

Checker Framework

Аннотации для статического анализа кода

Владимир Смирнов
Станислав Матковский

Это тулза для статик-анализа



- JDK уже разруливает кейсы типа `int count = "hello";`
- IDE подсказывает больше, например DEAD CODE;
- *FindBugs*, *CheckStyle* и прочие утилиты для предсказания ошибок в существующем коде;
- Для проверок на NPE, взаимодействия между слоями приложения или более глубоких concurrency errors требуется дополнительная метайнформация в коде.

Чем же хорош Checker Framework?

- Недостатки типизации Java компенсируются аннотациями;
- Он модульный, как конструктор, и расширяем сообществом;
- Есть встроенный компилятор;
- Кросс-тулзовый.



Checker совместим с

Вашим любимым
сборщиком:

- ▶ Maven
- ▶ Gradle
- ▶ Ant

Вашей незаменимой IDE:

IntelliJ IDEA ◀

Eclipse ◀

NetBeans ◀

Встроенные анализаторы

- [Nullness Checker](#) for null pointer errors
- [Initialization Checker](#) to ensure all fields are set in the constructor
- [Map Key Checker](#) to track which values are keys in a map
- [Interning Checker](#) for errors in equality testing and interning
- [Lock Checker](#) for concurrency and lock errors
- [Fake Enum Checker](#) to allow type-safe fake enum patterns and type aliases or typedefs
- [Tainting Checker](#) for trust and security errors
- [Regex Checker](#) to prevent use of syntactically invalid regular expressions
- [Format String Checker](#) to ensure that format strings have the right number and type of % directives
- [Internationalization Format String Checker](#) to ensure that i18n format strings have the right number and type of {} directives
- [Property File Checker](#) to ensure that valid keys are used for property files and resource bundles
- [Internationalization Checker](#) to ensure that code is properly internationalized
- [Signature String Checker](#) to ensure that the string representation of a type is properly used, for example in Class.forName
- [GUI Effect Checker](#) to ensure that non-GUI threads do not access the UI, which would crash the application
- [Units Checker](#) to ensure operations are performed on correct units of measurement
- [Signedness Checker](#) to ensure unsigned and signed values are not mixed
- [Constant Value Checker](#) to determine whether an expression's value can be known at compile time
- [Aliasing Checker](#) to identify whether expressions have aliases
- [Linear Checker](#) to control aliasing and prevent re-use
- [Subtyping Checker](#) for customized checking without writing any code

Простейший пример Nullness Checker

```
void showObjectSafe(@Nullable Object o) {  
    System.out.println(o.toString());  
}
```

```
void showObject(@Nullable Object o) {  
    showObjectUnsafe(o);  
}
```

```
void showObjectUnsafe(@NonNull Object o) {  
    if (o != null) {  
        System.out.println(o.toString());  
    }  
}
```

Description	Resource	Path	Location	Type
Errors (4 items)				
dereference of possibly-null reference s	BaseClass.java	/NullnessCheckerTest...	line 48	Checker Framework
incompatible types in argument.	BaseClass.java	/NullnessCheckerTest...	line 51	Checker Framework
redundant check; "s" is non-null	BaseClass.java	/NullnessCheckerTest...	line 54	Checker Framework
redundant check; "s" is non-null	BaseClass.java	/NullnessCheckerTest...	line 54	Checker Framework

Еще пример для Nullness Checker

```
public class BaseClass {  
  
    public @NonNull Object nnobj;  
  
    public BaseClass() { }  
  
    public void main() {  
        List<@Nullable String> l1 = new ArrayList<>();  
        l1.add("test");  
        l1.add(null);  
        for (String s : l1) showStringUnsafe(s);  
  
        List<@NonNull String> l2 = new ArrayList<>();  
        l2.add("test");  
        l2.add(null);  
        for (String s : l2) showStringUnsafe(s);  
    }  
  
    private void showStringUnsafe(@NonNull String s) {  
        System.out.println(s.toString());  
    }  
}
```

Я использую внешние библиотеки, там нет этих ваших аннотаций

- Фреймворк поддерживает механизм внешнего аннотирования библиотек;
- Уже есть набор мета-аннотаций наиболее популярных библиотек, он расширяется и в этом даже можно поучаствовать;
- Некоторые проекты распространяют уже аннотированные версии библиотек.

Subtyping Checker

```
void main() {
    Person p = new Person("a", "b");
    getStatistics(p.getId(), p.getJobId());
    getStatistics(p.getId(), p.getJobId(), "c");
}

void getStatistics(String personId, String jobId) {
    // TODO
}

void getStatistics(String jobId, String personId, Object o) {
    // TODO
}
```

```
public class Person {
    private String id;
    private String jobId;

    public Person(String id, String jobId) {
        this.id = id;
        this.jobId = jobId;
    }

    public String getId() {
        return id;
    }

    public String getJobId() {
        return jobId;
    }
}
```

Subtyping Checker

```
public class Person {
    private @PersonGuid String id;
    private @JobGuid String jobId;

    public Person(String id, String jobId) {
        super();
        // :: warning: (cast.unsafe)
        this.id = (@PersonGuid String) id;
        // :: warning: (cast.unsafe)
        this.jobId = (@JobGuid String)
jobId;
    }
    public @PersonGuid String getId() {
        return id;
    }
    public @JobGuid String getJobId() {
        return jobId;
    }
}
```

Subtyping Checker

```
void main() {
    Person p = new Person("a", "b");
    getStatistics(p.getId(), p.getJobId());
    getStatistics(p.getId(), p.getJobId(), "c"); // ЛОВИМ ОШИБКУ
}

void getStatistics(@PersonGuid String personId, @JobGuid String jobId) {
    // TODO
}

void getStatistics(@JobGuid String jobId, @PersonGuid String personId, String o) {
    // TODO
}
```

Альтернативное решение без чекера

```
public class Person {
    private PersonGuid id;
    private JobGuid jobId;

    public Person(String id, String jobId) {
        this.id = new PersonGuid(id);
        this.jobId = new JobGuid(jobId);
    }

    public PersonGuid getId() {
        return id;
    }

    public JobGuid getJobId() {
        return jobId;
    }
}
```

```
public class PersonGuid extends ValueHolder<String> {

    public PersonGuid(String value) {
        super(value);
    }
}
```

```
public class JobGuid extends ValueHolder<String> {

    public JobGuid(String value) {
        super(value);
    }
}
```

Альтернативное решение без чекера

```
void main() {  
    Person p = new Person("a", "b");  
    getStatistics(p.getId(), p.getJobId());  
    getStatistics(p.getId(), p.getJobId(), "c"); // ЛОВИМ ОШИБКУ  
}  
  
void getStatistics(PersonGuid personId, JobGuid jobId) {  
    // TODO  
}  
  
void getStatistics(JobGuid jobId, PersonGuid personId, String o) {  
    // TODO  
}
```

Fake Enum Checker

```
@SuppressWarnings("assignment.type.incompatible")
public class AuthChoice {
    @Fenum("AuthChoice1")
    public static final String AUTH_CHOICE_CORRECT = "CORRECT";
    @Fenum("AuthChoice1")
    public static final String AUTH_CHOICE_INCORRECT = "INCORRECT";

    @Fenum("AuthChoice2")
    public static final String AUTH_CHOICE_CORRECT_2 = "CORRECT2";
    @Fenum("AuthChoice2")
    public static final String AUTH_CHOICE_INCORRECT_2 = "INCORRECT2";
}
```

```
private Result generateResult(Request authRequest, @Fenum("AuthChoice1") String authChoice) {
    // Ошибка компиляции!
    // LOGGER.trace("Генерируется результат {}", authChoice);
    switch (authChoice) {
        case AUTH_CHOICE_CORRECT:
            return new CorrectResult();
        case AUTH_CHOICE_INCORRECT:
            return new IncorrectResult();
    }
    return null;
}
```

Interning Checker – простейший пример

```
@Interned String foo = "foo";  
@Interned String bar = "bar";  
  
if (foo == bar) {  
    System.out.println("foo == bar");  
}
```

Interning Checker

```
public class ActionType {
    private static final Map<String, ActionType> actionsMap = new ConcurrentHashMap<>();
    private String action;

    public ActionType(String action) {
        this.action = action;
        actionsMap.put(this.action, this);
        System.out.println();
    }

    public String getAction() { return action; }

    @SuppressWarnings("interning")
    public static @Interned ActionType getValueSafe(String actionTypeName) {
        actionTypeName = actionTypeName.toUpperCase();
        ActionType actionType = actionsMap.get(actionTypeName);
        return (actionType == null) ? UNKNOWN : actionType;
    }
}

if (ActionType.getValueSafe("DELETE") == clientAction) {
    // Важная логика
}

if (ActionType.getValueSafe("DELETE") == new ActionType("DELETE")) {
    // Важная логика
}
```


Initialization Checker

```
public abstract class Controller {  
    private int id;  
    public Controller(int id) {  
        de  
        Li  
        de  
    }  
    protected  
}
```



ChildContr

```
[ERROR] /I  
[method.in  
[ERROR] fo  
[ERROR] re  
[ERROR] ->
```

oller {

ller

Lock Checker

```
private final ReentrantLock requestsMapLock = new ReentrantLock();
```

```
@GuardedBy("requestsMapLock")
```

```
protected Map<Request, Result> requests = new HashMap<>();
```

```
@MayReleaseLocks
```

```
public ResponseEntity<Map<AuthRequest, AuthResult>> startRequest(String termUrl, String md, String paReq) {
```

```
    AuthRequest authRequest = new AuthRequest(validateTermUrl(termUrl), md, paReq);
```

```
    Map<AuthRequest, AuthResult> succeededRequests;
```

```
    requestsMapLock.lock();
```

```
    try {
```

```
        succeededRequests = getSucceededRequests();
```

```
    } finally {
```

```
        requestsMapLock.unlock();
```

```
    }
```

```
    return ResponseEntity.ok(succeededRequests);
```

```
}
```

```
@Holding("requestsMapLock")
```

```
protected Map<AuthRequest, AuthResult> getSucceededRequests() {
```

```
    return requests.entrySet().stream()
```

```
        .filter(e -> e.getValue() != null)
```

```
        .collect(Collectors.toMap(e -> e.getKey(), e -> e.getValue()));
```

```
}
```

Tainting Checker

```
public class BusinessObject {  
  
    private String sensitiveData;  
  
    public Request(String sensitiveData) {  
        this.sensitiveData = sensitiveData;  
    }  
  
    public String getData() { return sensitiveData; }  
}
```

```
public void executeBusinessLogic(String data) {  
    data = validateData(data);  
    BusinessObject obj = new BusinessObject(data);  
}  
  
private String validateData(String data) {  
    // Логика валидации...  
    return pureData;  
}
```

Tainting Checker

```
public class BusinessObject {  
  
    private String sensitiveData;  
  
    public Request(@Untainted String sensitiveData) {  
        this.sensitiveData = sensitiveData;  
    }  
  
    public String getData() { return sensitiveData; }  
}
```

```
public void executeBusinessLogic(@Tainted String data) {  
    data = validateData(data);  
    BusinessObject obj = new BusinessObject(data);  
}  
  
@SuppressWarnings("tainting")  
private @Untainted String validateData(@Tainted String data) {  
    // Логика валидации...  
    return pureData;  
}
```

Regex Checker

```
Pattern.compile(".*"); // Ловит IDE, если что не так
```

```
Pattern.compile(or(parenthesize("a*"), parenthesize("b*"))); // IDE уже не справится
```

```
public @Regex String parenthesize(@Regex String regex) {  
    return "(" + regex + " ";  
}
```

```
public @Regex String or(@Regex String a, @Regex String b)  
{  
    return a + "|" + b;  
}
```

Internationalization & Format String Checkers

```
System.out.printf("Float %f, number %g", 3.1415, 42);
```

```
void printFloatAndInt(@Format({FLOAT, INT}) String fs) {  
    System.out.printf(fs, 3.1415, 42);  
}
```

```
printFloatAndInt("Float %f, Number %d"); // ОК  
printFloatAndInt("Float %f");           // Ошибка
```

```
// Нет второго аргумента  
MessageFormat.format("{0} {1}", 3.1415);  
// Аргумент нельзя отформатировать как «время»  
MessageFormat.format("{0, time}", "my string");
```

```
@I18nFormat({GENERAL, NUMBER}) String format;  
  
format = "{0} {1} {2}";
```

Поддерживает также работу с ResourceBundle



Map Key Checker

```
private void processKey(String extKey) {
    Map<String, @NonNull Object> map = new HashMap<>();
    Collection<@KeyFor("map") String> coll = new ArrayList<>();
    map.put(extKey, new Request());

    // Некоторое время спустя
    coll.add(extKey);

    // Еще позже
    for (String s : coll) {
        showObjectUnsafe(map.get(s));
    }
}

private void showObjectUnsafe(@NonNull Object o) {
    System.out.println(o.toString());
}
```

Units Checker

```
@m int meters = 5 * UnitsTools.m;
```

```
@s int secs = 2 * UnitsTools.s;
```

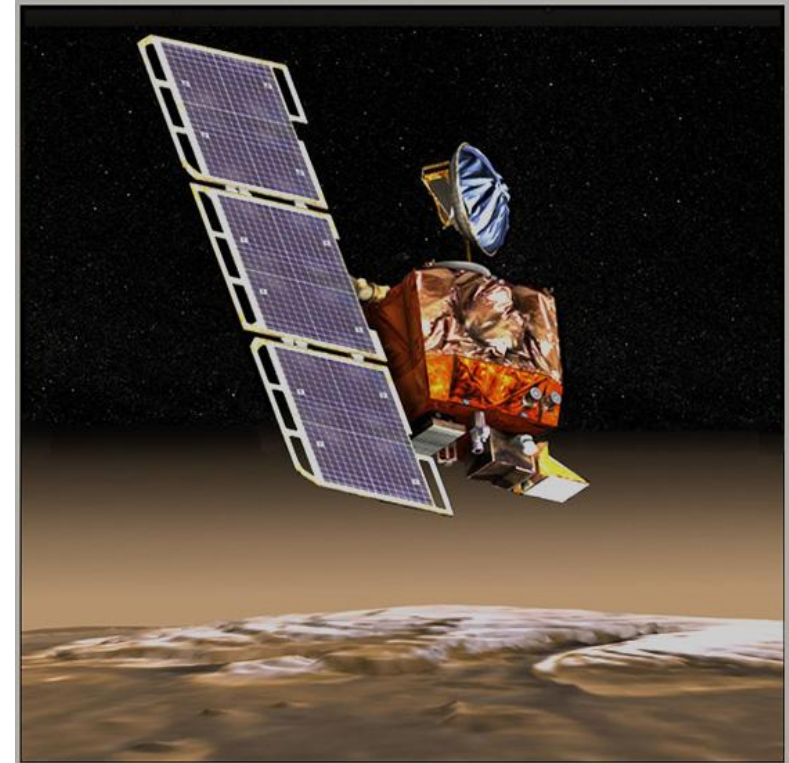
```
@mPERs double speed = meters / secs;
```

```
✘ @kmPERh double kmph = meters /  
secs;
```

```
✘ @kmPERh double kmph = speed * 3.6;
```

```
@kmPERh double kmph =  
UnitsTools.fromMeterPerSecondToKiloMeterPerHour(speed);
```

```
@kmPERh  
public static double fromMeterPerSecondToKiloMeterPerHour(@mPERs double mps) {  
    return mps * 3.6D;  
}
```



GUI Effect Checker

```
@SafeEffect
public void calledFromBackgroundThread()
{
    JLabel.setText("Foo");    // Ошибка
}

@UIEffect
private void calledFromUIThread() {
    heavyLoad();
}

@SafeEffect
private void heavyLoad() {
    // Some really heavy load...
}
```



Aliasing Checker

```
void testPlanet(@Unique Earth earth)
{
    Earth newPlanet = earth;
    // Какая-то логика...
    newPlanet.annihilate();
}
```

[ERROR] Reference annotated as @Unique is leaked.

Метод testPlanet никак не должен
МЕНЯТЬ earth!



Чекеры от сторонних разработчиков

Они есть,
да.

Я не использую Java 8

- Компилятор фреймворка обрабатывает и `List</* @NonNull */ String>`

У меня куча легаси без этих ваших аннотаций!

- Начинаем помаленьку, отдельные чекеры, отдельные ветви проекта;
- Думайте об аннотации как о части спецификации – обычно достаточно понимать и аннотировать сигнатуру метода, а не тело. В том же Javadoc хорошие ребята так же помечают, что например метод может вернуть null;
- Старайтесь сделать свой код лучше, как например использование параметризованных типов вместо raw types.

**Спасибо за
внимание!**