Using Type Annotations to Improve Your Code

Birds-of-a-Feather Session

Werner Dietl, University of Waterloo
Michael Ernst, University of Washington
Open for questions

Survey:
Did you attend the tutorial? The locking talk?
Which of these best describes you?
- Specific question / concern / feedback
- Specific problem / use case / tool
- Curious, want to learn more

Please raise questions / issues
Schedule

Java 8 syntax for type annotations
Pluggable types: a use of type annotations
Questions and discussion
Since Java 5: declaration annotations

Only for **declaration** locations:

```java
@Deprecated
class Foo {
    @Getter @Setter private String query;
    @SuppressWarnings("unchecked")
    void foo() {
        ...
    }
}
```
Java 8 adds type annotations

Annotations on all occurrences of types:

```java
@Untainted String query;
List<@NonNull String> strings;
myGraph = (@Immutable Graph) tmp;
class UnmodifiableList<T>
    implements @Readonly List<T> {}
How Java 8 treats type annotations

Stored in classfile
Handled by javac, javap, javadoc, ...
Writing type annotations has no effect unless you run an annotation processor
Write annotations before the element

Write declaration annotations before the decl.
Write type annotations before the type

```java
@Override
public @NonNull String toString() {...}
```

Don’t split them up:

```java
@NonNull
public String toString() {...}
```
Array annotations

String [][] [] a;

An array of arrays of strings
An array of arrays of Strings
Array annotations

A read-only array of non-empty arrays of English strings
Array annotations

@English  String  @ReadOnly  [][]  @NonEmpty  [][]  a;

A read-only array of non-empty arrays of English strings

Rule: write the annotation before the type
Explicit method receivers

class MyClass {
    public String toString() {}
    public boolean equals(Object other) {}
}

Explicit method receivers

class MyClass {
    public String toString() {}  
    public boolean equals(Object other) {}  
}

myval.toString();  
myval.equals(otherVal);
Explicit method receivers

class MyClass {
    public String toString(MyClass this) {}  
    public boolean equals(MyClass this, Object other) {}  
}

myval.toString();  
myval.equals(otherVal);

No impact on method binding and overloading
Explicit method receivers

class MyClass {
    public String toString(@ReadOnly MyClass this) {}{
    public boolean equals(@ReadOnly MyClass this, @ReadOnly Object other) {}{
}

myval.toString();
myval.equals(otherVal);  

Rationale: need a syntactic location for type annotations
Constructor return & receiver types

Every constructor has a return type

```java
class MyClass {
    @TReturn MyClass(@TParam String p) { ... }
}
```

Inner class constructors also have a receiver

```java
class Outer {
    class Inner {
        @TReturn Inner(@TRecv Outer Outer.this, @TParam String p) { ... }
    }
}
```
Why were type annotations added to Java?
Annotations are a specification

- More concise than English text or Javadoc
- Machine-readable
- Machine-checkable

- Improved documentation
- Improved correctness
Pluggable Type Systems

- Use Type Annotations to express properties
- Prevent errors at compile time

http://CheckerFramework.org/
Twitter: @CheckerFrmwrk
Facebook/Google+: CheckerFramework
Java's type system is too weak

Type checking prevents many errors

```java
int i = "hello";
```

Type checking doesn't prevent enough errors

```java
System.console().readLine();
Collections.emptyList().add("one");
```
Java's type system is too weak

Type checking prevents many errors

```java
int i = "hello";
```

Type checking doesn't prevent enough errors

```java
NullPointerException
System.console().readLine();
Collections.emptyList().add("one");
```
Java's type system is too weak

Type checking prevents many errors
   int i = "hello";

Type checking doesn't prevent enough errors

System
   UnsupportedOperationException

Collections.emptyList().add("one");
Solution: Pluggable Type Checking

1. Design a type system to solve a specific problem
2. Write type qualifiers in code (or, use type inference)
   ```java
   @Immutable Date date = new Date();
   date.setSeconds(0); // compile-time error
   ```
3. Type checker warns about violations (bugs)
   ```bash
   % javac -processor NullnessChecker MyFile.java
   ```
   ```bash
   MyFile.java:149: dereference of possibly-null reference bb2
   allVars = bb2.vars;
   ^
   ```
Type Checking

Source → Compiler → Executable

No errors

Fix bugs
Change types

Errors
Optional Type Checking

Source → Compiler → Executable

- Fix bugs
- Change types

Errors → Optional Type Checker → Warnings

- Fix bugs
- Add/change annotations

No errors → Guaranteed behavior
Optional Type Checking

Source → Compiler → No errors → Executable

Errors → Fix bugs → Change types

Optional Type Checker

Warnings → Fix bugs → Add/change annotations

Guaranteed behavior
Example type systems

Null dereferences (@NonNull)
Equality tests (@Interned)
Concurrency / locking (@GuardedBy)
Command injection vulnerabilities (@OsTrusted)
Privacy (@Source)
Regular expression syntax (@Regex)
printf format strings (@Format)
Signature format (@FullyQualified)
Compiler messages (@CompilerMessageKey)
Fake enumerations (@Fenum)

You can write your own checker!
Annotations in comments

List</*@NonNull*/ String> strings;

Comments for arbitrary source code

/*>>> import myquals.TRecv; */
...
int foo(/*>>> @TRecv MyClass this,*/
     @TParam String p) {...}
Static type system

Plug-in to the compiler

Doesn’t impact:

● method binding
● memory consumption
● execution

A future tool might affect run-time behavior
Problem: annotation effort

Programmer must write type annotations
  ● on program code
  ● on libraries
Very few: 1 per 100 lines, often much less
  ● depends on the type system

Solution: type inference
Type inference within a method

- Called “flow-sensitive refinement”
- A variable can have different types on different lines of code
- Low overhead
- Always used

```java
x.toString();  // warning: possible NPE
if (x!=null) {
    x.toString();  // no warning
}
x.toString();  // warning: possible NPE
```

Does not affect type signatures
Whole-program type inference

- Analyze **all** the code at once
- Determine the globally optimal annotations

Approach:
- Introduce placeholder for each location
- Use the same type rules to generate constraints
- Use a solver to find a solution

Available (beta) with the Checker Framework
Practicality

Testing

Built-in Type Systems

Guarantees

Pluggable Type Systems

Formal Verification
Conclusions

Type Annotations added in Java 8
Checker Framework for creating type checkers
  - Featureful, effective, easy to use, scalable
Prevent bugs at compile time
Improve your code!

http://CheckerFramework.org/