The Checker Framework in Action

Preventing Errors Before They Happen

`http://CheckerFramework.org/`

Twitter: @CheckerFrmwrk

Live demo: `http://eisop.uwaterloo.ca/live`

Werner Dietl, University of Waterloo

Michael Ernst, University of Washington

UNIVERSITY OF WATERLOO

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

1997: **225 deaths**: jet crash caused by radar software
1991: **28 deaths**: Patriot missile guidance system
2003: **11 deaths**: blackout
1985-2000: **>8 deaths**: Radiation therapy

2011: Software caused 25% of all medical device recalls

Outline

- Solution: Pluggable type-checking
- Tool: Checker Framework
- How to use it

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();
```

Java's type system is too weak

Type checking prevents many errors

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

Type checking doesn't prevent enough errors

NullPointerException

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

Java's type system is too weak

Type checking prevents many errors

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

Type checking doesn't prevent enough errors

```
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

```
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);
```

Corrupted map

Some errors are silent

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

Some errors are silent

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

SQL injection attack

Initialization, data formatting, equality tests, …

Prevent information leakage

Goal: the program sends only
encrypted data across the network

Types of data:

`@Encrypted`	data is encrypted
`@Unencrypted`	data might be plaintext

Information leakage

```
int op(String in) {
   …
   sendToNetwork(in);
}
…
int i = op(unencryptedData);
```

Information leakage

Where is the defect?

```
int op(String in) {
   …
   sendToNetwork(in);
}
…
int i = op(unencryptedData);
```

Information leakage

```
int op(String in) {
   …
   sendToNetwork(in);
}
…
int i = op(unencryptedData);
```

Where is the defect?

Information leakage

```
int op(String in) {
   …
   sendToNetwork(in);
}
…
int i = op(unencryptedData);
```

Where is the defect?

Can’t decide without specification!

Specification 1: encrypted parameter

```
int op(@Encrypted String in) {
   …
   sendToNetwork(in);
}
…
int i = op(unencryptedData);
```

Specification 1: encrypted parameter

```
int op(@Encrypted String in) {
   …
   sendToNetwork(in);
}
…
int i = op(unencryptedData);  // error
```

Specification 2: unencrypted parameter

```
int op(@Unencrypted String in) {
   …
   sendToNetwork(in);
}
…
int i = op(unencryptedData);
```

Specification 2: unencrypted parameter

```
int op(@Unencrypted String in) {
  …
  sendToNetwork(in);              // error
}
…
int i = op(unencryptedData);
```

Demo: Preventing SQL injection

Goal: don't execute user input as a SQL command

```
private String wrapQuery(String s) {
  return "SELECT * FROM User WHERE userId='" + s + "'";
}
```

If a user inputs his name as: `' or 'x'='x`
the SQL query is: `... WHERE userID='' or 'x'='x'`

@Tainted = untrusted user input

@Untainted = sanitized, safe to use

Solution: Pluggable Type Checking

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

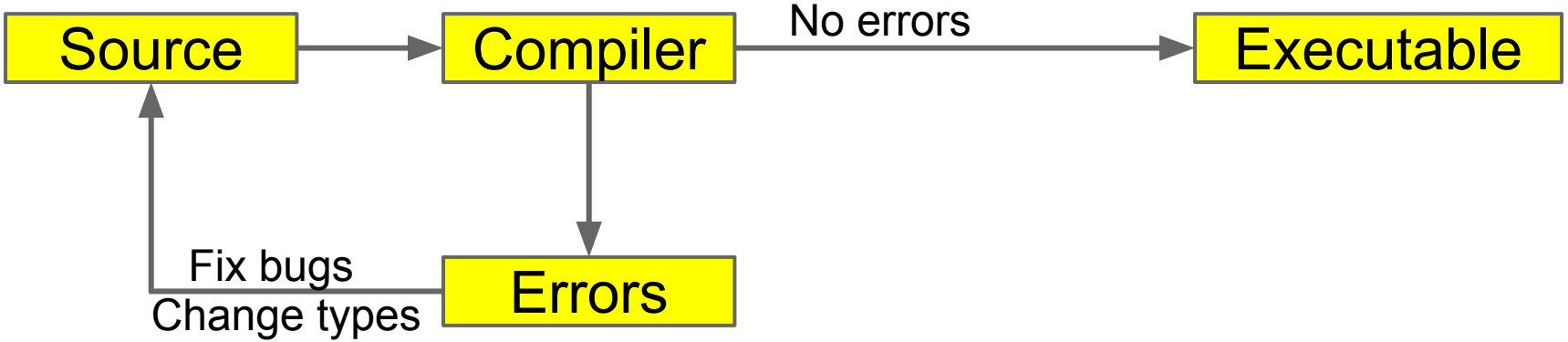
```
@Immutable Date date = new Date();
date.setSeconds(0);  // compile-time error
```

3. Type checker warns about violations (bugs)

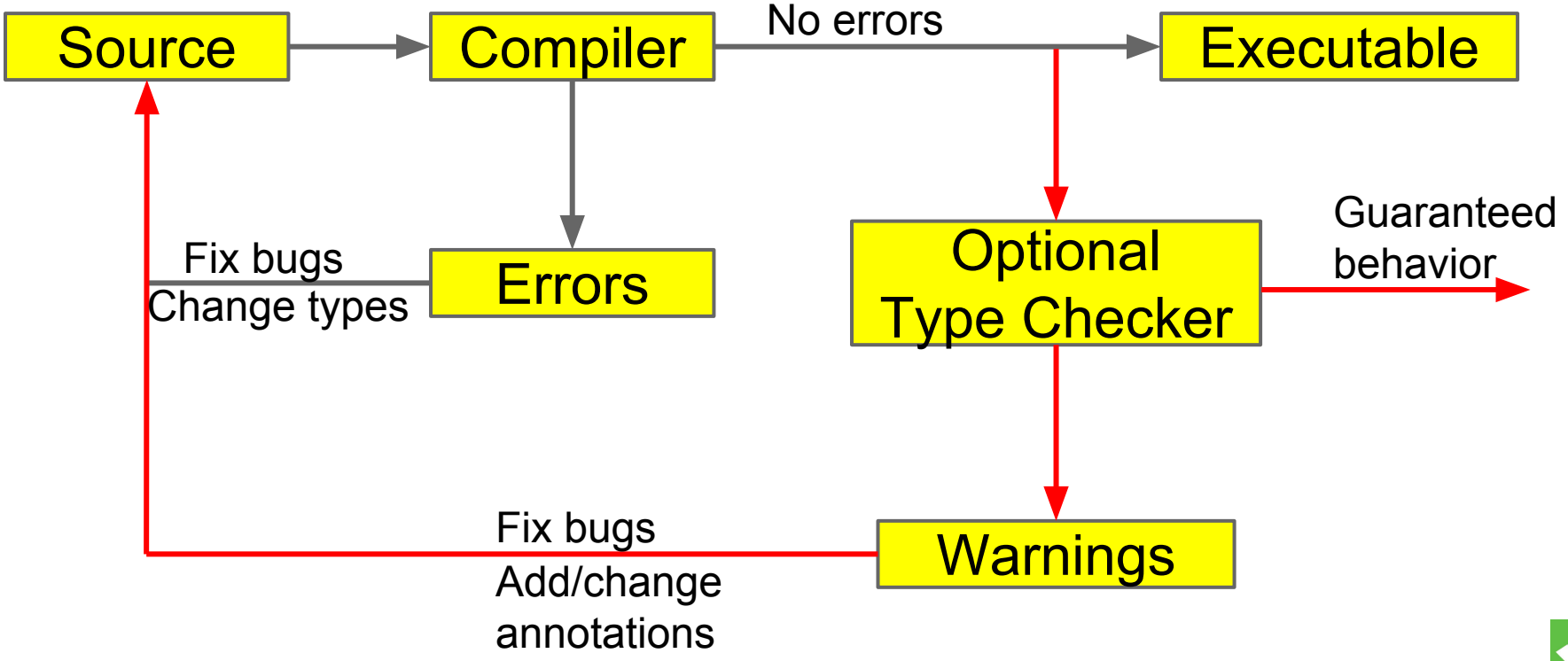
```
% javac -processor NullnessChecker MyFile.java

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

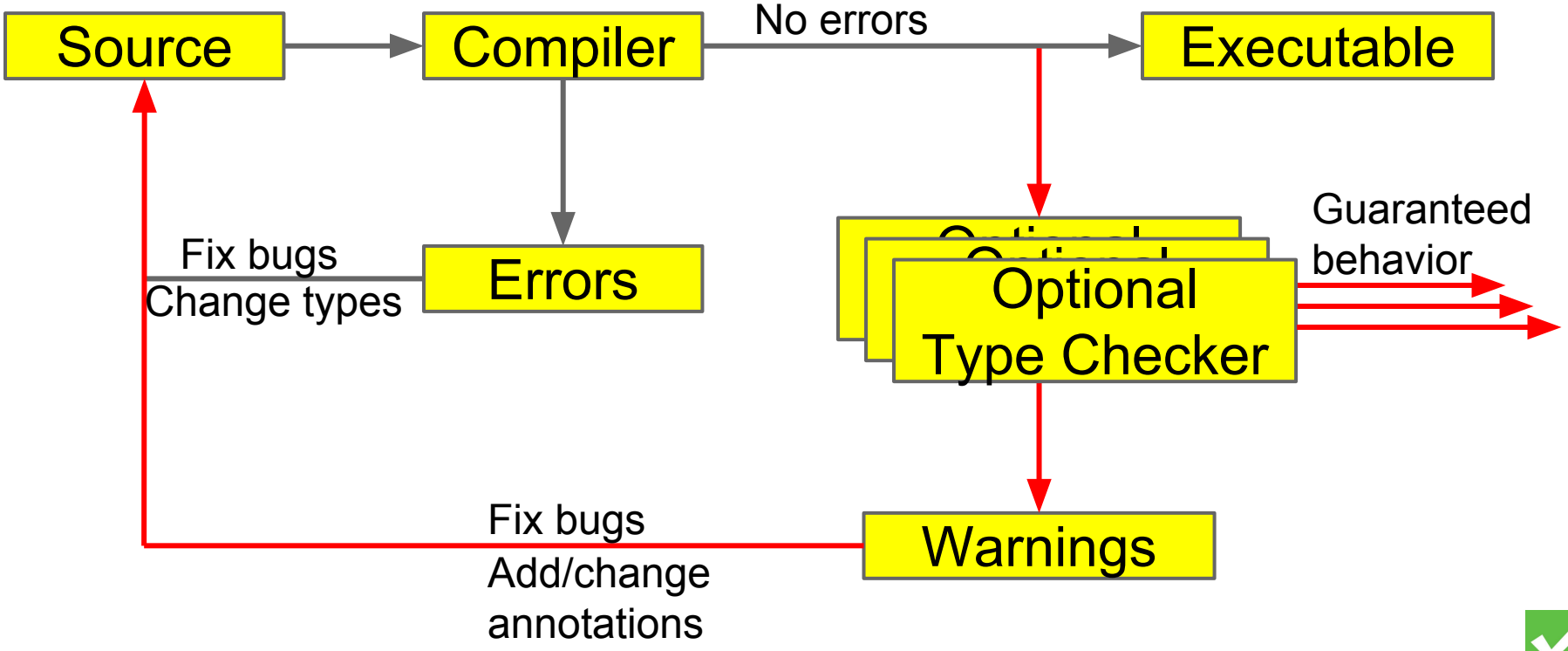
Type Checking



Optional Type Checking



Optional Type Checking



The Checker Framework

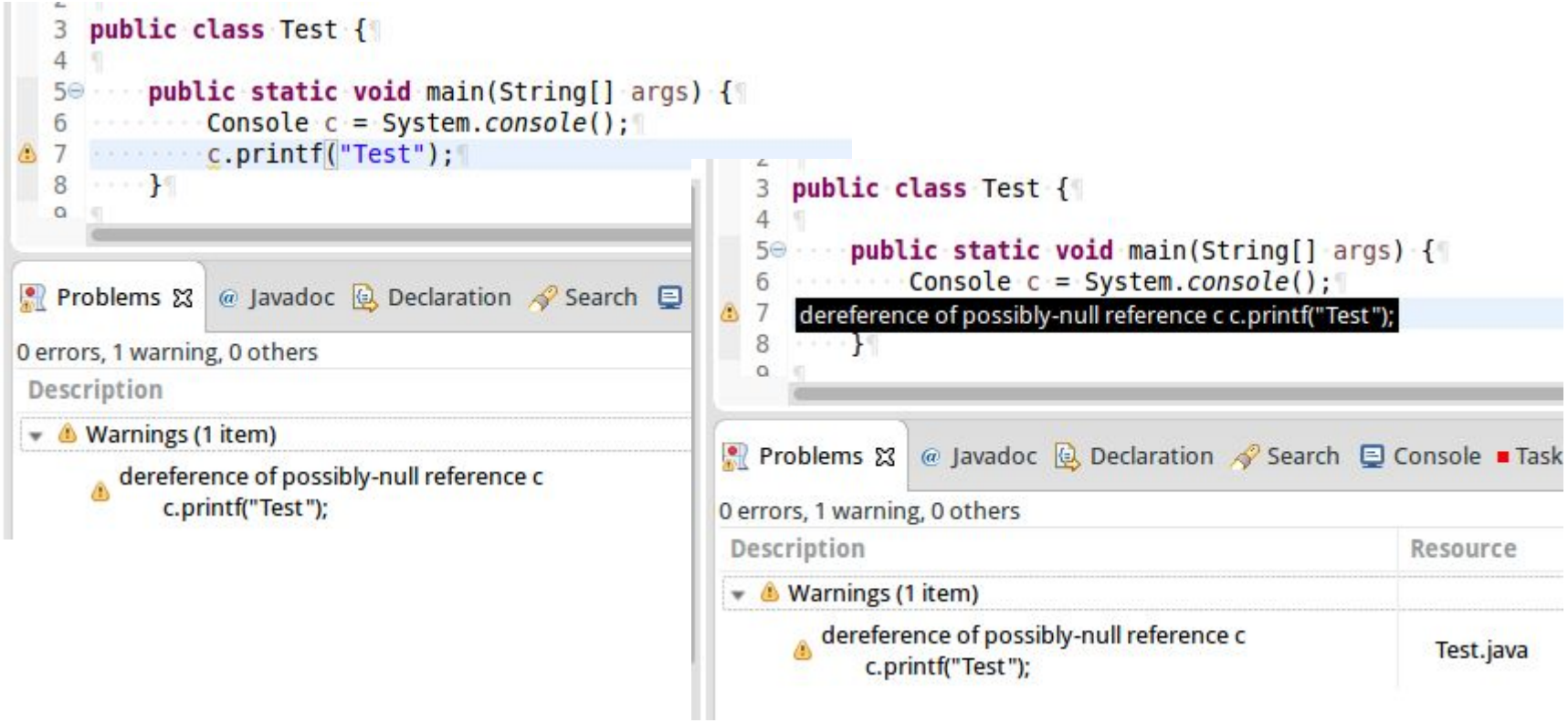
A framework for pluggable type checkers

“Plugs” into the OpenJDK or OracleJDK compiler

```
javac -processor MyChecker ...
```

Standard error format allows tool integration

Eclipse, IntelliJ, NetBeans plug-ins



Ant, Maven, Gradle 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>
```

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

Checker Framework Live Demo

Write Java code here:

```
1  import org.checkerframework.checker.nullness.qual.Nullable;
2  class YourClassNameHere {
3      void foo(Object nn, @Nullable Object nbl) {
4          nn.toString(); // OK
5          nbl.toString(); // Error
6      }
7  }
```

Choose a type system: Nullness Checker ▼

Check

Examples:

Nullness: NullnessExample | NullnessExampleWithWarnings

MapKey: MapKeyExampleWithWarnings

Interning: InterningExample | InterningExampleWithWarnings

Lock: GuardedByExampleWithWarnings | HoldingExampleWithWarnings | EnsuresLockHeldExample | Locl

Demo: regular expression errors

@Regex = valid regular expression

"colou?r"

NOT: "1) first point"

@Regex(2) = has 2+ capturing groups

"((Linked)?Hash)?Map"

"(http|ftp)://([^/]+)(/.*)?"

NOT: "(brown|beige)"

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
   String conten
   Pattern pat = Pattern.compile(regex);
   Matcher mat =
   if (mat.match
      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);
}
...
```

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)

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 use in Silicon Valley, on Wall Street, etc.

Example type systems

See our talk **“Preventing null pointer exceptions at compile time”** tomorrow at 10:20 in room GB 220A

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 / typedefs (`@Fenum`)

problems in Swing, JabRef

String type systems

Regular expression syntax (`@Regex`)

56 errors in Apache, etc.; 200 annos required

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

Information flow privacy (`@Source`)

- SPARTA detected malware in Android apps

You can write your own checker!
The Checker Framework makes it easy.

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.

Checker Framework Community

Open source project:

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

- Monthly release cycle
- 12,000 commits, 75 authors
- 40 issues closed since January 1, 2017
- Welcoming & responsive community

Pluggable type-checking improves code

Checker Framework for creating type checkers

- Featureful, effective, easy to use, scalable

Prevent bugs at compile time

Create custom type-checkers

Improve your code!

Learn more Wednesday
at 10:20 in room GB 220A:

Preventing null pointer exceptions at compile time

`http://CheckerFramework.org/`