Tool Support for Annotations

Rob Harwood, Technology Evangelist

JetBrains

What are Annotations?

- Annotations are used to mark a declaration as special:
  - Classes
  - Methods
  - Parameters
  - Variables
  - Etc.

```
public class PeopleTests {
    @Test
    public void something() {
        PeopleImpl people = new PeopleImpl();
```

What are Annotations?

- Can have parameters to specify details:
  - String
  - Number
  - Enumeration
  - Etc.

```
public class Client {
    public static void main(String[] args) {
        @Happy(face = "hello") String x = "";
    }
}
```

What for?

- Three retention models
  1. Runtime
  2. Class
  3. Source

What for?

- Two primary usages
  1. Dynamic (and also static)
     - Runtime
  2. Static (only)
     - Class
     - Source

What for?

- Static annotations can be used to aid static analysis.
  - IDE support
    - Basic
    - Advanced
  - Other static tools
    - FindBugs

What for?

- Static analysis has proven invaluable for improved productivity
  - All IDEs have at least basic static analysis to highlight syntax errors in code
  - More advanced analysis makes the IDE much smarter
  - By showing analysis “on-the-fly”, the IDE makes YOU much smarter

Examples of Static Analysis

- Common coding errors
- Security issues
- Poor threading practices
- Poor object-oriented design
- Replace for(; ;) loop with for(:)
- Technology-specific (EJB, JavaScript, Ant, etc.)
- Hundreds more!

Basic IDE Support for Annotations

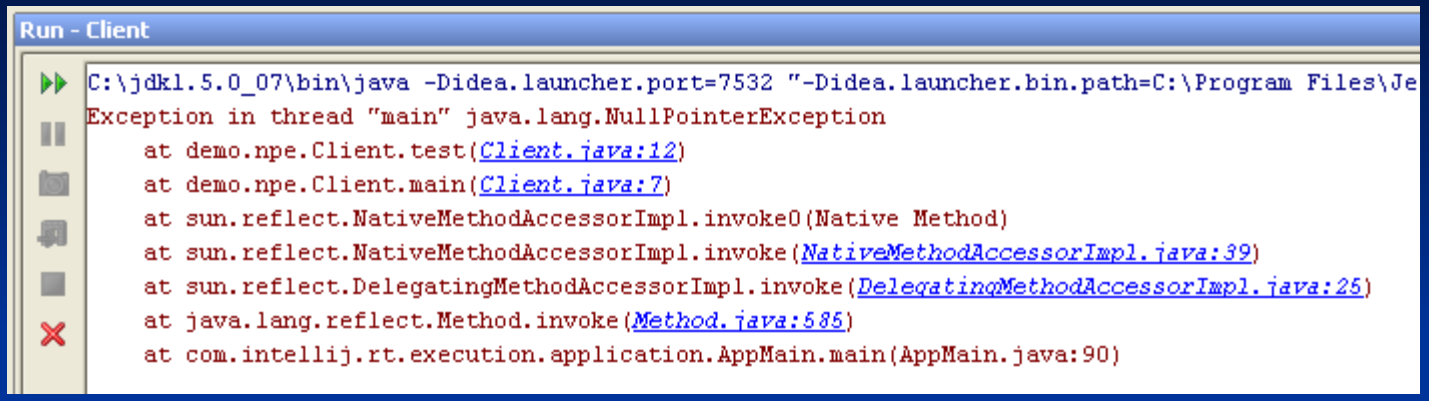
- Coloring
- Syntax checking
- Code completion
- Find usages
- Refactoring

Advanced IDE Support

- Use special annotations to make the IDE smarter
  - NullPointerException (NPE) analysis
  - Property file key analysis
  - Localization analysis

NPE Analysis demo

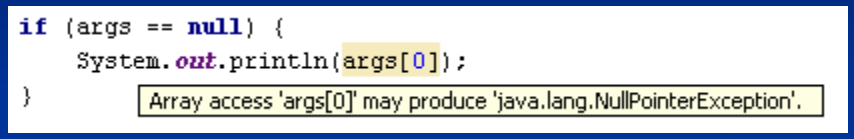
NPE Stacktraces

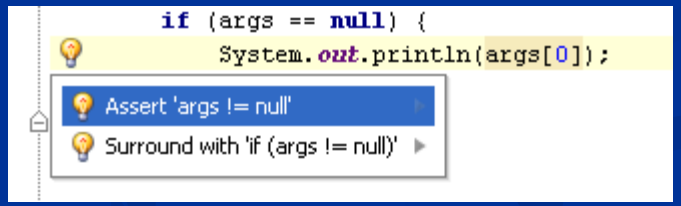


- Stacktrace: Requires 3 mental contexts
  1. Runtime state: Running the program/tests
  2. Server code: Investigate dereference for logic errors
  3. Client code: Find source of null value
- Switching contexts takes TIME
- Eliminates “flow”

Simple detection, no annotations

- Quickly detects a problem
- Displays a warning
- Gives options for fixing automatically





- No context switching!
- No breaking of “flow”!

Limitations of Simple Detection

- Sometimes, you just don't know whether a null is okay or not.

```
// Is null allowed?
check(null, null);
```

```
private void check(String columnName, String rowName) {
    // Cannot tell if null is okay from method signature.
    // Long and complicated method...

    // Passing null for columnName would cause a NPE
    if (columnName.equals(rowName)) {
        doSomething();
    }
}
```

- The IDE cannot guess the intended design without a hint from the programmer

Use Annotations

- Use annotations to mark @NotNull or @Nullable

```
// Annotated with @NotNull
checkAnnotated(null, null);
```

```
private void checkAnnotated(@NotNull String columnName, @Nullable String rowName) {
    if (columnName.equals(rowName)) {
        doSomething();
    }
}
```

- Now IDE is able to give smart feedback

NPE Annotations

- @NotNull: Null is an error
- @Nullable: Check for null before using
- No annotation: No analysis

- Applies to: Method, field, parameter, local variable

Why is this important?

- Quality
  - Detect MORE errors
- Productivity
  - Detect errors FASTER
  - Fix many errors AUTOMATICALLY (Alt-Enter)
  - Stay in “the flow”, no context switching
- Interface documentation
  - Teamwork (separation of responsibility)
  - Communication

Property File Key Analysis

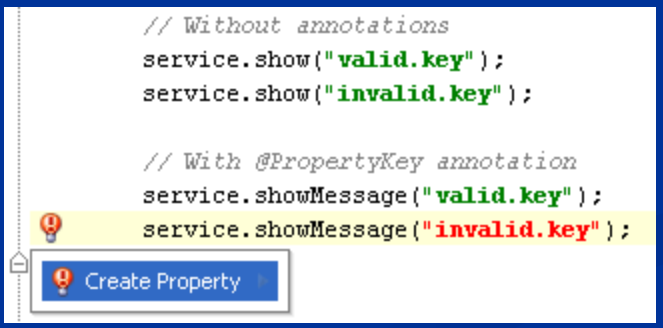
- Properties in properties files are just text, so the IDE can only do limited validation
- @PropertyKey annotation
  - Detect invalid keys
  - Provide automatic fix

```
private static final String BUNDLE = "demo.properties.messages";

public void show(String messageKey) {
    ResourceBundle bundle = ResourceBundle.getBundle(BUNDLE);
```

```
        // Without annotations
        service.show("valid.key");
        service.show("invalid.key");

        // With @PropertyKey annotation
        service.showMessage("valid.key");
        service.showMessage("invalid.key");
```



Localization Analysis

- Large projects often need to be internationalized for different locales
- Output to the user should be localized, but other Strings can be hard-coded
- But Strings all look the same
- How to tell the difference?

```
private void doSomething() {
    debug("In doSomething method");
    if (userLoggedIn()){
        sendUser("Welcome!");
    }
    sendUser(getMessage("main.message"));
```

Localization Analysis

- "Hard coded strings" inspection

```
private void doSomething() {
    debug("In doSomething method");
    if (userLoggedIn()) {
        sendUser("Welcome!");
    }
    sendUser(getMessage("main.message"));
}
```

- Plus @NonNls (Non Natural Language String)

```
private void doSomething() {
    debug("In doSomething method");
    if (userLoggedIn()){
        sendUser("Welcome!");
    }
    sendUser(getMessage("main.message"));
}
```

- Can separate NLS vs. non-NLS strings

Custom Annotation-Support Plugins

- Custom resource reference annotations
  - Similar to @PropertyKey
  - Configuration files, icons, custom resources
- IntelliJ IDEA Open API
- Pay off for mid/large projects

Summary of IntelliJ IDEA's Annotations

- <install>/redist/annotations.jar
- Apache open source license
- @Nullable/@NotNull for detecting NPEs
- @NonNls/@Nls for detecting localization problems
- @PropertyKey for detecting property file key problems
- Full IDE support for best productivity

FindBugs

- Static code analysis
- Open Source
- Analyzes class files, batch processing
- Can run in IDE (NetBeans, Eclipse) or Ant
  - Not fully integrated into IDE
- Includes annotations for:
  - @CheckForNull, @NonNull, @Nullable
  - Similar but not the same as IntelliJ IDEA's

JSR 305: Annotations for Software Defect Detection

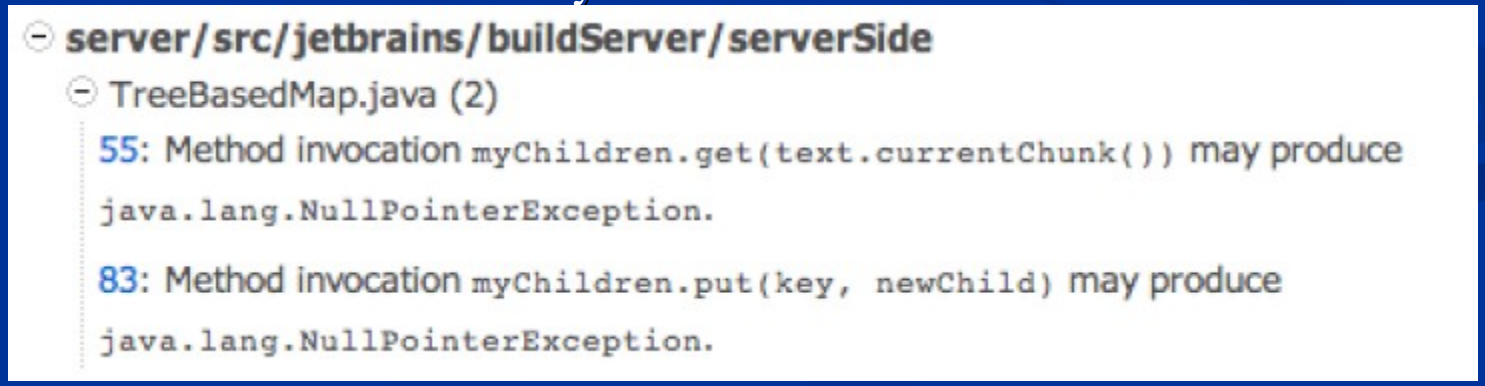
- New tool-oriented annotations for JDK
- "This JSR will work to develop standard annotations (such as @NonNull) that can be applied to Java programs to assist tools that detect software defects."
- Lead by Bill Pugh, author of FindBugs
  - JetBrains actively participating
- Ballot has been approved; under development

Proposed Annotations for JSR 305

- Nullness
- Check return values
- SQL-injection “taint”
- Concurrency
- Internationalization

Static analysis can be used in your build process

- Ant
  - IntelliJ IDEA
  - FindBugs
- TeamCity
  - New continuous-build, and collaboration server.
  - Also runs static analysis



IntelliLang

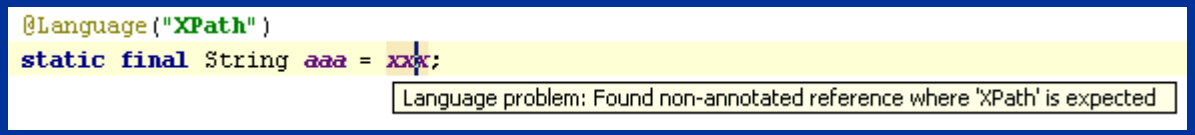
- Plugin and collection of annotations which supports 'language injection': Allows arbitrary languages to be inserted into Java strings

```
@Language(value = "JavaScript", prefix = "function doSomethingElse(a){ }")
String code = "function doSomething() {\n" +
"  var x = 1;\n" +
"  doSomethingElse(x);\n" +
"}";
```

- Examples: Regular Expressions, XML, XPath, JavaScript, DTD, etc.

IntelliLang

- Validates correctness of code embedded in strings and string variables



- Available from IntelliJ IDEA plugin repository
- Very new, so somewhat unfinished

Information

- http://jetbrains.com/idea/
- http://www.cs.umd.edu/~pugh/
- http://findbugs.sourceforge.net/
- http://jcp.org/en/jsr/detail?id=305