного метода. *Синтаксис этой функции зависит от версии JDK.* @Setter(onParam=@__({@AnnotationsGoHere}))

@Getter

Модификатор и тип	Элемент	Описание
boolean	lazy	Более подробно на следующем слайде

@GETTER(LAZY=TRUE)

СБЕРБАНК ТЕХНОЛОГИИ

- Lombok может генерировать getter, который будет вычислять значение один раз, при первом вызове этого getter и кэшировать его с этого момента
- При этом будет обеспечен потокобезопасный доступ к полю
- Никогда не обращайтесь к полю напрямую, **всегда используйте геттер**, сгенерированный ломбоком!

```
public class GetterLazyExample {
    @Getter(lazy = true)
    private final double[] cached = expensive();

    private double[] expensive() {
        double[] result = new double[1000000];
        for (int i = 0; i < result.length; i++) {
            result[i] = Math.asin(i);
        }
        return result;
    }}
```

@GETTER(LAZY=TRUE)

СБЕРБАНК ТЕХНОЛОГИИ

```
public class GetterLazyExample {
    private final java.util.concurrent.AtomicReference<java.lang.Object> cached
= new java.util.concurrent.AtomicReference<java.lang.Object>();

    public double[] getCached() {
        java.lang.Object value = this.cached.get();
        if (value == null) {
            synchronized(this.cached) {
                value = this.cached.get();
                if (value == null) {
                    final double[] actualValue = expensive();
                    value = actualValue == null ? this.cached : actualValue;
                    this.cached.set(value);
                }
            }
        }
        return (double[])(value == this.cached ? null : value);
    }
    private double[] expensive() { … }
}
```

@ToString

Любое определение класса может быть аннотировано с помощью @ToString, чтобы lombok генерировал реализацию метода toString (). По умолчанию оно будет печатать имя вашего класса вместе с каждым полем в порядке, разделенным запятыми.

- Можно добавить (или исключить) имена полей
- Исключить поле из вывода
- Добавлять вывод родительского класса.
- **Бойтесь рекурсивных ссылок!**

@TOSTRING

СБЕРБАНК ТЕХНОЛОГИИ

Модификатор и тип	Элемент	Описание
boolean	callSuper	Включите результат реализации суперкласса toString в выходе. default: false
boolean	DoNotUseGetters	Обычно, если доступны геттеры, они вызываются. default: false
java.lang.String[]	exclude	Любые поля, перечисленные здесь, не будут напечатаны в сгенерированной реализации toString. Взаимное исключение с of().
boolean	includeFieldNames	При печати используется имя каждого поля default: true
java.lang.String[]	of	Если присутствует, явным образом перечисляет поля, которые должны быть напечатаны. Взаимное исключение с exclude().

@EqualsAndHashCode

- Этой аннотацией можно пометить любой класс, чтобы позволить ломбоку генерировать реализации методов **equals (Object other) и hashCode ().** *По умолчанию он будет использовать все нестатические, непереходные поля, но можно настроить участников.*
- Аннотирование класса, расширяющего другой сложнее. Обычно автоматическое создание метода equals и hashCode для таких классов является **плохой идеей**.
- **callSuper=true**, когда вы ничего не расширяете (вы расширяете java.lang.Object) -> **ошибка компиляции**
- **callSuper=false** при расширении другого класса -> **генерация предупреждения**

@EQUALSANDHASHCODE

СБЕРБАНК ТЕХНОЛОГИИ

Модификатор и тип	Элемент	Описание
boolean	callSuper	Исп. реализацию суперкласса equals и hashCode перед вычислением для полей этого класса. default: false
boolean	DoNotUseGetters	Не использовать геттеры. default: false
java.lang.String[]	exclude	Любые поля, перечисленные здесь, не будут учитываться в сгенерированных реализациях equals и hashCode. Взаимное исключение с of().
java.lang.String[]	of	Если присутствует, явным образом перечисляет поля, которые должны использоваться для идентификации. Взаимное исключение с exclude().
EqualsAndHashCode .AnyAnnotation[]	onParam	Любые аннотации, перечисленные здесь, помещаются в сгенерированный параметр equals и canEqual. Это полезно, например, при добавлении аннотации Nullable. Синтаксис этой функции зависит от версии JDK @EqualsAndHashCode(onParam=@__({@AnnotationsGoHere}))

С ЛОМБОКОМ

СБЕРБАНК ТЕХНОЛОГИИ

```
@EqualsAndHashCode(exclude = {"id", "shape"})
public class EqualsAndHashCodeExample {
    private transient int transientVar = 10;
    private String name;
    private double score;
    private Shape shape = new Square(5, 10);
    private String[] tags;
    private int id;

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

КОМПИЛИРУЕТСЯ В

СБЕРБАНК ТЕХНОЛОГИИ

```
public class EqualsAndHashCodeExample {
    private transient int transientVar = 10;
    private String name;
    private double score;
    private Shape shape = new Square(5, 10);
    private String[] tags;
    private int id;
    public String getName() {
        return this.name;
    }
    @Override
    public boolean equals(Object o) {
        if (o == this) return true;
        if (!(o instanceof EqualsAndHashCodeExample)) return false;
        EqualsAndHashCodeExample other = (EqualsAndHashCodeExample) o;
        if (!other.canEqual((Object) this)) return false;
        if (this.getName() == null ? other.getName() != null : !this.getName().equals(other.getName()))
            return false;
        if (Double.compare(this.score, other.score) != 0) return false;
        if (!Arrays.deepEquals(this.tags, other.tags)) return false;
        return true;
    }
    @Override
    public int hashCode() {
        final int PRIME = 59;
        int result = 1;
        final long temp1 = Double.doubleToLongBits(this.score);
        result = (result * PRIME) + (this.name == null ? 43 : this.name.hashCode());
        result = (result * PRIME) + (int) (temp1 ^ (temp1 >>> 32));
        result = (result * PRIME) + Arrays.deepHashCode(this.tags);
        return result;
    }
    protected boolean canEqual(Object other) {
        return other instanceof EqualsAndHashCodeExample;
    }
}
```

@NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor

- **@NoArgsConstructor** генерирует конструктор без параметров. Если это возможно.
- **@NoArgsConstructor (force = true),** тогда все конечные поля инициализируются с помощью 0 / false / null.
- **@RequiredArgsConstructor** генерирует конструктор с 1 параметром для каждого поля, для которого требуется специальная обработка.
- **@AllArgsConstructor** создает конструктор с 1 параметром для каждого поля вашего класса.
- @NonNull на поле приводит к реализации проверки на поле

КОНСТРУКТОРЫ

СБЕРБАНК ТЕХНОЛОГИИ

@NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor

Модификатор и тип	Элемент	Описание
AccessLevel	access	Устанавливает уровень доступа конструктора. Default: lombok.AccessLevel.PUBLIC
Boolean (только для NoArgsConstructor)	force	Если true, инициализирует все конечные поля 0 / null / false. В противном случае возникает ошибка компиляции. Default: false
AllArgsConstructor .AnyAnnotation[]	onConstructor	Любые аннотации, перечисленные здесь, помещаются в сгенерированный конструктор. Синтаксис этой функции зависит от версии JDK @NoArgsConstructor(onConstructor=@__({@AnnotationsGoHere}))
java.lang.String	staticName	Сгенерированный конструктор будет приватным, а дополнительный статический «конструктор» создается с тем же списком аргументов, который обертывает реальный конструктор

С ЛОМБОКОМ

СБЕРБАНК ТЕХНОЛОГИИ

```
@RequiredArgsConstructor(staticName = "of")
@AllArgsConstructor(access = AccessLevel.PROTECTED)
public class ConstructorExample<T> {
    private int x, y;
    @NonNull
    private T description;

    @NoArgsConstructor
    public static class NoArgsExample {
        @NonNull
        private String field;
    }
}
```

КОМПИЛИРУЕТСЯ В

СБЕРБАНК ТЕХНОЛОГИИ

```
public class ConstructorExample<T> {
    private int x, y;
    @NonNull
    private T description;

    private ConstructorExample(T description) {
        if (description == null) throw new NullPointerException("description");
        this.description = description;
    }
    @java.beans.ConstructorProperties({"x", "y", "description"})
    protected ConstructorExample(int x, int y, T description) {
        if (description == null) throw new NullPointerException("description");
        this.x = x;
        this.y = y;
        this.description = description;
    }
    public static <T> ConstructorExample<T> of(T description) {
        return new ConstructorExample<T>(description);
    }
    public static class NoArgsExample {
        @NonNull
        private String field;

        public NoArgsExample() {
        }
    }
}
```

ШАБЛОН СТРОИТЕЛЬ ОДНОЙ АННОТАЦИЕЙ

СБЕРБАНК ТЕХНОЛОГИИ

@Builder

- Легкая реализация шаблона «Строитель».
- Помечаем класс, метод или конструктор
- Может генерировать так называемые «сингулярные» методы для параметров/полей коллекции. Поле помечается аннотацией @Singular

@BUILDER

СБЕРБАНК ТЕХНОЛОГИИ

Модификатор и тип	Элемент	Описание
String	builderClassName	Имя класса билдера. По умолчанию для @Builder типов и конструкторов: (TypeName) Builder.
String	builderMethodName	Имя метода, создающего экземпляр нового экземпляра. Default: "builder"
String	buildMethodName	Имя метода в классе строителя, который создает экземпляр @Builder-аннотированного класса. Default: "build"
boolean	toBuilder	сгенерирует метод экземпляра, чтобы получить builder, который инициализируется значениями этого экземпляра. Только в том случае, если **@Builder используется в конструкторе**, самом типе или статическом методе, который возвращает экземпляр объявленного типа. Default: false

С ЛОМБОКОМ

СБЕРБАНК ТЕХНОЛОГИИ

```
import java.util.Set;

import lombok.Singular;

@Builderpublic
class BuilderExample {
    private String name;
    private int age;
    @Singular
    private Set<String> occupations;
}
```

КОМПИЛИРУЕТСЯ В

СБЕРБАНК ТЕХНОЛОГИИ

```
import java.util.Set;

public class BuilderExample {
    private String name;
    private int age;
    private Set<String> occupations;

    BuilderExample(String name, int age, Set<String> occupations) {
        this.name = name;
        this.age = age;
        this.occupations = occupations;
    }

    public static BuilderExampleBuilder builder() {
        return new BuilderExampleBuilder();
    }

    public static class BuilderExampleBuilder {
        private String name;
        private int age;
        private java.util.ArrayList<String> occupations;

        BuilderExampleBuilder() {
        }

        public BuilderExampleBuilder name(String name) {
            this.name = name;
            return this;
        }

        public BuilderExampleBuilder age(int age) {
            this.age = age;
            return this;
        }

public BuilderExampleBuilder occupation(String occupation) {
            if (this.occupations == null) {
                this.occupations = new java.util.ArrayList<String>();
            }

            this.occupations.add(occupation);
            return this;
        }

        public BuilderExampleBuilder occupations(Collection<? extends String> occupations) {
            if (this.occupations == null) {
                this.occupations = new java.util.ArrayList<String>();
            }

            this.occupations.addAll(occupations);
            return this;
        }

        public BuilderExampleBuilder clearOccupations() {
            if (this.occupations != null) {
                this.occupations.clear();
            }

            return this;
        }

        public BuilderExample build() {
            // complicated switch statement to produce a compact properly sized immutable set
omitted.
            // go to https://projectlombok.org/features/Singular-snippet.html to see it.
            Set<String> occupations = ...;
            return new BuilderExample(name, age, occupations);
        }

        @java.lang.Override
        public String toString() {
            return "BuilderExample.BuilderExampleBuilder(name = " + this.name + ", age = " +
this.age + ", occupations = " + this.occupations + ")";
        }
    }
}
```

СБЕРБАНК ТЕХНОЛОГИИ

```
public String example() {
    val example = new ArrayList<String>();
    example.add("Hello, World!");
    var foo = example.get(0);
    log.info(foo);
    foo = example.get(1);
    return foo.toLowerCase();
}
```

ВЫ ПРОСИТЕ VAL'Ы? ИХ ЕСТЬ У МЕНЯ!

СБЕРБАНК ТЕХНОЛОГИИ

val

Вы можете использовать **val** как тип объявления финальной локальной переменной вместо фактического написания этого типа (тип будет выведен из выражения инициализации). Эта функция работает только с локальными переменными и только с циклами foreach.

var **Experimental (1.16.12)**

То же что и val, только не финальная.

@Delegate **Experimental**

- Любой метод поля или без аргумента может быть аннотирован с помощью @Delegate, чтобы позволить lombok генерировать методы делегирования, которые перенаправляют вызов этому полю (или результат вызова этого метода).
- Ломбок делегирует все общедоступные методы типа поля (или возвращаемого типа метода), а также его супертипы.
- Все методы в Object, а также canEqual (Object other) не будут делегированы.
- Можно передать типы с объявленными методами для включения в делегирование или наоборот – исключения.

С ЛОМБОКОМ

СБЕРБАНК ТЕХНОЛОГИИ

```
public class ExcludesDelegateExample {
    @Delegate(excludes = Add.class)
    private final Collection<String> collection = new ArrayList<String>();
    long counter = 0L;

    public boolean add(String item) {
        counter++;
        return collection.add(item);
    }
    public boolean addAll(Collection<? extends String> col) {
        counter += col.size();
        return collection.addAll(col);
    }
    private interface Add {
        boolean add(String x);

        boolean addAll(Collection<? extends String> x);
    }
}
```

КОМПИЛИРУЕТСЯ В

СБЕРБАНК ТЕХНОЛОГИИ

```
class ExcludesDelegateExample {
    private final Collection<String> collection = new ArrayList<String>();
    long counter = 0L;

    public boolean add(String item) {
        counter++;
        return collection.add(item);
    }

    public boolean addAll(Collection<? extends String> col) {
        counter += col.size();
        return collection.addAll(col);
    }

    @java.lang.SuppressWarnings("all")
    public int size() {
        return this.collection.size();
    }

    @java.lang.SuppressWarnings("all")
    public boolean isEmpty() {
        return this.collection.isEmpty();
    }
    …
```

СБЕРБАНК ТЕХНОЛОГИИ

@Cleanup

- Автоматическое закрытие ресурсов.
- Не так актуально с появлением try-with-resources

```
@Cleanup InputStream in = new FileInputStream(args[0]);
```

Превращается в:

```
InputStream in = new FileInputStream(args[0]);
try {
  ...
} finally {
  if (in != null) {  in.close();   }
}
```

@SneakyThrows

- Позволяет пробрасывать исключения из метода без объявления **throws**
- Наконец то можно избавиться от надоедливых, «невозможных» **checked-exception**

@Synchronized

- Оборачивает тело метода в synchronized-блок
- Инкапсулирует ваши локи

СБЕРБАНК ТЕХНОЛОГИИ

@Data

- **@ToString**
- **@EqualsAndHashCode**
- **@Getter** на все поля
- **@Setter** на все не final-поля
- **@RequiredArgsConstructor** (если уже не существует объявленного конструктора)

ЧТО ЭТО И ЗАЧЕМ?

СБЕРБАНК ТЕХНОЛОГИИ