Software engineering and programmer productivity tools

Programming languages

Automated testing

Debuggers

IDEs

Type systems

Verification

Security

Preventing Errors Before They Happen

`http://CheckerFramework.org/`

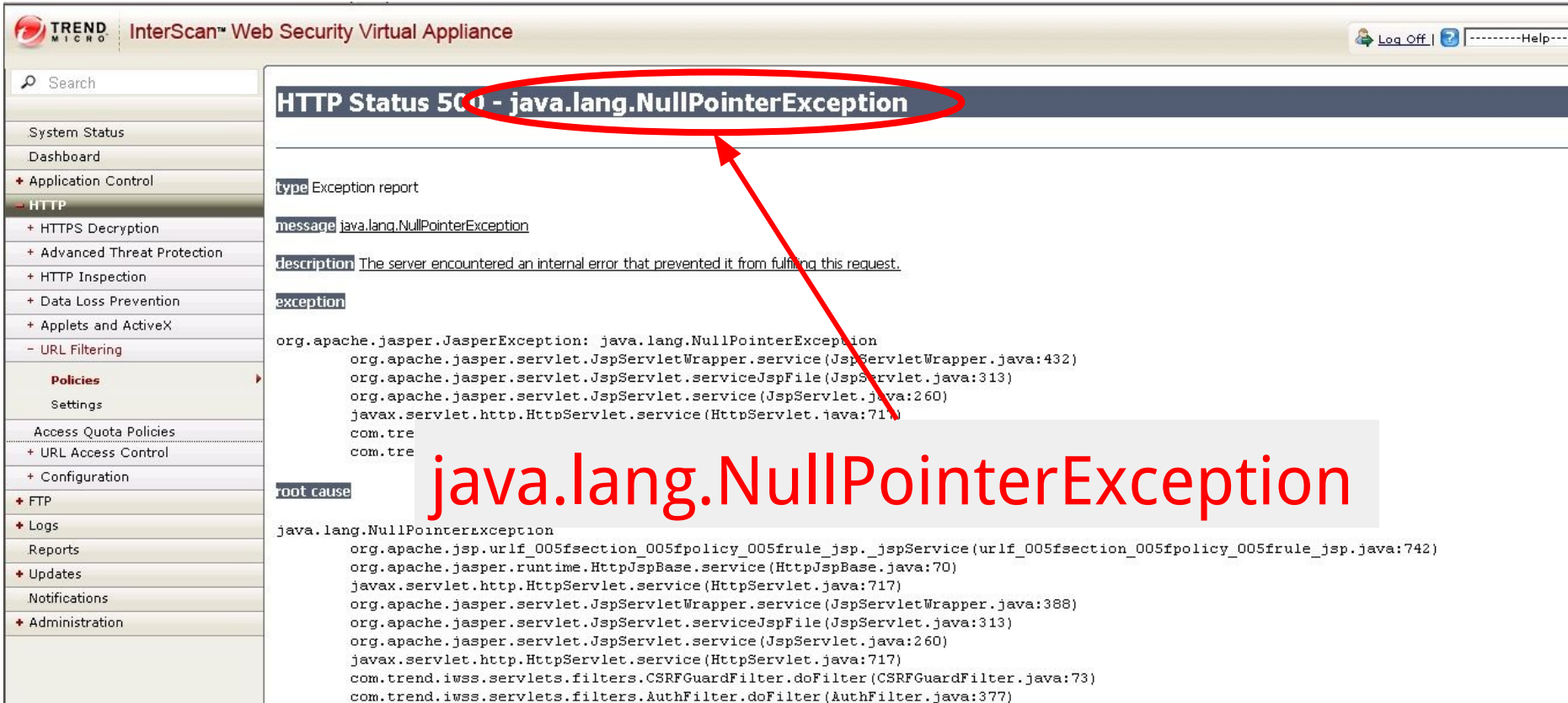
Twitter: @CheckerFrmwrk

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

Werner Dietl, University of Waterloo
Michael Ernst, University of Washington

UNIVERSITY OF WATERLOO

Motivation


TREND MICRO
InterScan™ Web Security Virtual Appliance
Log Off |
Help
Search
System Status
Dashboard
Application Control
HTTP
HTTPS Decryption
Advanced Threat Protection
HTTP Inspection
Data Loss Prevention
Applets and ActiveX
URL Filtering
Policies
Settings
Access Quota Policies
URL Access Control
Configuration
FTP
Logs
Reports
Updates
Notifications
Administration
HTTP Status 500 - java.lang.NullPointerException
type Exception report
message java.lang.NullPointerException
description The server encountered an internal error that prevented it from fulfilling this request.
exception
org.apache.jasper.JasperException: java.lang.NullPointerException
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:432)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:313)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:260)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
com.tre
com.tre
root cause
java.lang.NullPointerException
org.apache.jsp.urlf_005fsection_005fpolicy_005frule_jsp._jspService(urlf_005fsection_005fpolicy_005frule_jsp.java:742)
org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:388)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:313)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:260)
javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
com.trend.iwss.servlets.filters.CSRFGuardFilter.doFilter(CSRFGuardFilter.java:73)
com.trend.iwss.servlets.filters.AuthFilter.doFilter(AuthFilter.java:377)
java.lang.NullPointerException

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();
```

```
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

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
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, …

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
```

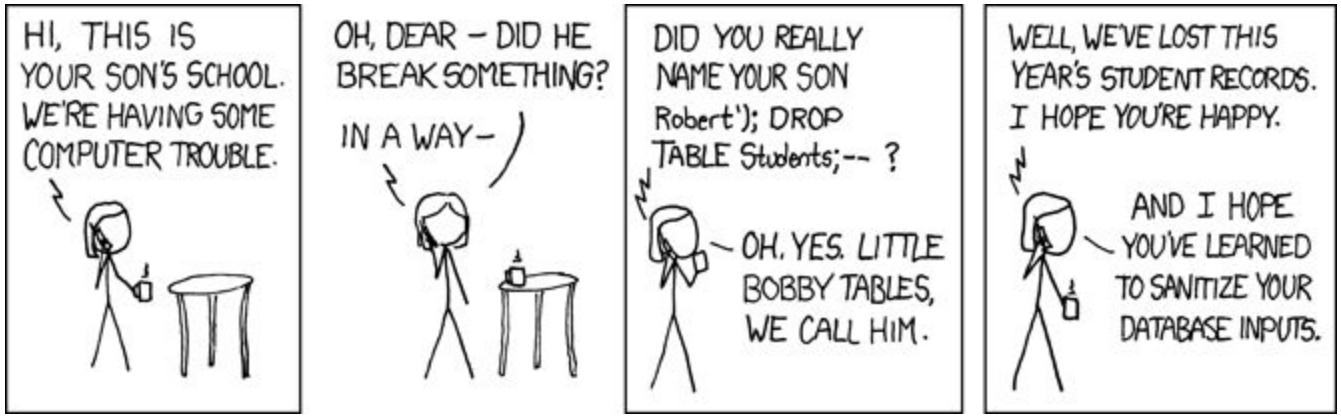
3. Type checker warns about violations (bugs)

```
% javac -processor NullnessChecker MyFile.java

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

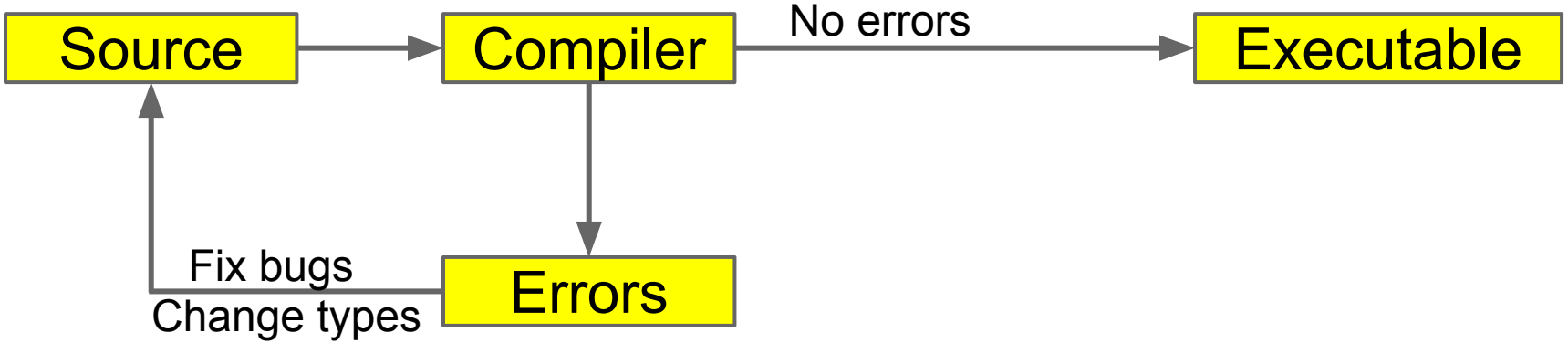
Nullness and encryption demo

- Detect errors
- Guarantee the absence of errors
- Verify the correctness of optimizations

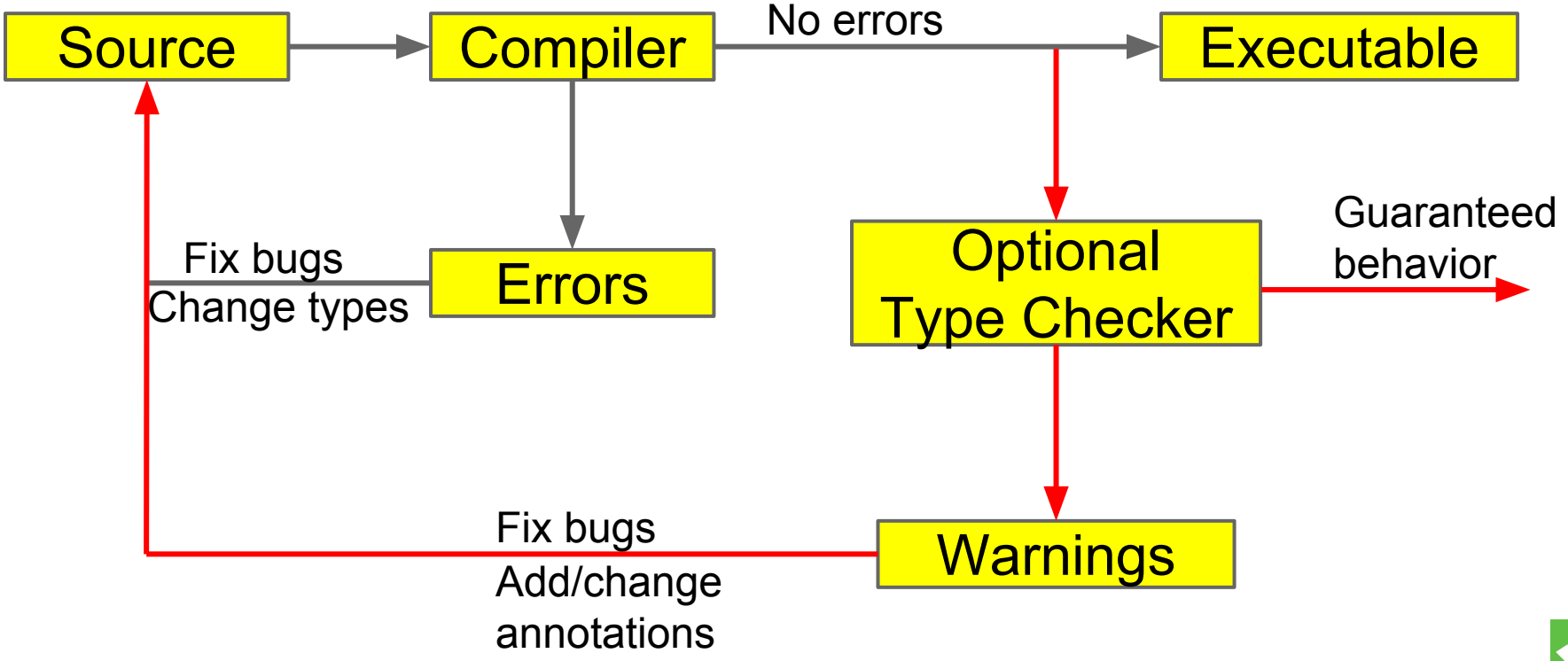


https://xkcd.com/327/

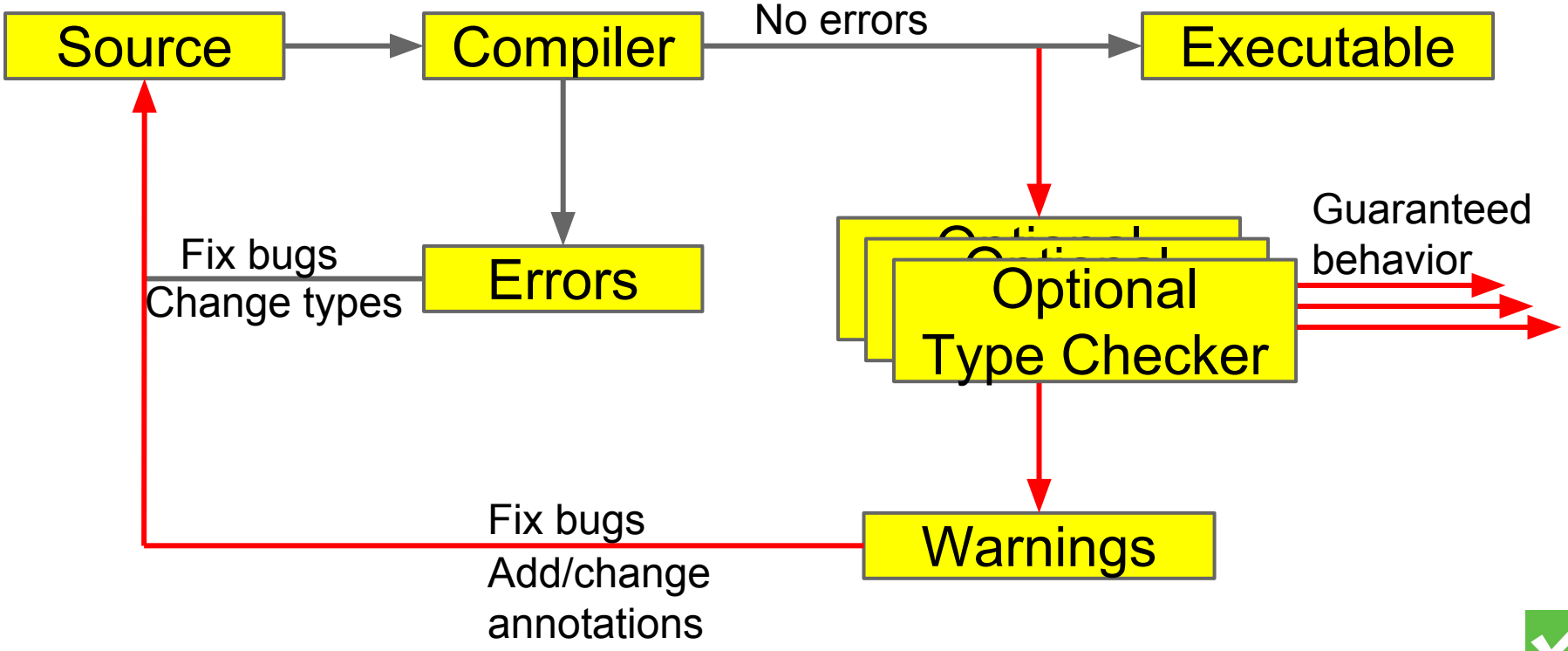
Type Checking



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);
```

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)

The Checker Framework

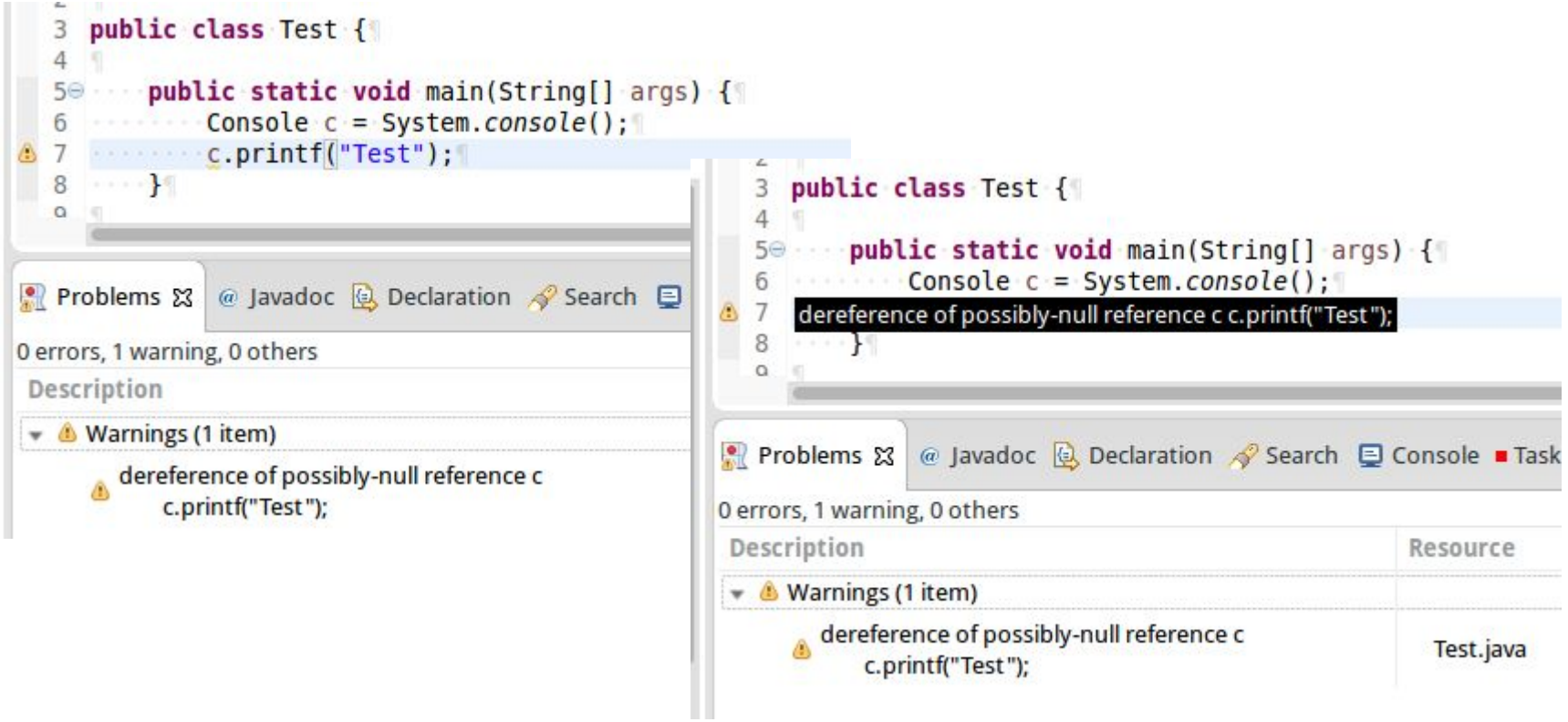
A framework for pluggable type checkers

“Plugs” into the OpenJDK or OracleJDK compiler

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

Standard error format allows tool integration

Eclipse plug-in

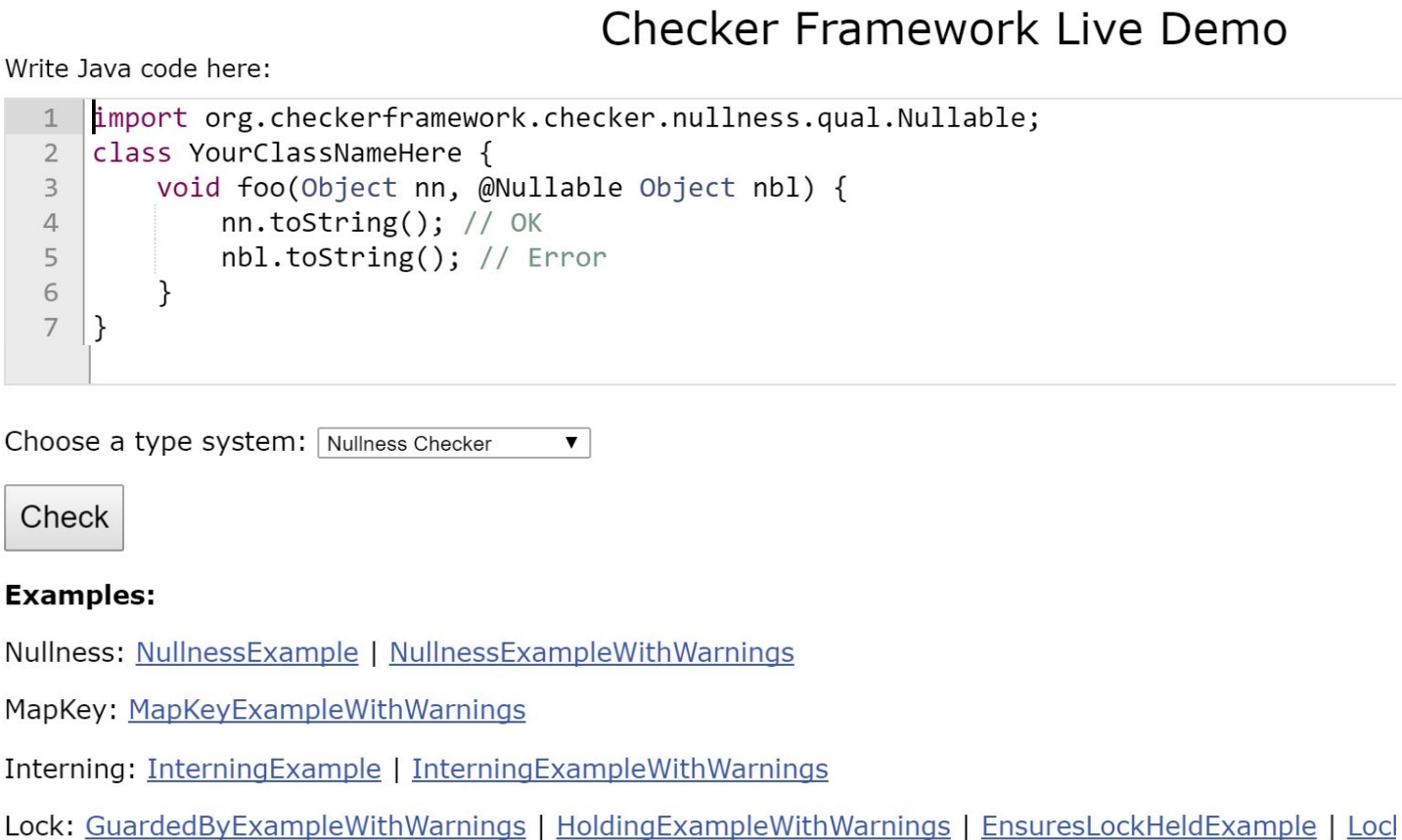


Ant and Maven integration

```
<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>
```

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



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 / 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!

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 (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

$$
\begin{array}{rcll}
\mathtt{P} & \in & \text{Program} & ::= \ \overline{\mathtt{Class}},\ \mathtt{ClassId},\ \mathtt{Expr} \\
\mathtt{Cls} & \in & \text{Class} & ::= \ \mathtt{class\ ClassId}\langle\overline{\mathtt{TVarId}}\rangle \\
 & & & \quad \mathtt{extends\ ClassId}\langle\overline{{}^{\mathtt{s}}\mathtt{Type}}\rangle \\
 & & & \quad \{\ \overline{\mathtt{FieldId}\ {}^{\mathtt{s}}\mathtt{Type};}\ \overline{\mathtt{Met}} \\
{}^{\mathtt{s}}\mathtt{T} & \in & {}^{\mathtt{s}}\text{Type} & ::= \ {}^{\mathtt{s}}\mathtt{NType} \mid \mathtt{TVarId} \\
{}^{\mathtt{s}}\mathtt{N} & \in & {}^{\mathtt{s}}\text{NType} & ::= \ \mathtt{OM\ ClassId}\langle\overline{{}^{\mathtt{s}}\mathtt{Type}}\rangle \\
\mathtt{u} & \in & \text{OM} & ::= \\
\mathtt{mt} & \in & \text{Meth} & ::= \\
 & & \mathtt{MethSig} & ::= \\
\mathtt{w} & \in & \text{Purity} & ::= \\
\mathtt{e} & \in & \text{Expr} & ::= \\
 & & & \quad \mathtt{Expr.MethId}\langle{}^{\mathtt{s}}\mathtt{Type}\rangle\mathtt{(Expr)} \mid \\
 & & & \quad \mathtt{new}\ {}^{\mathtt{s}}\mathtt{Type} \mid \mathtt{(}{}^{\mathtt{s}}\mathtt{Type}\mathtt{)\ Expr} \\
{}^{\mathtt{s}}\Gamma & \in & {}^{\mathtt{s}}\text{Env} & ::= \ \overline{\mathtt{TVarId}\ {}^{\mathtt{s}}\mathtt{NType};}\ \overline{\mathtt{ParId}\ {}^{\mathtt{s}}\mathtt{Type}}
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{h} & \in & \text{Heap} & = \ \mathtt{Addr} \to \mathtt{Obj} \\
\iota & \in & \text{Addr} & = \ \text{Set of Addresses} \cup \{\mathtt{null}_a\} \\
\mathtt{o} & \in & \text{Obj} & = \ {}^{\mathtt{r}}\mathtt{Type},\ \mathtt{Fields} \\
{}^{\mathtt{r}}\mathtt{T} & \in & {}^{\mathtt{r}}\text{Type} & = \ \mathtt{OwnerAddr\ ClassId}\langle\overline{{}^{\mathtt{r}}\mathtt{Type}}\rangle \\
\mathtt{Fs} & \in & \text{Fields} & = \ \mathtt{FieldId} \to \mathtt{Addr} \\
\iota & \in & \text{OwnerAddr} & = \ \mathtt{Addr} \cup \{\mathtt{any}_a\} \\
{}^{\mathtt{r}}\Gamma & \in & {}^{\mathtt{r}}\text{Env} & = \ \overline{\mathtt{TVarId}\ {}^{\mathtt{r}}\mathtt{Type};}\ \overline{\mathtt{ParId\ Addr}}
\end{array}
$$

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

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

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

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

$$
\text{DYN}\ \frac{\begin{array}{l}\mathtt{h} \vdash {}^{\mathtt{r}}\Gamma : {}^{\mathtt{s}}\Gamma \\ \mathtt{h} \vdash \iota_1 : dyn({}^{\mathtt{s}}\mathtt{N}, \mathtt{h}, {}^{\mathtt{r}}\iota_1) \\ \mathtt{h} \vdash \iota_2 : dyn({}^{\mathtt{s}}\mathtt{T}, \iota_1, \mathtt{h}(\iota_1)\downarrow_1) \\ {}^{\mathtt{s}}\mathtt{N} = \mathtt{u}_N\ \mathtt{C}_N\langle\_\rangle \\ \mathtt{u}_N = \mathtt{this}_u \Rightarrow {}^{\mathtt{r}}\Gamma(\mathtt{this}) \\ free({}^{\mathtt{s}}\mathtt{T}) \subseteq dom(\mathtt{C}_N)\end{array}}{\left.\begin{array}{l}\\ \\ \\ \\ \\ \\ {}^{\mathtt{r}}\mathtt{T} = \iota'\ \_\langle\_\rangle \qquad \iota \vdash {}^{\mathtt{r}}\mathtt{T}\ {}^{\mathtt{r}}{<:}\ \iota'\ \mathtt{C}\langle\overline{{}^{\mathtt{r}}\mathtt{T}}\rangle \qquad \iota \vdash {}^{\mathtt{r}}\mathtt{T}\ {}^{\mathtt{r}}{<:}\ \iota'\ \mathtt{C}\langle\overline{{}^{\mathtt{r}}\mathtt{T}_a}\rangle \Rightarrow \iota \vdash \overline{{}^{\mathtt{r}}\mathtt{T}}\ {}^{\mathtt{r}}{<:}\ \overline{{}^{\mathtt{r}}\mathtt{T}_a} \\ \mathrm{dom}(\mathtt{C}) = \overline{\mathtt{X}} \qquad \mathrm{free}({}^{\mathtt{s}}\mathtt{T}) \subseteq \overline{\mathtt{X}} \circ \overline{\mathtt{X}'}\end{array}\right\} \implies \mathtt{h} \vdash \iota_2 : dyn({}^{\mathtt{s}}\mathtt{N} \rhd {}^{\mathtt{s}}\mathtt{T}, \mathtt{h}, {}^{\mathtt{r}}\Gamma)}{dyn({}^{\mathtt{s}}\mathtt{T}, \iota, {}^{\mathtt{r}}\mathtt{T}, (\overline{\mathtt{X}'\ {}^{\mathtt{r}}\mathtt{T}'}; \_)) = {}^{\mathtt{s}}\mathtt{T}[\iota'/\mathtt{this}, \iota'/\mathtt{peer}, \iota/\mathtt{rep}, \mathtt{any}_a/\mathtt{any}_u, \overline{{}^{\mathtt{r}}\mathtt{T}/\mathtt{X}}, \overline{{}^{\mathtt{r}}\mathtt{T}'/\mathtt{X}'}]}
$$

Tips

- Start by type-checking part of your code
- Only type-check properties that matter to you
- Use subclasses (not type qualifiers) if possible
- Write the spec first (and think of it as a spec)
- Avoid warning suppressions when possible
- Avoid raw types such as `List`; use `List<String>`

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.

Community

Open source project:

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

- Monthly release cycle
- 11,000 commits, 75 authors

Issue tracker:

- 110 issues closed in releases June 1 - Sep 16

Mailing lists:

- to reach developers
- to reach whole 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!

`http://CheckerFramework.org/`

Why get involved in research?

Unique experience available only at college

Deeper understanding of computing

Close collaboration with faculty, grad students

Contribute to science

Fame (4 single-authored papers on undergrad research, at FSE 2016)

Fun!

How to get involved in PLSE research

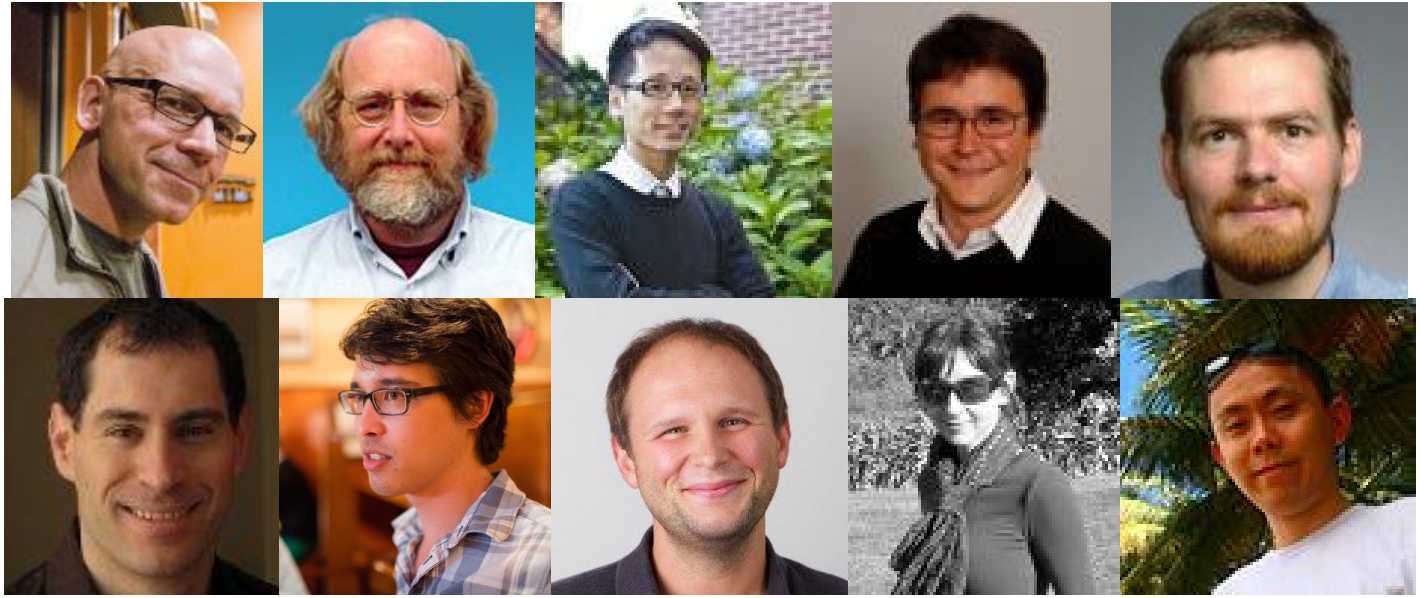
Take CSE 331 (“software design & implementation”)
Contact faculty in the PLSE group:

Rastislav Bodik
Alan Borning
Luis Ceze
Alvin Cheung
Michael Ernst
Daniel Grossman
Andrew Ko
Zachary Tatlock
Emina Torlak
Xi Wang

How to get involved in PLSE research

Take CSE 331 (“software design & implementation”)

Contact faculty in the PLSE group:



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
   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);
}
...
```