Compile-time Typechecking for Custom Java Type Qualifiers

Matt Papi, Michael Ernst
Program Analysis Group
MIT CSAIL

November 28, 2007

Example:

NonNull typechecker

Benefits of custom type qualifiers for Java

Type qualifiers can:

- guarantee the absence of certain errors
- help programmers find bugs
- provide clear, checkable documentation
- eliminate assertions and run-time checks

The demo

I will demonstrate the process of finding and fixing bugs using three typecheckers:

- NonNull/Nullable
- Interned
- Javari (reference immutability)

NonNull subject programs

The examples I am showing come from 3 real programs:

Program	**Lines**	**Annotations**	**Bugs found**
Lookup	3961	83	7
NonNull checker	1031	65	5
Checkers framework	5451	308	29

Examples:

Lookup, checkers framework

Comparison with other tools: Lookup

Lookup contained 7 null-related bugs.

Tool	**Warnings**	**Errors**
Our typechecker	0	7
FindBugs	1	0
JLint	0	0

Interning

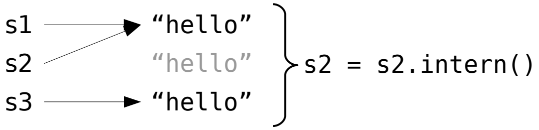
- Also known as canonicalization or hash-consing
- A space optimization: reuse an existing object instead of creating a new one
    - The space savings can be significant
- Built into `java.lang.String`: `intern()` method

$$\left.\begin{array}{l} \texttt{s1} \longrightarrow \text{“}\texttt{hello}\text{”} \\ \texttt{s2} \longrightarrow \text{“}\texttt{hello}\text{”} \\ \texttt{s3} \longrightarrow \text{“}\texttt{hello}\text{”} \end{array}\right\} \texttt{s1 = s1.intern()}$$

- Users can add interning for their own classes

Interning

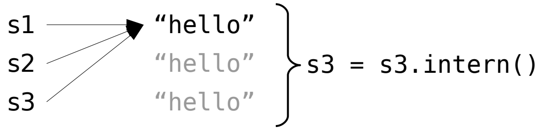
- Also known as canonicalization or hash-consing
- A space optimization: reuse an existing object instead of creating a new one
  - The space savings can be significant
- Built into `java.lang.String`: `intern()` method



- Users can add interning for their own classes

Interning

- Also known as canonicalization or hash-consing
- A space optimization: reuse an existing object instead of creating a new one
  - The space savings can be significant
- Built into `java.lang.String`: `intern()` method



- Users can add interning for their own classes

Interning (2)

- Interning also saves time: can compare with ==

```
myString.equals(otherString)
myString == otherString // and, this is more readable
                        // and emphasises the interning
```

- Potential for error: using == on non-interned objects

```
new Integer(22) == new Integer(22) // yields false!
```

- Benefits of automatic checking:
    - guarantee that no space savings were overlooked
    - guarantee of no equality-checking errors

Daikon invariant detector

- Memory is the limiting factor in scaleability
- Daikon makes extensive use of space optimizations such as interning
- 250KLOC of Java code
- 1200 lines of code/comments about interning
- 200 run-time assertions checking interning

Daikon case study

Added to Daikon:

- 127 @Interned annotations
    - Most files require no annotations
- 14 @SuppressWarnings annotations

Results:

- Detected 9 correctness errors
- Detected 2 performance bugs

Examples:

Daikon

Javari: Java with reference immutability

A ReadOnly reference cannot be used to modify its referent.

```
@ReadOnly Date readonlyDate;
          Date mutableDate;

mutableDate.getTime();
readonlyDate.getTime();

mutableDate.setTime(time);
readonlyDate.setTime(time);
```

Javari: Java with reference immutability

A ReadOnly reference cannot be used to modify its referent.

```
@ReadOnly Date readonlyDate;
          Date mutableDate;

mutableDate.getTime();
readonlyDate.getTime();

mutableDate.setTime(time);
readonlyDate.setTime(time);  // Error: modifies a ReadOnly object!
```

Examples:

Listmatcher, Javari checker

Javarifier

- The Javarifier infers Javari annotations:
  - input: a set of class files
  - output: a set of annotated class files (or source, if available)
- Useful when working with third-party libraries or legacy code

Writing annotations on types

- Annotations on types enabled by JSR 308
- Backward-compatibility mode so code can compile in Java 5 and 6 (annotations in comments: `/*@NonNull*/`)

Creating new typecheckers

We have developed a framework for writing typecheckers:

- a template for traversing a program's source code
- an API for querying the annotations on types
- interfaces to the Java compiler (for reporting errors, querying the parse tree, etc.)

(The Interned and NonNull typecheckers are each around 350 lines of code.)

Summary: Custom type qualifiers for Java

We have created typecheckers for NonNull, Interned, the Javari language, and the IGJ language.

Programmers can:

- write qualifiers anywhere that types are used
- find and prevent bugs at compile time
- obtain guarantees that programs are free of certain errors
- create custom qualifiers and typecheckers

Download:
`http://pag.csail.mit.edu/jsr308`

Discuss: `mpapi@csail.mit.edu`