A type system for regular expressions

Eric Spishak
Werner Dietl
Michael Ernst

http://types.cs.washington.edu/

University of Washington
Computer Science & Engineering

Regular Expressions

`(.*) ([0-9]+)`

- Used to:
  - Match text
  - Extract from text
- Represented as sequence of characters
- Easy to make mistakes
  - Syntax is complex
  - Differences in syntax/features across programming languages

Regular Expressions in Java

```
Pattern p = Pattern.compile("(.*) ([0-9]+)");
Matcher m = p.matcher("number 447");
if (m.matches()) {
    System.out.println(m.group(2));
}
```

Regular Expressions in Java

PatternSyntaxException

```
Pattern p = Pattern.compile("(.*) ([0-9]+)(");
Matcher m = p.matcher("number 447");
if (m.matches()) {
    System.out.println(m.group(2));
}
```

Regular Expressions in Java

```
Pattern p = Pattern.compile("(.*) ([0-9]+)");
Matcher m = p.matcher("number 447");
if (m.matches()) {
    System.out.println(m.group(3));
}
```

IndexOutOfBoundsException

Regular Expressions in Java

PatternSyntaxException

```
Pattern p = Pattern.compile("(.*) ([0-9]+)(");
Matcher m = p.matcher("number 447");
if (m.matches()) {
    System.out.println(m.group(3));
}
```

IndexOutOfBoundsException

Could these errors be caught at compile time instead of at run time?

Outline

- Regular Expressions in Java
- Regular Expression Type System
- Capturing Groups
- Case Studies
- Conclusions

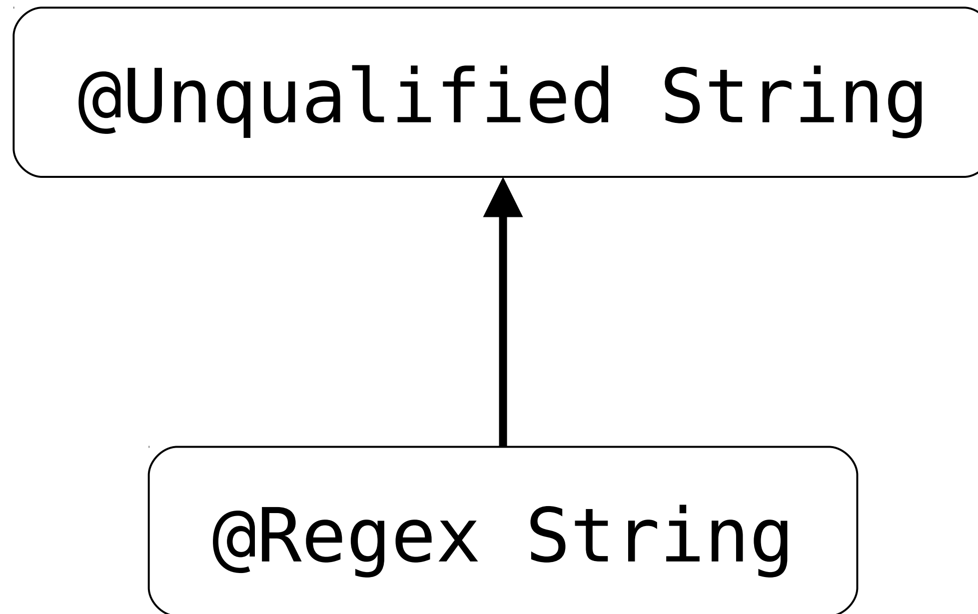
Goal and Approach

- At compile time, prevent `PatternSyntaxExceptions` and `IndexOutOfBoundsExceptions`
- Pluggable type system for Java 8,
using the Checker Framework
- @Regex annotation qualifies Strings that are valid regular expressions

Example from class `java.util.regex.Pattern`:

```
Pattern compile(@Regex String regex) { … }
```

Regex Qualifier Hierarchy



APIs that take `@Unqualified Strings` are unaffected

@Regex Annotation Example

```
@Regex String regex = "(regex)*";
```

@Regex Annotation Example

```
@Regex String regex = "(regex)*";
```

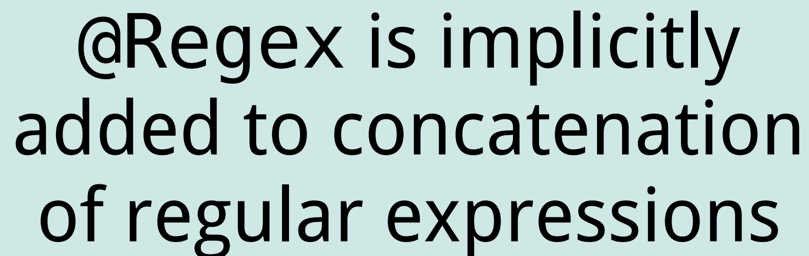
@Regex is implicitly added to String literals that are regular expressions

@Regex Annotation Example

```
@Regex String regex = "(regex)*";
@Regex String regex2 = regex + "(re)";
```

@Regex Annotation Example

```
@Regex String regex = "(regex)*";
@Regex String regex2 = regex + "(re)";
```

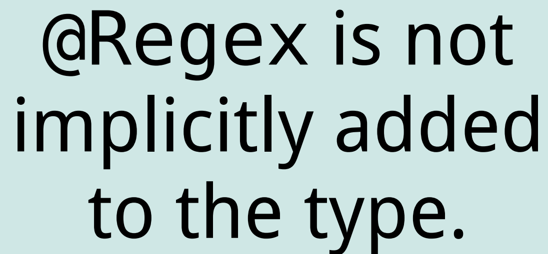


@Regex Annotation Example

```
@Regex String regex = "(regex)*";
@Regex String regex2 = regex + "(re)";
@Regex String nonRegex = "non (";
       // compile-time error
```

@Regex Annotation Example

```
@Regex String regex = "(regex)*";
@Regex String regex2 = regex + "(re)";
       String nonRegex = "non (";
```



@Regex Annotation Example

```
@Regex String regex = "(regex)*";
@Regex String regex2 = regex + "(re)";
       String nonRegex = "non (";

       Pattern.compile(regex);
       Pattern.compile(regex2);
       Pattern.compile(nonRegex);
           // compile-time error
```

Flow-sensitive Inference

```
String regex = getRegexFromUser();
if (!RegexUtil.isRegex(regex)) {
    System.err.println("error parsing " +
           regex +
           RegexUtil.regexException(regex));
    System.exit(99);
}
// Type of regex refined to @Regex String
Pattern p = Pattern.compile(regex);
```

Outline

- Regular Expressions in Java
- Regular Expression Type System
- Capturing Groups
- Case Studies
- Conclusions

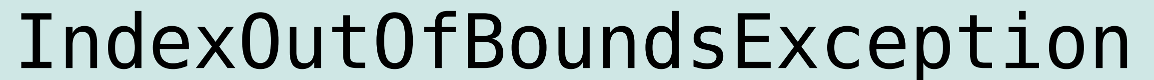
Capturing Groups

Can wrap capturing groups in parentheses to extract text

```
Pattern p = Pattern.compile("(.*) ([0-9]+)");
Matcher m = p.matcher("number 447");
if (m.matches()) {
    System.out.println(m.group(2)); // output: 447
}
```

Capturing Groups - Errors

```
Pattern p = Pattern.compile("(.*) ([0-9]+)");
Matcher m = p.matcher("number 447");
if (m.matches()) {
    System.out.println(m.group(3));
}
```



@Regex(*groups*) Annotation

- @Regex(n) has at least n groups

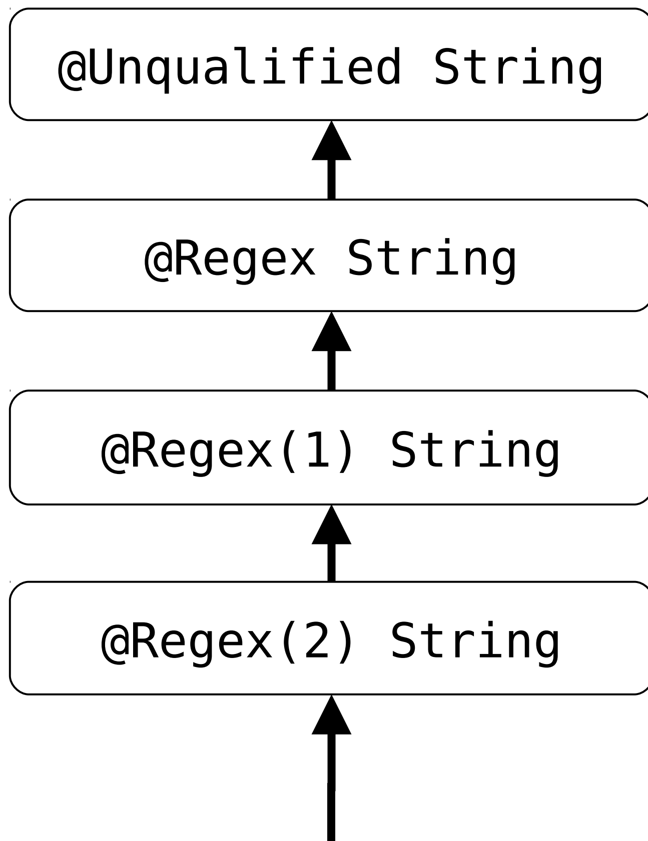
```
@Regex(2) String s = "(reg)(ex)";
@Regex(4) String s2 = "(regex)"; // compile-time error
```

@Regex(*groups*) Annotation

- `@Regex(n)` has at least n groups
- Group counts inferred from `String` literals and `String` concatenation.

```
@Regex(2) String s = "(reg)(ex)";
@Regex(1) String s2 = "(regex)";
@Regex(3) String s3 = s + "(regex)";
```

Regex Qualifier Hierarchy




```
@Regex(5) String s5 = null;
@Regex(3) String s3 = "(r)(e)(g)";
foo(s3);

…

void foo(@Regex(1) String s1) {
    …
}
```

Verifying Group Counts

`@Regex(`*`groups`*`)` annotation applicable to `Pattern` and `Matcher` classes.

Information is propagated from String to Pattern to Matcher.

```
@Regex(2) String s = "(reg)(ex)";
@Regex(2) Pattern p = Pattern.compile(s);
@Regex(2) Matcher m = p.matcher("regex");
```

Verifying Group Counts

Annotations are added implicitly.

```
@Regex(2) String s = "(reg)(ex)";
@Regex(2) Pattern p = Pattern.compile(s);
@Regex(2) Matcher m = p.matcher("regex");
```

Verifying Group Counts

```
@Regex(2) String s = "(reg)(ex)";
@Regex(2) Pattern p = Pattern.compile(s);
@Regex(2) Matcher m = p.matcher("regex");
          if (m.matches()) {
              m.group(2);

          }
```

Verifying Group Counts

```
@Regex(2) String s = "(reg)(ex)";
@Regex(2) Pattern p = Pattern.compile(s);
@Regex(2) Matcher m = p.matcher("regex");
          if (m.matches()) {
              m.group(2);
              m.group(3); // compile-time error
          }
```

Outline

- Regular Expressions in Java
- Regular Expression Type System
- Capturing Groups
- Case Studies
- Conclusions

API Annotations

Added 18 annotations total

```
class Pattern {
    public static Pattern compile(@Regex String regex);
    public static @Regex String quote(String s);
}

class String {
    public boolean matches(@Regex String regex);
    public String replaceFirst(@Regex String regex,
                               String replacement);
    public String replaceAll(@Regex String regex,
                             String replacement);
    public String[] split(@Regex String regex);
}
```

Case Studies - Results

Project	LOC	@Regex Annotations	Bugs	False Positives
Plume-lib	13k	40	2	9
Daikon	166k	8	6	1
Lucene	154k	11	6	5
Chukwa	37k	45	7	3
YaCy	112k	71	35	7
TOTALS	482k	175	56	25

False Positives

8	StringBuilder/StringBuffer/char[]
8	tests whether s is a regex
3	substring operation
2	Variable group count
1	character class in a regex
1	flow-sensitivity bug
1	line.separator property is a legal regex
1	Deprecated code
25	Total

50 Bugs – No Input Validation

From Table_API_p in YaCy:

```
String inputquery = post.get("query", "");
```

```
Pattern.compile(".*" + inputquery + ".*");
```

50 Bugs – No Input Validation – Fix

From Table_API_p in YaCy:

```
String inputquery = post.get("query", "");
if (!RegexUtil.isRegex(inputquery)) {
    throw new Error("error parsing " +
               inputquery + ": " +
               RegexUtil.regexError(inputquery));
}
Pattern.compile(".*" + inputquery + ".*");
```

6 Bugs – Missed Quoting

From Load_RSS_p in YaCy:

```
String messageurl;
...
messageurl = row.get("url", "");

if (r == null ||
    !r.get("comment", "")
      .matches(".*\\Q" + messageurl + "\\E.*")) {
    …
}
```

6 Bugs – Missed Quoting – Fix

From plume-lib/java/src/plume/Lookup.java:

```
String messageurl;
...
messageurl = row.get("url", "");
messageurl = Pattern.quote(messageurl);
if (r == null ||
    !r.get("comment", "")
      .matches(".*\\Q" + messageurl + "\\E.*")) {
    …
}
```

Case Study Experience

- Easy to use
  - Regex Checker points out places to get started
  - Short amount of time to annotate
- The difficult sections were ugly code
  - Using Regex Checker from the beginning of a project would result in cleaner code

The Checker Framework

1. Publicly available
2. Comes with > 12 type systems, including
   1. Non-null types
   2. Immutability
   3. Fake enumerations
   4. Regular expressions

http://types.cs.washington.edu/

Regex Checker Implementation

1. 605 non-comment, non-blank lines
2. Simple type system easy to implement
3. Interesting code for flow-sensitivity and special casing

Checker Framework Tutorial

Do you want to learn how to build your own pluggable type systems?

Come see my PLDI tutorial!

Saturday, 16 June from 9:00 to 12:00

Conference 9

Conclusions

- Worth creating a new type system
  - Found 56 real bugs
- Easy to do
- Overhead is low
- Improves documentation
- Encourages cleaner code

Come to the Checker Framework tutorial
on Saturday at 9am!