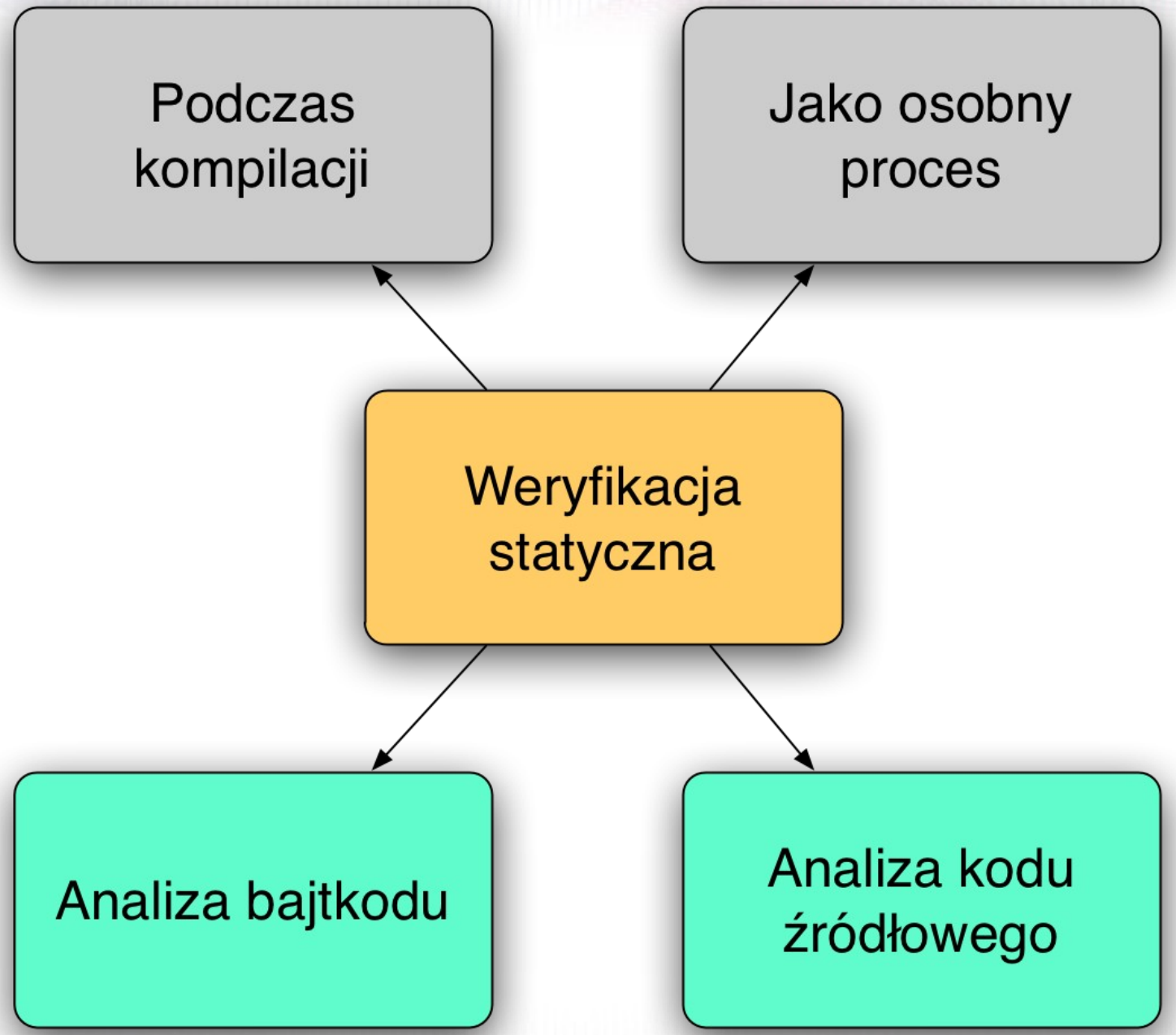
Co jeszcze mogą nam dać adnotacje?

Adam Warski


Podczas kompilacji
Jako osobny proces
Weryfikacja statyczna
Analiza bajtkodu
Analiza kodu źródłowego

Weryfikacja statyczna

Sprawdzanie własności programu bez jego uruchomienia

Weryfikacja statyczna

Najprostszy przykład:

Sprawdzanie typów

Przy wywołaniu metody parametr musi być odpowiedniego typu

To samo przy przypisaniu

FindBugs

- Analizuje bajtkod
- Osobny proces, ale:
  - Wtyczka do Mavena
  - Target Anta
- Szuka „bug patterns”
- Wykrywa bardzo wiele częstych błędów

FindBugs

- equals() method defined that doesn't override Object.equals(Object)
- int value cast to float and then passed to Math.round
- Class defines tostring(); should it be toString()
- The equals and hashCode methods of URL are blocking
- Method synchronizes on an updated field

FindBugs - adnotacje

- Adnotując nasz program udostępniamy dodatkowe informacje
- Jaki może być parametr?
- Co może zwrócić metoda?
- Dodatkowa specyfikacja
- Dokumentacja

FindBugs - null

- **NPE** – chyba najczęstszy wyjątek
- Generalnie trudno zweryfikować że program jest w 100% „NPE-safe”
- Na przykład:

```
map.get(„wiem   e to jest kluczem”);
```

FindBugs - null

- Parametr/zwracana wartość nie może być `null`: `@NonNull`
- Może być: `@Nullable`
- Trzeba sprawdzić: `@CheckForNull`

FindBugs - null

```
void test1(@NonNull Object param) {
    param.toString();     // ok
}

void test2() {
    test1(null);          // b    d
}
```

FindBugs - null

```
void test1(@Nullable Object param) {
    param.toString();     // ?
}

void test2() {
    test1(null);          // ok
    test1(new Object());  // ok
}
```

FindBugs - null

```
void test1(@CheckForNull Object param) {
    param.toString();                   // b    d
    if (param != null) {
        param.toString();               // ok
    }
}

void test2() {
    test1(null);                        // ok
    test1(new Object());                // ok
}
```

FindBugs - @CheckReturnValue

```
@CheckReturnValue
public boolean isValid() {
    // walidacja obiektu
}

public void test() {
    isValid();                  // b    d
    …
    if (isValid()) { … }  // ok
}
```

FindBugs – własny detector

Demo

FindBugs sprawdza FindBugsa

- Dużo operacji na nazwach klas
- Dwie anotacje:
  - `@DottedClassName`
  - `@SlashedClassName`

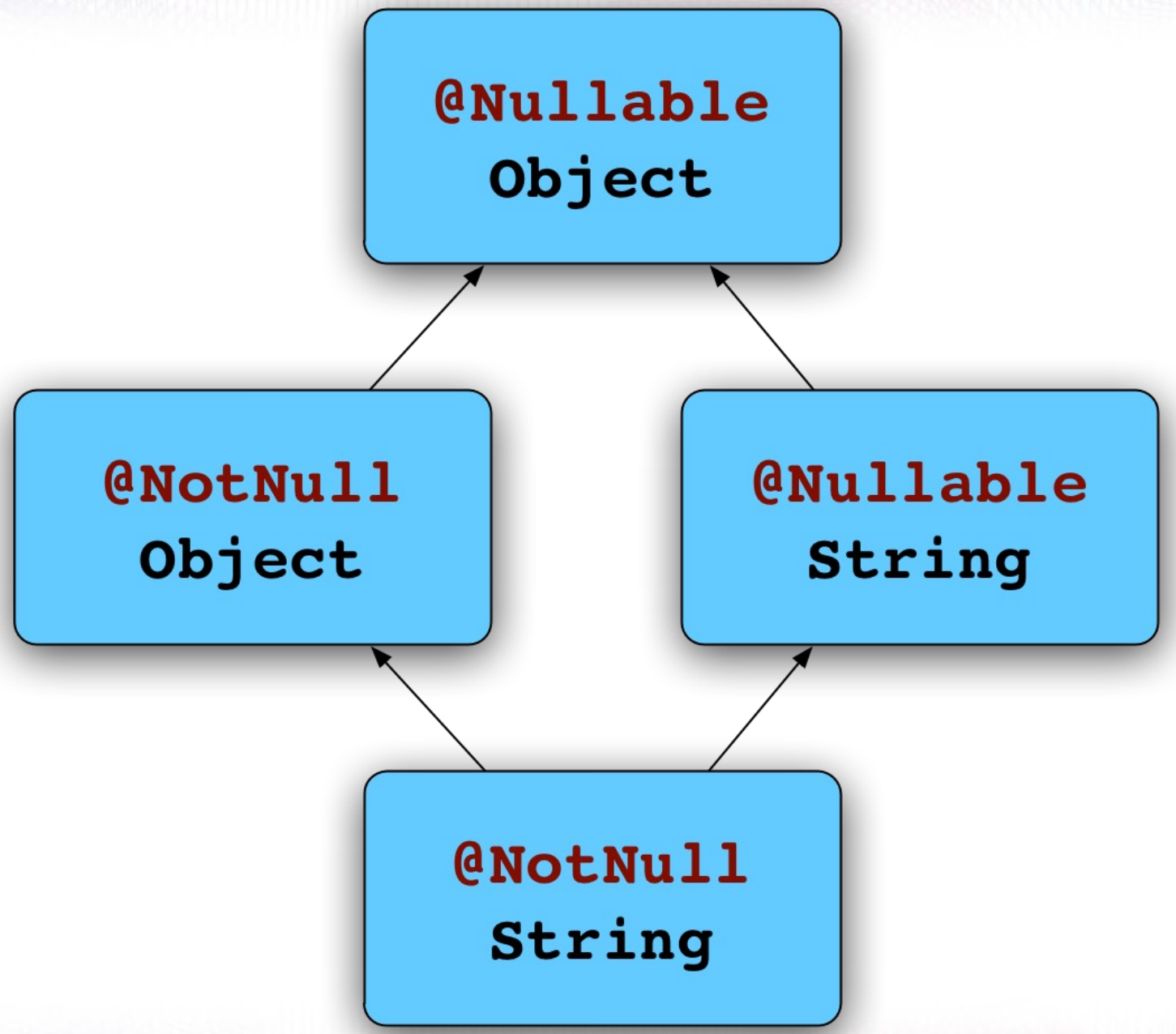
JSR308 – type annotations

- Część Javy 7
- Rozszerzenie miejsc, które możemy adnotować o **typy**
  - Wszędzie tam, gdzie występuje typ (klasa), możemy dodać adnotację

```
@NotNull String zmienna;
Map<String, @NotNull String> mapa;
class Klasa<? extends @NotNull String>
```

Ale po co?

- Chcemy powiedzieć, że obiekt może mieć ograniczony zbiór wartości
- Np. `@NonNull`: wartość `null` jest zabroniona
- Adnotowanie parametrów/metod to tylko trick


@Nullable
Object
@NotNull
Object
@Nullable
String
@NotNull
String

JSR 308 – checkers framework

- Analizuje kod źródłowy
- Sprawdza poprawne użycie adnotacji na typach
- Wymaga kompilatora Javy 7 ale …
- Część kompilacji lub osobny proces
  - Wtyczka do Mavena
  - Target Anta

JSR 308 bez Javy 7

- javac jest napisany w javie
- Można uruchomić za pomocą Javy 6
  - Wystarczy podać dobry **bootclasspath**
- Anotacje w „nowych” miejscach: w komentarzach

Nullness checker

- Adnotacje: `@NonNull`, `@Nullable`, `@PolyNull`, `@LazyNonNull`, ...
- Domyślnie: NNEL
- Robi to co FindBugs tylko ma „łatwiej”
- Podstawowe heurystyki dla map/kolekcji:
  - for-each loop (List<@NonNull String>)
  - containsKey() przed get()

IGJ Checker

- `@Immutable`
- `@ReadOnly`
- `@Mutable`
- `@I`
- Domyślnie: @RO dla zmiennych lokalnych i parametrów typowych

Inne checkery

- Interning checker
- Regex checker
- Internationalization checker
- Tainting checker
- Lock checker

Własny checker

- Tworzymy adnotacje
  - Hierarchia typów
  - Domyślnie dla (…)
- Tworzymy checker: podklasa Checker
- Uruchamiamy

```
javac -processor $CHECKER
      -proc:only
      -sourcepath $SOURCEPATH
      -cp $CLASSPATH -d build $SOURCES
```

Własny checker – @Encrypted

Demo

Co jeszcze? Typestate checker

- Typ obiektu jest stały
- Stan obiektu może się zmieniać …
  - Przez wywoływanie metod
- Typestate = stan + typ

Typestate checker

- Zmieniający się obiekt ~ automat
- Definiujemy przestrzeń stanów
- Definiujemy przejścia

- Oczywiście za pomocą anotacji :)

Typestate checker

Demo

Co dalej?

- Kto jest właścicielem obiektu? (aliasowanie)
- `@Pure`
- Scala – pluginy do kompilatora
- (...)

Linki

- http://types.cs.washington.edu/jsr308/
- http://findbugs.sourceforge.net/
- http://www.warski.org/typestate.html

Dziękuje!

Pytania?

http://www.warski.org
adam@warski.org