# Писать код быстрее, ошибаться реже

Чашников Николай
JetBrains


BUG

# Пример 1. Найди меня

```
public static URL getClassUrl(Class<?> aClass) {
    String path = aClass.getName().replaceAll(".", "/")
                        + ".class";
    return aClass.getClassLoader().getResource(path);
}
```

# Пример 2. Аннотации правят миром

```
@interface Anno {  String value(); }

@Anno("Hello, World!")
public class HelloAnno {
 public static void main(String[] args) {
   System.out.println(
      HelloAnno.class.getAnnotation(Anno.class).value());
 }
}
```

# Пример 3. Все писали это

```
public static String toString(List<String> list) {
    StringBuilder buf = new StringBuilder('(');
    for (int i = 0; i < list.size(); i++) {
        if (i > 0) buf.append(", ");
        buf.append(list.get(i));
    }
    return buf.append(')').toString();
}
```

# Пример 4. Хитрый прогресс

```
public void showCurrentProgress(JLabel label) {
    int percents = (int) Math.random()*100;
    label.setText(String.format("%d%% completed...",
                                percents));
}
```

# FindBugs

- анализирует байт-код
- встраивается в процесс компиляции
- плагины для Maven, IntelliJ IDEA, Eclipse
- 400+ типов ошибок

# PMD

- анализирует java код
- 270 проверок
- плагины для Maven, Eclipse
- расширяется при помощи шаблонов

# PMD

Есть интересные проверки:

**AvoidUsingVolatile**

Use of the keyword 'volatile' is generally used to fine tune a Java application, and therefore, requires a good expertise of the Java Memory Model. Moreover, its range of action is somewhat misknown. Therefore, the volatile keyword should not be used for maintenance purpose and portability.

# Error-prone compiler

- разрабатывается в Google
- подменяет javac
- анализирует AST дерево
- находит 24 типа ошибок (и 40 экспериментальных)
- легко расширять

```
public class NonRuntimeAnnotation extends BugChecker {
 public Description matchMethodInvocation(MethodInvocationTree tree,
                                          VisitorState state) {
   if (!methodSelect(
       instanceMethod(Matchers.<ExpressionTree>isSubtypeOf(
                                                  "java.lang.Class"), "getAnnotation"))
       .matches(tree, state)) {
     return Description.NO_MATCH;
   }
   MemberSelectTree memTree = (MemberSelectTree) tree.getArguments().get(0);
   TypeSymbol annotation = ASTHelpers.getSymbol(memTree.getExpression()).type.tsym;
   Retention retention = ASTHelpers.getAnnotation(annotation, Retention.class);
   if (retention != null && retention.value().equals(RUNTIME)) {
     return Description.NO_MATCH;
   }
   return describeMatch(tree, SuggestedFix.replace(tree, "null"));
 }
}
```

# Error-prone compiler

Из требований к новым проверкам:

**The bug pattern should have no false positives.**

In essentially no cases should the detected code actually be working as intended.

# Как проявляются ошибки

Ошибки компиляции

Исключения

Неправильное поведение

Что, если хочется не только находить ошибки по шаблонам, но и полностью исключить возможность появления некоторых классов ошибок?


Het Coninkryk
CONDE UDA
of
CEYLON
in syn Binnenste
MARE DI INDIA
Staet te weeten dat al 't Landt 't welck met de Streeck-
gequanteerde linien is ingesloten, ondert gebiedt des Konings
van Conde Uda staet. Doch onder de Hollanders
behoord alles wat buyten de gemelde linie is behalven
Coylat Wanees land, waer in de Malabaren woonen;
ook in deese Kaart voorgestelt.
Plaetsen, daer de Wachten werden gehouden.
De Wegh waer door d' Autheur ontkomen is.
PANOVA REGNUM
Panova
Leawava
PAUNAO
Kleyne Baxos
Groote Baxos
Paleevepatane
Bendele
Chalmadde
Maleteure
Blockhuis
Mago
Maluwe
LAELE PROVINCIA
Iaele
BATTACILOW REGNUM
Battecalow
Vellas
Negoga
Fegama
COTIARUM
Cotiar REGNUM
VELLAS.
TAMMANQUAD
Badoura
Yotta Kinde
Maunda Kinde
Gerrewery
of Eliphants
MATURA
Kurdekoes Bay
Ettal gos
Vangh Plaets
BINTANA
OUVAH
Ouda
Henn
Gallottes
Dondere
COYLAT
Allouteneur
WALLA PONAPOY
Kinde
HABARAG OM
Dolles das
Corla
Matura
PRIN:
TENE WARA
Tenewara
WANEES
NEURE CALAVA
Anurodeburro
HORSEPOT
HATTAT
COTE MUL
COMITAT
Billigamme
Corla
Cochela
TATTANOUR
OUDANOUR
CIPAT.
Walal
Point de Galle
COMITATUS
Carla
LAND
Sette
HOTCOURLY
Handa
Pondown
HOTTERA COURLY
Billgol
Courle
Corla
Pasdum
TUN COURLY
MAUGOL COURLY
Pittigal
Corle
Putelan
Palay
Raygam
Corla
QUATRE CORLA
Sapita
Corla
Negombo
Madampe
Calpentyn
Columbo
Manaar
OCEANUS qui et MARE
ORIENTALIS INDICUM
Franssche Mylen
Spaensche Mylen
Duytsche Mylen
Lankapura.com

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. ... This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

Sir Charles Antony Richard Hoare

# NullPointerException

Можно ли при помощи статического анализа доказать, что в программе нет NPE?

# Это непростая задача

```
public class Sensitivity {
 public static void printInfo(String firstName,
                              String lastName) {
   if (firstName != null) {
     System.out.println(firstName + " " +
                        lastName.toUpperCase());
   } else {
     System.out.println("Unknown");
   }
 }
```


http://ceylon-lang.org

# В Ceylon нет NPE

```
void run() {
   String s = null;
}
```

Этот код не скомпилируется: 'null' is not assignable to 'String'

# Зато есть Nullable types

```
void run(String? s) {
   if (exists s) {
      //здесь s типа String
   }
}
```

# Union types

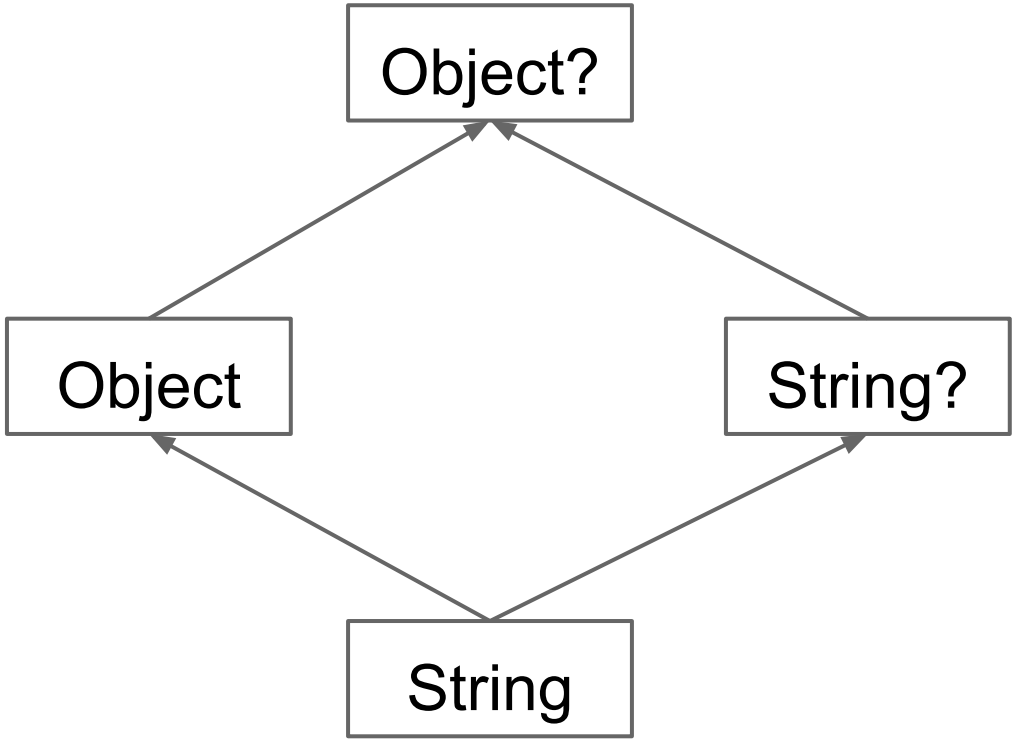
Nullable типы — частный случай union types

String? — просто сокращение для String|Null

Можно использовать List<String|Integer>

# В Kotlin тоже есть Nullable types

```
val s: String = null //error
val p: String? = null //ok
print(p.length) //error
if (p != null) {
  print(p.length) //ok
}
```


Object?
Object
String?
String

# Nullable types в Java

В Java 8 появился класс Optional:

```
public final class Optional<T> {
    public static<T> Optional<T> empty() {...}
    public static <T> Optional<T> of(T value) {...}
}
```

- но придётся писать Optional<? extends T>
- и делать явное преобразование T в Optional<T>

# Аннотации

- появились в Java 5
  - но только на декларациях
- в Java 8 могут быть везде, где встречаются типы:
  - `List<@Nullable String>`
  - `@Nullable String @NonNull[]`

# Checkers Framework

- встраивается в javac в виде annotation processor
- обрабатывает исходный код
- поддерживает несколько проверок
- можно расширять

# Nullness checker

- по умолчанию всё `@NonNull`, кроме локальных переменных
  - для локальных переменных аннотации выводятся
- для классов JDK есть готовые наборы аннотаций
- гарантирует* отсутствие NPE, если не выдал предупреждений

# Nullness checker: инициализация

```
class Person {
    final String name;
    public Person(String name) {
        PersonRegistry.register(this);
        this.name = name;
    }
}
```

# Контракты для методов

```
public void m(@Nullable String p) {
    if (p == null || p.isEmpty()) return;
    System.out.println(p.endsWith("!"));
}
```

# Контракты для методов

```
public void m(@Nullable String p) {
    if (isEmpty(p)) return;
    System.out.println(p.endsWith("!"));
}
@EnsuresNonNullIf(expression = {"#1"}, result = false)
private boolean isEmpty(@Nullable String p) {
    return p == null || p.isEmpty();
}
```

# @Nullable в IntelliJ IDEA

```
public void m(String unknown, @Nullable String nullable) {
    String s = null;//ok
    @NonNull String q = unknown;//ok
    q = nullable;//warning
}
```

# Контракты выводятся

```
public void m(@Nullable String p) {
    if (isEmpty(p)) return;
    System.out.println(p.endsWith("!"));
}
//@Contract("null->true")
private boolean isEmpty(@Nullable String p) {
    return p == null || p.isEmpty();
}
```

# Не слишком ли много аннотаций?

IntelliJ IDEA sources:

- @NotNull – 65K
- @Nullable – 200K
- в сумме занимают 2Mb

# Статистика по исключениям

| | 2011 год | 2012 год | 2013 год | 2014 год |
|---|---|---|---|---|
| NPE | 18% | 17% | 15% | 14% |
| NotNull assertion | 5% | 5% | 6% | 5% |

По продуктам на базе платформы IntelliJ IDEA, показана доля среди всех исключений.

# Аннотации помогают

- можно генерировать assertions
  - вместо `Objects.requireNonNull`
- ошибки проявляются раньше
- помогают читать код

# Что дальше?

- другие проверки в checkers framework:
  - @GuardedBy, @Holding
  - @SideEffectFree, @Pure
  - @Immutable, @ReadOnly
- собственные проверки

# Ссылки

http://types.cs.washington.edu/checker-framework/

http://pmd.sourceforge.net/

http://findbugs.sourceforge.net/

http://code.google.com/p/error-prone/

http://ceylon-lang.org/

http://kotlinlang.org/

http://eclipse.org/

http://jetbrains.org/

Всё бесплатно и open source