jax® 2016

Business Technology | Days

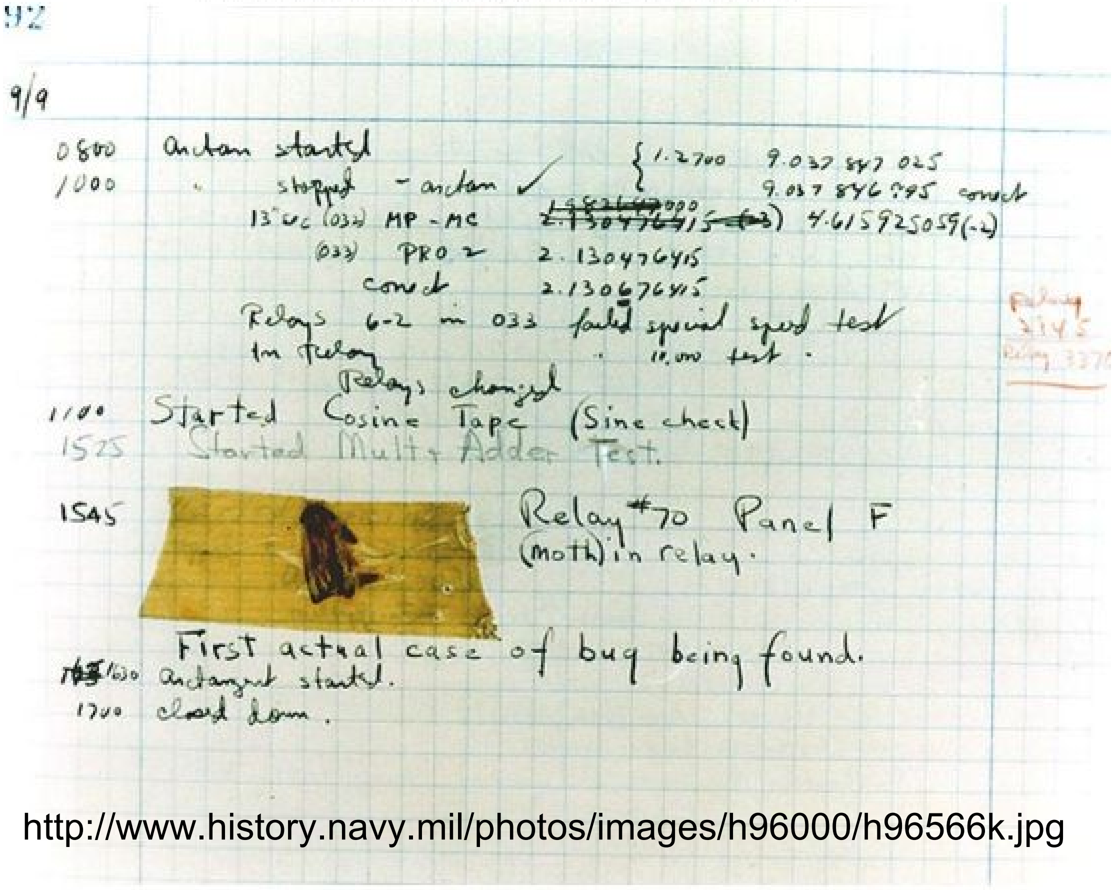
BIG DATA CON

Werner Dietl | University of Waterloo

Preventing Runtime Errors at Compile Time using the ✅*Checker Framework*

Bug Evolution

Photo # NH 96566-KN (Color) First Computer "Bug", 1947



92

9/9

0800 Antan started
1000 " stopped - antan ✓ {1.2700 9.037 847 025
9.037 846 995 conect
13" uc (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-3) 4.615925059(-2)
(033) PRO 2 2.130476415
conect 2.130676415
Relays 6-2 in 033 failed special speed test
In Relay " 11,000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Antangent started.
1700 closed down.

Relay #2145
Relay 3370

http://www.history.navy.mil/photos/images/h96000/h96566k.jpg

Bug Evolution

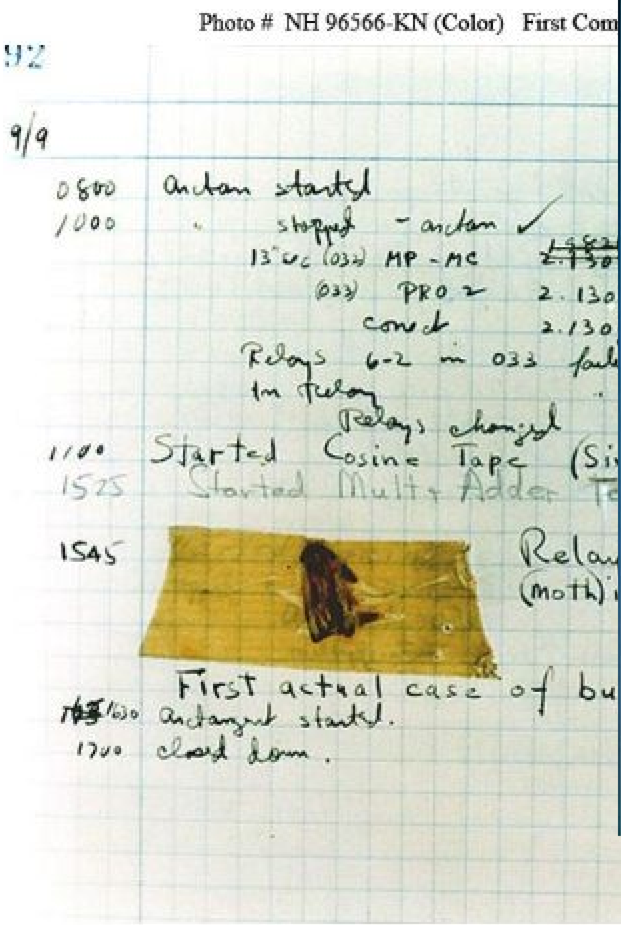


Photo # NH 96566-KN (Color) First Com

92

9/9

0800 Anctan started
1000 " stopped - anctan ✓
13" uc (033) MP - MC ~~2.130~~
(033) PRO 2 2.130
conect 2.130
Relays 6-2 in 033 fail
in relay
Relays changed
1100 Started Cosine Tape (Si
1525 Started Mult + Adder Te
1545 Relay
(moth) i
First actual case of bu
1630 Antangent started.
1700 closed down.

http://dlimages.businessweek.
com/imageserve/0arRbTIdRSg3e/630x418.jpg

Cost of software failures

$312 billion per year global cost of software bugs (2013)

$300 billion dealing with the Y2K problem

$440 million loss by Knight Capital Group Inc. in 30 minutes in August 2012

$650 million loss by NASA Mars missions in 1999; unit conversion bug

$500 million Ariane 5 maiden flight in 1996; 64 bit to 16 bit conversion bug

Software bugs can cost lives

225 deaths: jet crash caused by radar software (1997)

28 deaths: Patriot missile guidance system (1991)

11 deaths: blackout (2003)

>8 deaths: Radiation therapy (1985-2000)

2011: Software cause for 25% of all medical device recalls

Outline

- Solution: Pluggable type-checking
- Tool: Checker Framework
- Experience report
- Creating your own type system
- Java 8 type annotation features

Java's type system is too weak

Type checking prevents many errors

```
int i = “hello”;
```

Type checking doesn't prevent enough errors

```
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.console().readLine();
```

NullPointerException

```
Collections.e
```

Java's type system is too weak

Type checking prevents many errors

```
int i = “hello”;
```

Type checking doesn't prevent enough errors

```
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.console().readLine();

Collections.emptyList().add(“one”);
```

UnsupportedOperationException

Some errors are silent

```
Date date = new Date();
myMap.put(date, "now");
date.setSeconds(0);  // round to minute
myMap.get(date);
```

Some errors are silent

```
Date date = new Date();
myMap.put(date, "now");
date.setSeconds(0);  // round to minute
myMap.get(date);
```

Element not found

Some errors are silent

```
dbStatement.executeQuery(userInput);
```

Some errors are silent

```
dbStatement.executeQuery(userInput);
```

SQL injection attack

Initialization, data formatting, equality tests, …

Solution: Pluggable Type Checking

1. Design a type system to solve a specific problem

2. Write type qualifiers in code (or, use type inference)

3. Type checker warns about violations (bugs)

Solution: Pluggable Type Checking

1. Design a type system to solve a specific problem
2. Write type qualifiers in code (or, use type inference)

```
@Immutable Date date = new Date();
date.setSeconds(0); // compile-time error
```

3. Type checker warns about violations (bugs)

Solution: Pluggable Type Checking

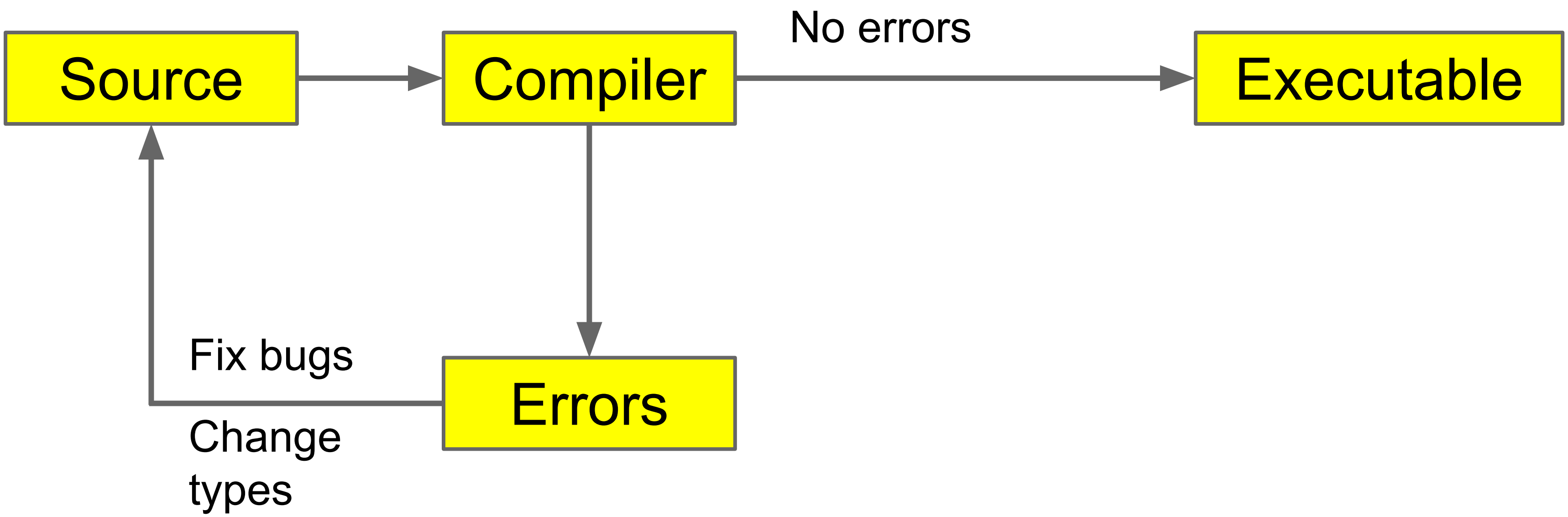
1. Design a type system to solve a specific problem
2. Write type qualifiers in code (or, use type inference)

```
@Immutable Date date = new Date();
date.setSeconds(0); // compile-time error
```

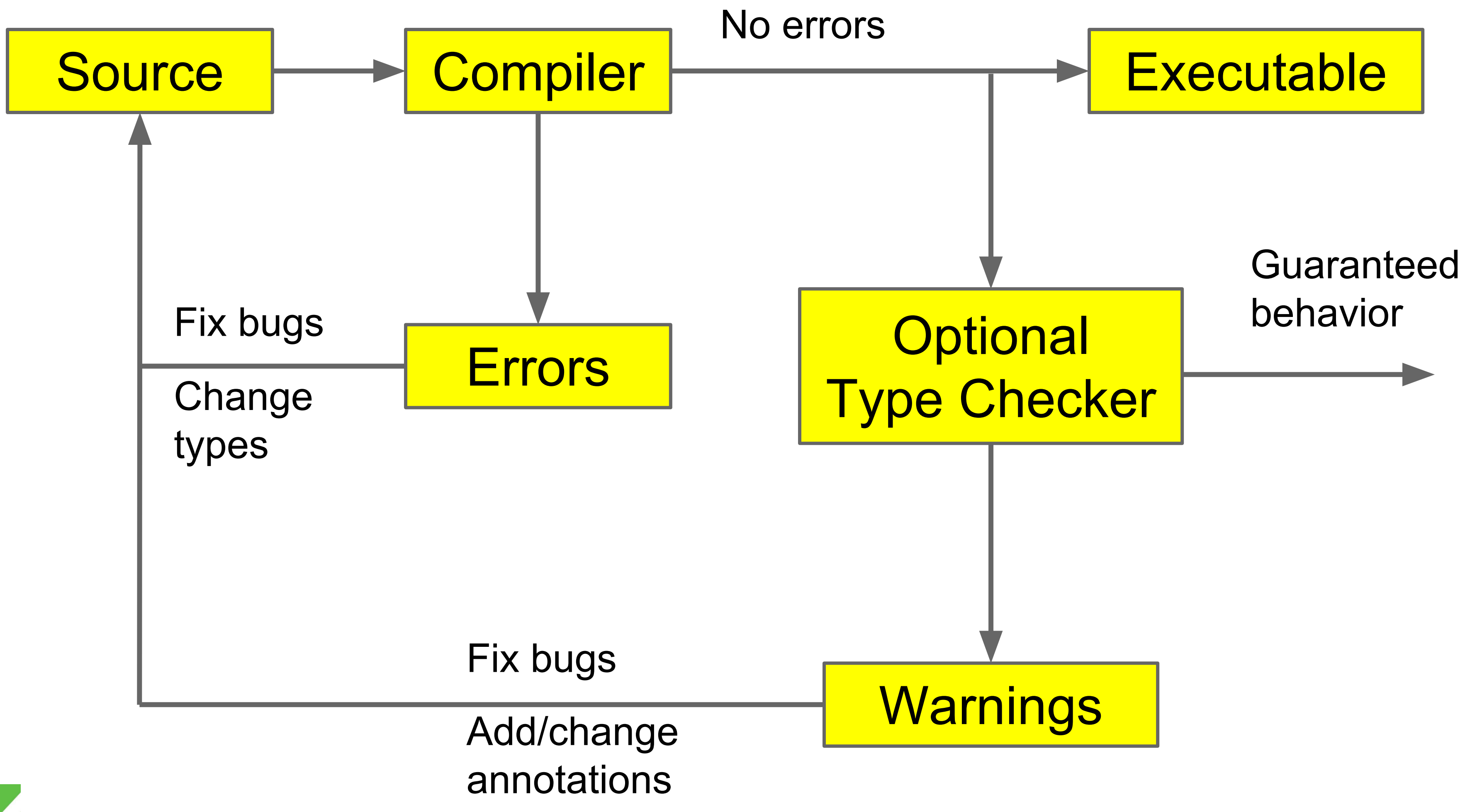
3. Type checker warns about violations (bugs)

```
% javac -processor NullnessChecker MyFile.java
MyFile.java:149: dereference of possibly-null
   reference bb2
      allVars = bb2.vars;
                   ^
```

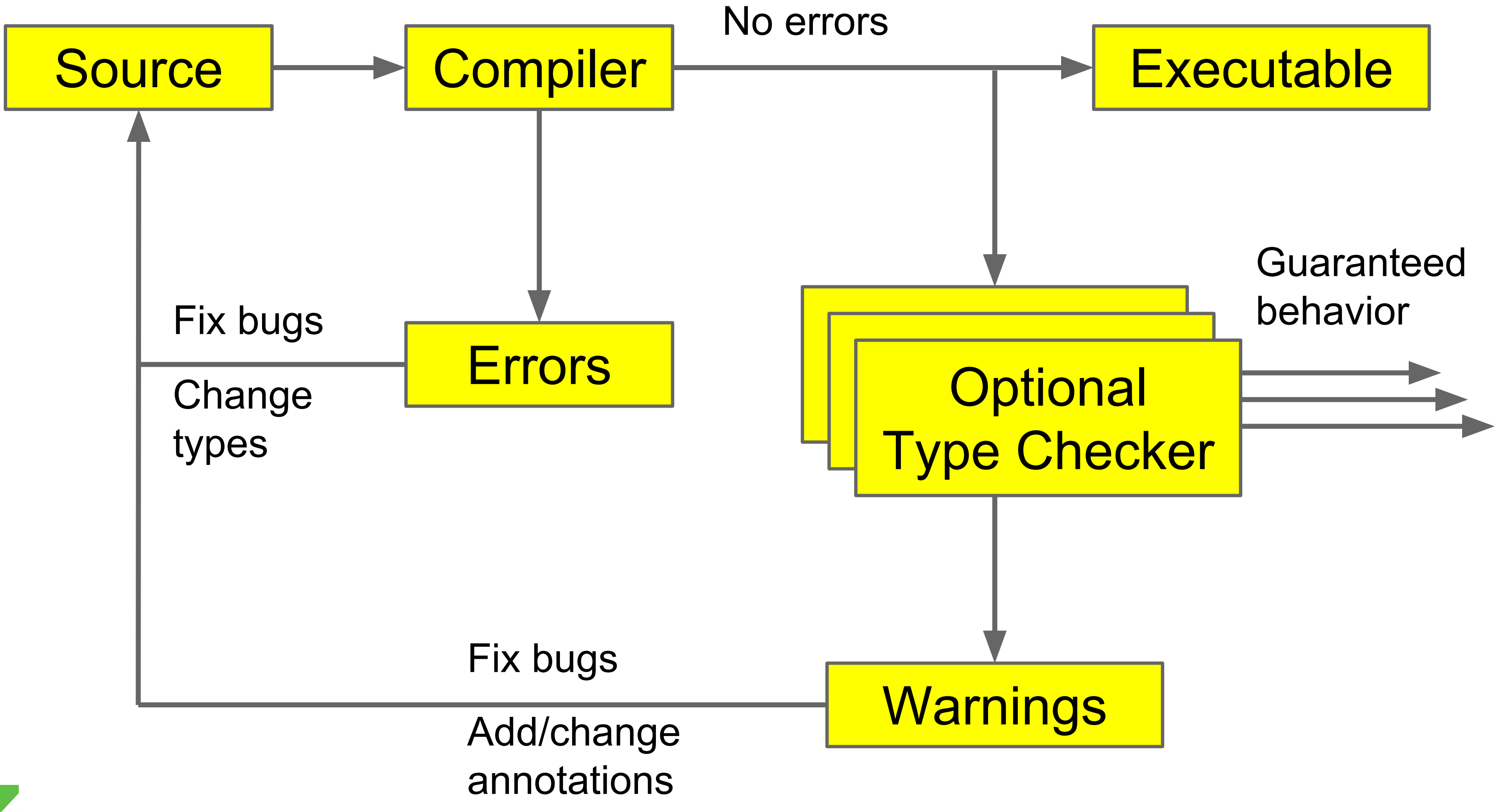
Type Checking



Optional Type Checking



Optional Type Checking



Prevent null pointer exceptions

Type system that statically guarantees that
the program only dereferences
known non-null references

Types of data

`@NonNull`	reference is never null
`@Nullable`	reference may be null

Null pointer exception

```
String op(Data in) {
  return “transform: ” + in.getF();
}
…
String s = op(null);
```

Null pointer exception

Where is the defect?

```
String op(Data in) {
  return “transform: ” + in.getF();
}
…
String s = op(null);
```

Null pointer exception

Where is the defect?

```
String op(Data in) {
  return “transform: ” + in.getF();
}
…
String s = op(null);
```

Null pointer exception

Where is the defect?

```
String op(Data in) {
  return “transform: ” + in.getF();
}
…
String s = op(null);
```

Can’t decide without specification!

Specification 1: non-null parameter

```
String op(@NonNull Data in) {
  return “transform: ” + in.getF();
}
...
String s = op(null);
```

Specification 1: non-null parameter

```
String op(@NonNull Data in) {
  return “transform: ” + in.getF();
}
…
String s = op(null);     // error
```

Specification 2: nullable parameter

```
String op(@Nullable Data in) {
  return “transform: ” + in.getF();
}
...
String s = op(null);
```

Specification 2: nullable parameter

```
String op(@Nullable Data in) {
  return “transform: ” + in.getF();
}                          // error
…
String s = op(null);
```

More @ JAX 2016

Hands-on with the *Checker Framework*: Preventing Null Pointer Exceptions at Compile Time

Today! 14:45 - 15:45

Goldsaal C

Benefits of type systems

- **Find bugs** in programs
    - Guarantee the **absence of errors**
- **Improve documentation**
    - Improve code structure & maintainability
- Aid compilers, optimizers, and analysis tools
    - E.g., could reduce number of run-time checks

Benefits of type systems

- **Find bugs** in programs
  - Guarantee the **absence of errors**
- **Improve documentation**
  - Improve code structure & maintainability
- Aid compilers, optimizers, and analysis tools
  - E.g., could reduce number of run-time checks

- Possible negatives:
  - Must write the types (or use type inference)
  - False positives are possible (can be suppressed)

Input Format Validation

Demo: ensure that certain strings contain **valid regular expressions**.

Regular Expression Example

```
public static void main(String[] args) {
  String regex = args[0];
  String content = args[1];
  Pattern pat = Pattern.compile(regex);
  Matcher mat = pat.matcher(content);
  if (mat.matches()) {
    System.out.println("Group: " + mat.group(1));
  }
}
```

Regular Expression Example

```
public static void main(String[] args) {
  String regex = args[0];
  String conten
  Pattern pat =
  Matcher mat = pat.matcher(content);
  if (mat.matches()) {
    System.out.println("Group: " + mat.group(1));
  }
}
```

PatternSyntaxException

IndexOutOfBoundsExceptionon

Fixing the Errors

`Pattern.compile`	only on valid regex
`Matcher.group(i)`	only if > i groups

```
...
if (!RegexUtil.isRegex(regex, 1)) {
  System.out.println("Invalid: " + regex);
  System.exit(1);
}
...
```

The Checker Framework

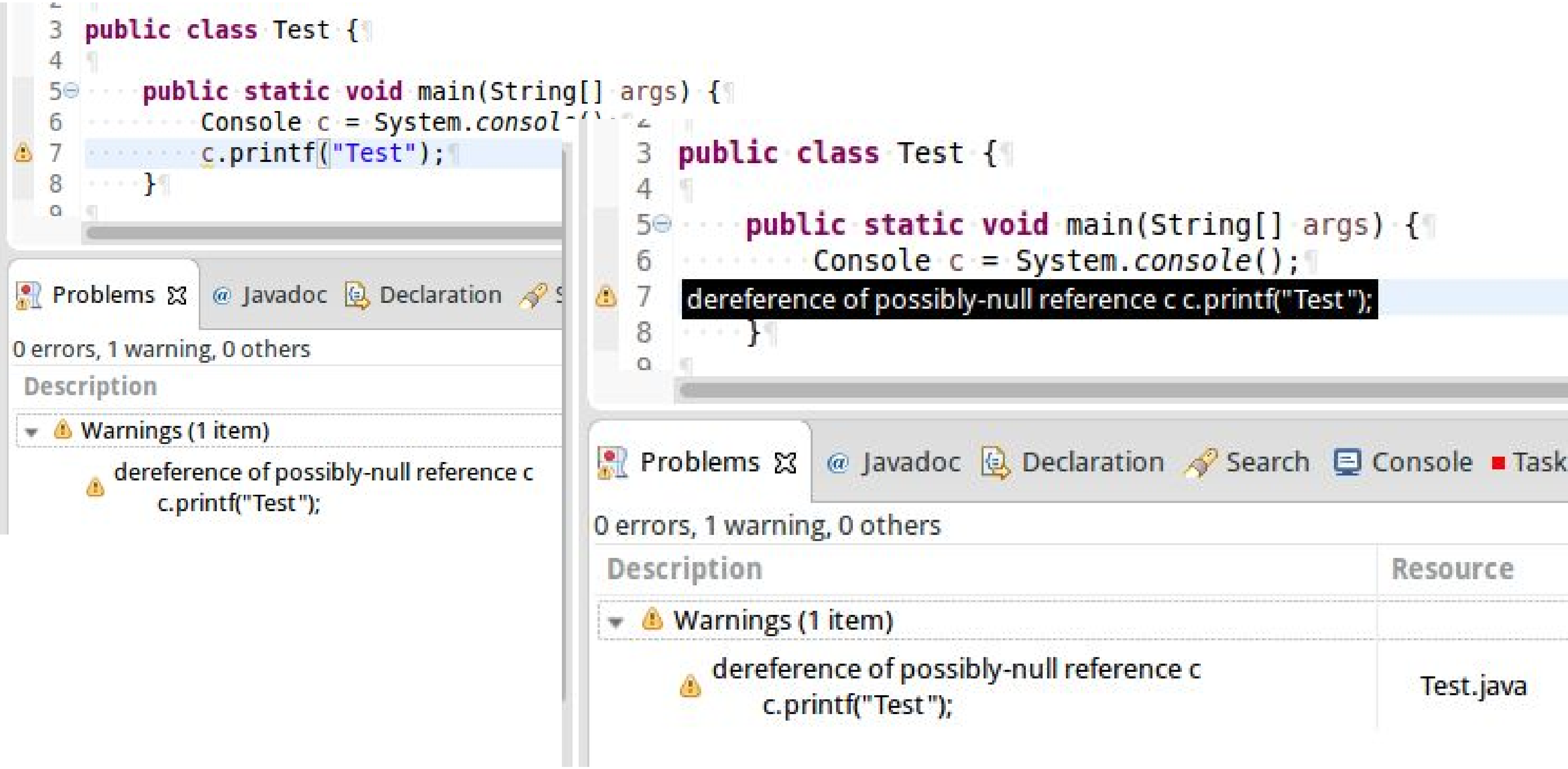
A framework for pluggable type checkers

“Plugs” into the OpenJDK or OracleJDK compiler

```
    javac -processor MyChecker …
```

Standard error format allows tool integration

Eclipse plug-in



```java
public class Test {

    public static void main(String[] args) {
        Console c = System.consol
        c.printf("Test");
    }

```

0 errors, 1 warning, 0 others

Description

Warnings (1 item)

dereference of possibly-null reference c c.printf("Test");

```java
public class Test {

    public static void main(String[] args) {
        Console c = System.console();
dereference of possibly-null reference c c.printf("Test");
    }

```

0 errors, 1 warning, 0 others

Description | Resource

Warnings (1 item)

dereference of possibly-null reference c c.printf("Test"); | Test.java

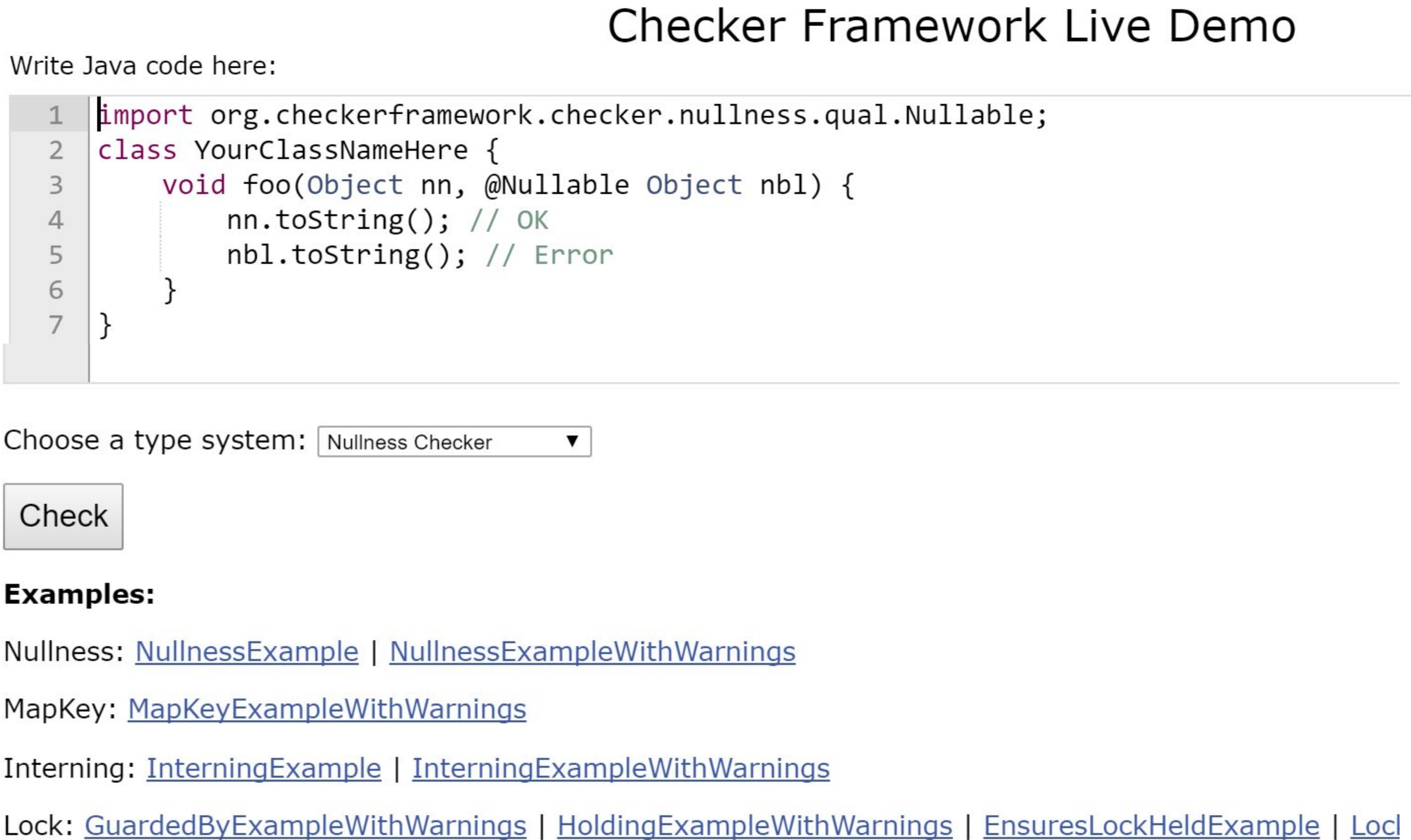
Ant and Maven integration

```
<presetdef name="jsr308.javac">
  <javac fork="yes"
    executable="${checkerframework}/checker/bin/${cfJavac}" >
    <!-- JSR-308-related compiler arguments -->
    <compilerarg value="-version"/>
    <compilerarg value="-implicit:class"/>
  </javac>
</presetdef>
```

```
<dependencies>
  ... existing <dependency> items ...
  <!-- annotations from the Checker Framework:
       nullness, interning, locking, ... -->
    <dependency>
      <groupId>org.checkerframework</groupId>
      <artifactId>checker-qual</artifactId>
      <version>1.9.7</version>
    </dependency>
</dependencies>
```

Web interface

http://eisop.uwaterloo.ca/live/



Checker Framework Live Demo

Write Java code here:

```
import org.checkerframework.checker.nullness.qual.Nullable;
class YourClassNameHere {
    void foo(Object nn, @Nullable Object nbl) {
        nn.toString(); // OK
        nbl.toString(); // Error
    }
}
```

Choose a type system: Nullness Checker ▼

Examples:

Nullness: NullnessExample | NullnessExampleWithWarnings

MapKey: MapKeyExampleWithWarnings

Interning: InterningExample | InterningExampleWithWarnings

Lock: GuardedByExampleWithWarnings | HoldingExampleWithWarnings | EnsuresLockHeldExample | Locl

Example type systems

Null dereferences (`@NonNull`)

>200 errors in Google Collections, javac, ...

Equality tests (`@Interned`)

>200 problems in Xerces, Lucene, ...

Concurrency / locking (`@GuardedBy`)

>500 errors in BitcoinJ, Derby, Guava, Tomcat, ...

Fake enumerations (`@Fenum`)

problems in Swing, JabRef

String type systems

Regular expression syntax (`@Regex`)

- 56 errors in Apache, etc.; 200 annos

printf format strings (`@Format`)

- 104 errors, only 107 annotations required

Signature format (`@FullyQualified`)

- 28 errors in OpenJDK, ASM, AFU

Compiler messages (`@CompilerMessageKey`)

- 8 wrong keys in Checker Framework

Security type systems

Command injection vulnerabilities (`@OsTrusted`)

- 5 missing validations in Hadoop

Privacy (`@Source`)

- SPARTA detected malware in Android apps

You can write your own checker!

Brainstorming new type checkers

What runtime exceptions to prevent?
What properties of data should always hold?
What operations are legal and illegal?

Type-system checkable properties:

- Dependency on values
- Not on program structure, timing, ...

Example: Nullness Checker

What runtime exceptions to prevent?

What properties of data should always hold?

What operations are legal and illegal?

Example: Nullness Checker

What runtime exceptions to prevent?

NullPointerException

What properties of data should always hold?

What operations are legal and illegal?

Example: Nullness Checker

What runtime exceptions to prevent?

NullPointerException

What properties of data should always hold?

@NonNull references always non-null

What operations are legal and illegal?

Example: Nullness Checker

What runtime exceptions to prevent?

NullPointerException

What properties of data should always hold?

@NonNull references always non-null

What operations are legal and illegal?

Dereferences only on @NonNull references

Example: Regex Checker

What runtime exceptions to prevent?

What properties of data should always hold?

What operations are legal and illegal?

Example: Regex Checker

What runtime exceptions to prevent?

PatternSyntaxException,
IndexOutOfBoundsException

What properties of data should always hold?

What operations are legal and illegal?

Example: Regex Checker

What runtime exceptions to prevent?

PatternSyntaxException, IndexOutOfBoundsException

What properties of data should always hold?

Whether a string is a regex and number of groups

What operations are legal and illegal?

Example: Regex Checker

What runtime exceptions to prevent?

PatternSyntaxException, IndexOutOfBoundsException

What properties of data should always hold?

Whether a string is a regex and number of groups

What operations are legal and illegal?

Pattern.compile with non-@Regexp, etc,

New type system

What runtime exceptions to prevent?

1

What properties of data should always hold?

2

What operations are legal and illegal?

3

New type system

What runtime exceptions to prevent?

1

What properties of data should always hold?

2

What operations are legal and illegal?

3

New type system

What runtime exceptions to prevent?

1

What properties of data should always hold?

2

What operations are legal and illegal?

3

Checkers are usable

- Type-checking is familiar to programmers
- Modular: fast, incremental, partial programs
- Annotations are not too verbose
  - **`@NonNull`**: 1 per 75 lines
  - **`@Interned`**: 124 annotations in 220 KLOC revealed 11 bugs
  - **`@Format`**: 107 annotations in 2.8 MLOC revealed 104 bugs
  - Possible to annotate part of program
  - Fewer annotations in new code
- Few false positives
- First-year CS majors preferred using checkers to not
- **Practical**: in daily use at Google, on Wall Street, etc.

Comparison: other nullness tools

	Null pointer errors		False warnings	Annotations written
	Found	Missed		
Checker Framework	8	0	4	35
FindBugs	0	8	1	0
Jlint	0	8	8	0
PMD	0	8	0	0

Checking the Lookup program for file system searching (4kLOC)

False warnings are suppressed via an annotation or assertion

What a checker guarantees

The program satisfies the type property. There are:

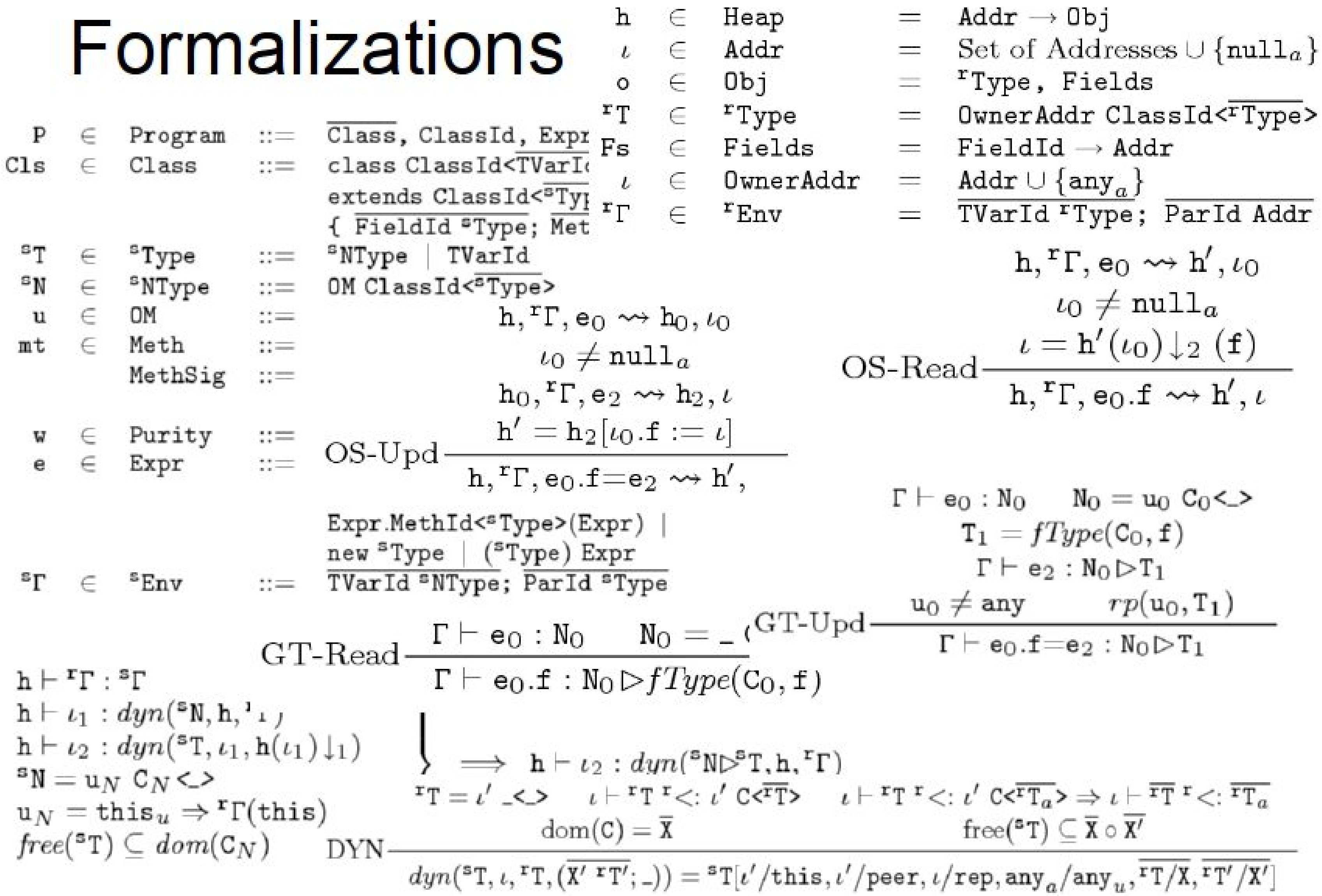
- No bugs (of particular varieties)
- No wrong annotations

Caveat 1: only for code that is checked

- Native methods (but handles reflection!)
- Code compiled without the pluggable type checker
- Suppressed warnings
  - Indicates what code a human should analyze
- Checking part of a program is still useful

Caveat 2: The checker itself might contain an error

Formalizations



$\mathtt{P}$	$\in$	$\mathtt{Program}$	$::=$	$\overline{\mathtt{Class}}$, $\mathtt{ClassId}$, $\mathtt{Expr}$
$\mathtt{Cls}$	$\in$	$\mathtt{Class}$	$::=$	$\mathtt{class\ ClassId<}\overline{\mathtt{TVarI}}$
				$\mathtt{extends\ ClassId<{}^{s}Typ}$
				$\mathtt{\{\ }\overline{\mathtt{FieldId\ {}^{s}Type;}}\ \mathtt{Met}$
${}^{s}\mathtt{T}$	$\in$	${}^{s}\mathtt{Type}$	$::=$	${}^{s}\mathtt{NType}\ \vert\ \mathtt{TVarId}$
${}^{s}\mathtt{N}$	$\in$	${}^{s}\mathtt{NType}$	$::=$	$\mathtt{OM\ ClassId<}\overline{{}^{s}\mathtt{Type}}\mathtt{>}$
$\mathtt{u}$	$\in$	$\mathtt{OM}$	$::=$	
$\mathtt{mt}$	$\in$	$\mathtt{Meth}$	$::=$	
		$\mathtt{MethSig}$	$::=$	
$\mathtt{w}$	$\in$	$\mathtt{Purity}$	$::=$	
$\mathtt{e}$	$\in$	$\mathtt{Expr}$	$::=$	
				$\mathtt{Expr.MethId<}\overline{{}^{s}\mathtt{Type}}\mathtt{>(}\overline{\mathtt{Expr}}\mathtt{)}\ \vert$
				$\mathtt{new\ {}^{s}Type}\ \vert\ \mathtt{({}^{s}Type)\ Expr}$
${}^{s}\Gamma$	$\in$	${}^{s}\mathtt{Env}$	$::=$	$\overline{\mathtt{TVarId\ {}^{s}NType;}}\ \overline{\mathtt{ParId\ {}^{s}Type}}$

$\mathtt{h}$	$\in$	$\mathtt{Heap}$	$=$	$\mathtt{Addr} \to \mathtt{Obj}$
ι	$\in$	$\mathtt{Addr}$	$=$	Set of Addresses $\cup \{\mathtt{null}_a\}$
$\mathtt{o}$	$\in$	$\mathtt{Obj}$	$=$	${}^{r}\mathtt{Type},\ \mathtt{Fields}$
${}^{r}\mathtt{T}$	$\in$	${}^{r}\mathtt{Type}$	$=$	$\mathtt{OwnerAddr\ ClassId<}\overline{{}^{r}\mathtt{Type}}\mathtt{>}$
$\mathtt{Fs}$	$\in$	$\mathtt{Fields}$	$=$	$\mathtt{FieldId} \to \mathtt{Addr}$
ι	$\in$	$\mathtt{OwnerAddr}$	$=$	$\mathtt{Addr} \cup \{\mathtt{any}_a\}$
${}^{r}\Gamma$	$\in$	${}^{r}\mathtt{Env}$	$=$	$\overline{\mathtt{TVarId\ {}^{r}Type;}}\ \overline{\mathtt{ParId\ Addr}}$

$$\text{OS-Upd}\ \frac{\begin{array}{c}\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}_0, \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \mathtt{h}_0, {}^{r}\Gamma, \mathtt{e}_2 \rightsquigarrow \mathtt{h}_2, \iota \\ \mathtt{h}' = \mathtt{h}_2[\iota_0.\mathtt{f} := \iota]\end{array}}{\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 \rightsquigarrow \mathtt{h}',}$$

$$\text{OS-Read}\ \frac{\begin{array}{c}\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}', \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \iota = \mathtt{h}'(\iota_0)\downarrow_2(\mathtt{f})\end{array}}{\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0.\mathtt{f} \rightsquigarrow \mathtt{h}', \iota}$$

$$\text{GT-Upd}\ \frac{\begin{array}{c}\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \mathtt{u}_0\ \mathtt{C}_0\mathtt{<\_>} \\ \mathtt{T}_1 = \mathit{fType}(\mathtt{C}_0, \mathtt{f}) \\ \Gamma \vdash \mathtt{e}_2 : \mathtt{N}_0 \rhd \mathtt{T}_1 \\ \mathtt{u}_0 \neq \mathtt{any} \qquad \mathit{rp}(\mathtt{u}_0, \mathtt{T}_1)\end{array}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 : \mathtt{N}_0 \rhd \mathtt{T}_1}$$

$$\text{GT-Read}\ \frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \_\ (}{\Gamma \vdash \mathtt{e}_0.\mathtt{f} : \mathtt{N}_0 \rhd \mathit{fType}(\mathtt{C}_0, \mathtt{f})}$$

$$\left.\begin{array}{l}\mathtt{h} \vdash {}^{r}\Gamma : {}^{s}\Gamma \\ \mathtt{h} \vdash \iota_1 : \mathit{dyn}({}^{s}\mathtt{N}, \mathtt{h}, {}^{r} \\ \mathtt{h} \vdash \iota_2 : \mathit{dyn}({}^{s}\mathtt{T}, \iota_1, \mathtt{h}(\iota_1)\downarrow_1) \\ {}^{s}\mathtt{N} = \mathtt{u}_N\ \mathtt{C}_N\mathtt{<>} \\ \mathtt{u}_N = \mathtt{this}_u \Rightarrow {}^{r}\Gamma(\mathtt{this}) \\ \mathit{free}({}^{s}\mathtt{T}) \subseteq \mathit{dom}(\mathtt{C}_N)\end{array}\right\} \Longrightarrow \mathtt{h} \vdash \iota_2 : \mathit{dyn}({}^{s}\mathtt{N} \rhd {}^{s}\mathtt{T}, \mathtt{h}, {}^{r}\Gamma)$$

$$\text{DYN}\ \frac{\begin{array}{c}{}^{r}\mathtt{T} = \iota'\ \mathtt{\_<\_>} \qquad \iota \vdash {}^{r}\mathtt{T}\ {}^{r}{<:}\ \iota'\ \mathtt{C<}\overline{{}^{r}\mathtt{T}}\mathtt{>} \qquad \iota \vdash {}^{r}\mathtt{T}\ {}^{r}{<:}\ \iota'\ \mathtt{C<}\overline{{}^{r}\mathtt{T}_a}\mathtt{>} \Rightarrow \iota \vdash \overline{{}^{r}\mathtt{T}}\ {}^{r}{<:}\ \overline{{}^{r}\mathtt{T}_a} \\ \mathrm{dom}(\mathtt{C}) = \overline{\mathtt{X}} \qquad \mathrm{free}({}^{s}\mathtt{T}) \subseteq \overline{\mathtt{X}} \circ \overline{\mathtt{X}'}\end{array}}{\mathit{dyn}({}^{s}\mathtt{T}, \iota, {}^{r}\mathtt{T}, (\overline{\mathtt{X}'\ {}^{r}\mathtt{T}'}; \_)) = {}^{s}\mathtt{T}[\iota'/\mathtt{this}, \iota'/\mathtt{peer}, \iota/\mathtt{rep}, \mathtt{any}_a/\mathtt{any}_u, \overline{{}^{r}\mathtt{T}/\mathtt{X}}, \overline{{}^{r}\mathtt{T}'/\mathtt{X}'}]}$$

Since Java 5: declaration annotations

Only for declaration locations:

```
@Deprecated
class Foo {
  @Getter @Setter private String query;
  @SuppressWarnings(“unchecked”)
  void foo() { … }
}
```

But we couldn’t express

A non-null reference to my data

An interned String

A non-null List of English Strings

A non-empty array of English strings

With Java 8 Type Annotations we can!

A non-null reference to my data

```
    @NonNull Data mydata;
```

An interned String

```
    @Interned String query;
```

A non-null List of English Strings

```
    @NonNull List<@English String> msgs;
```

A non-empty array of English strings

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

Java 8 extends annotation syntax

Annotations on all occurrences of types:

```
@Untainted String query;
List<@NonNull String> strings;
myGraph = (@Immutable Graph) tmp;
class UnmodifiableList<T>
  implements @Readonly List<T> {}
```

Stored in classfile

Handled by javac, javap, javadoc, …

Java 6 & 7 compatibility

Annotations in comments:

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

(Requires use of jsr308-langtools compiler.)

Array annotations

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

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

Explicit method receivers

```
class MyClass {
  int foo(@TParam String p) {…}
  int foo(@TRecv MyClass this,
          @TParam String p) {…}
```

No impact on method binding and overloading

Constructor return & receiver types

Every constructor has a return type

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

Inner class constructors also have a receiver

```
class Outer {
  class Inner {
    @TReturn Inner(@TRecv Outer Outer.this,
                   @TParam String p) {...}
```

Annotating external libraries

When type-checking clients, need library spec

Can write manually or automatically infer

Two syntaxes:

- As separate text file (stub file)
- Within its .jar file (from annotated partial source code)

Checker Framework facilities

- Full type systems: inheritance, overriding, ...
- Generics (type polymorphism)
  - Also qualifier polymorphism
- Qualifier defaults
- Dataflow framework
- Pre-/post-conditions
- Warning suppression
- Testing infrastructure

Building a checker is easy

Example: Ensure encrypted communication

```
void send(@Encrypted String msg) {…}
@Encrypted String msg1 = ...;
send(msg1);   // OK
String msg2 = ....;
send(msg2);   // Warning!
```

Building a checker is easy

Example: Ensure encrypted communication

```
void send(@Encrypted String msg) {…}
@Encrypted String msg1 = ...;
send(msg1);   // OK
String msg2 = ....;
send(msg2);   // Warning!
```

The complete checker:

```
@Target(ElementType.TYPE_USE)
@SubtypeOf(Unqualified.class)
public @interface Encrypted {}
```

Testing infrastructure

jtreg-based testing as in OpenJDK

Lightweight tests with in-line expected errors:

```
String s = "%+s%";
//:: error: (format.string.invalid)
f.format(s, "illegal");
```

Verification

- **Goal**:
  prove that no bug exists
- **Specifications**:
  user provides
- **False negatives**:
  none
- **False positives**:
  user suppresses warnings
- **Downside**:
  user burden

Bug-finding

- **Goal**:
  find some bugs at low cost
- **Specifications**:
  infer likely specs
- **False negatives:**
  acceptable
- **False positives**:
  heuristics focus on most important bugs
- **Downside**:
  missed bugs

Neither is “better”; each is appropriate in certain circumstances.

More @ JAX 2016

Hands-on with the *Checker Framework:* Preventing Null Pointer Exceptions at Compile Time

Today! 14:45 - 15:45

Goldsaal C

Community

Open source project:

`https://github.com/typetools/checker-framework`

Community:

- uWashington: Michael Ernst, Suzanne Millstein, Javier Thaine, Dan Brown …
- uWaterloo: Werner Dietl, Jeff Luo, Jason Li, Mier Ta, Charles Chen, …
- Bug reports, test cases, patches, … from users

Conclusions

Checker Framework for creating type checkers

- Featureful, effective, easy to use, scalable

Prevent bugs at compile time

Create custom type-checkers

Improve your code!

`http://CheckerFramework.org/`

@CheckerFrmwrk on Twitter
CheckerFramework on Facebook & Google+