Annotation Improvements

Type Annotations allow developers to write annotations in more places than before. The compiler can then verify these annotations, for example identifying uses of null values, accidental value modifications, and cases where data crosses a trust boundary without proper validation.

The [Checker Framework](http://types.cs.washington.edu/checker-framework/current/checkers-manual.html) provides a few Type Annotations that could benefit both library and application developers, such as:

* [@NonNull](http://types.cs.washington.edu/checker-framework/current/checkers-manual.html#nullness-checker) – The compiler can determine cases where a code path might receive a null value, without ever having to debug a NullPointerException.
* [@ReadOnly](http://types.cs.washington.edu/checker-framework/current/checkers-manual.html#igj-checker) – The compiler will flag any attempt to change the object.  This is similar to Collections.unmodifiableList, but more general and verified at compile time.
* [@Regex](http://types.cs.washington.edu/checker-framework/current/checkers-manual.html#regex-checker) – Provides compile-time verification that a String intended to be used as a regular expression is a properly formatted regular expression.
* [@Tainted and @Untainted](http://types.cs.washington.edu/checker-framework/current/checkers-manual.html#tainting-checker) – Identity types of data that should not be used together, such as remote user input being used in system commands, or sensitive information in log streams.
* [@m](http://types.cs.washington.edu/checker-framework/current/checkers-manual.html#units-checker) – Units of measure ensures that numbers used for measuring objects are used and compared correctly, or have undergone the proper unit conversion.

Putting Type Annotations on your code

Java SE 8 allows type annotations anywhere that a type is used. Previously, annotations were only allowed on definitions. Some examples of this are:

|  |  |
| --- | --- |
| **ANNOTATION EXAMPLE** | **MEANING** |
| @NonNull List<String> | A non-null list of Strings. |
| List<@NonNull String> | A list of non-null Strings. |
| @Regex String validation = "(Java|JDK) [7,8]" | Check at compile time that this String is a valid regular expression. |
| private String getInput(String parameterName){ final String retval = @Tainted request.getParameter(parameterName);   return retval; } | The object assigned to retval is tainted and not for use in sensitive operations. |
| private void runCommand(@Untainted String… commands){ ProcessBuilder processBuilder = new ProcessBuilder(command);   Process process = processBuilder.start(); } | Each command must be untainted. For example, the previously tainted String must be validated before being passed in here. |

Type Annotations are best used with common forms of validation that relate to computer science. There are certain types of business validation that are not applicable.

|  |  |
| --- | --- |
| Well-suited for Type Annotations |  |
| * Null value checks. * Numeric range checks. * Basic type checks, such as regular expressions. * Assignments and updates (e.g. read-only) * Dataflow validation detection (e.g. have the incoming function arguments gone through the right validation functions) |  |