





Introducing Java 7

# MOVING JAVA FORWARD



ORACLE®

## Making heads and tails of Project Coin, Small language changes in JDK 7

Joseph D. Darcy



*“Project Coin is a suite of language and library changes to make things programmers do everyday easier.”*

coin, *n.* A piece of small change  
coin, *v.* To create new language

# Project Coin ready to use today!

- Remove extra text to make programs more *readable*
- Encourage writing programs that are more *reliable*
- Integrate well with past and future changes

# Coin Constraints

- *Small* language changes
  - Specification
  - Implementation
  - Testing
- Coordinate with larger language changes, past and future, such as Project Lambda
- Complex language interactions; need to be wary in
  - Specification
  - Implementation

# Coin Details

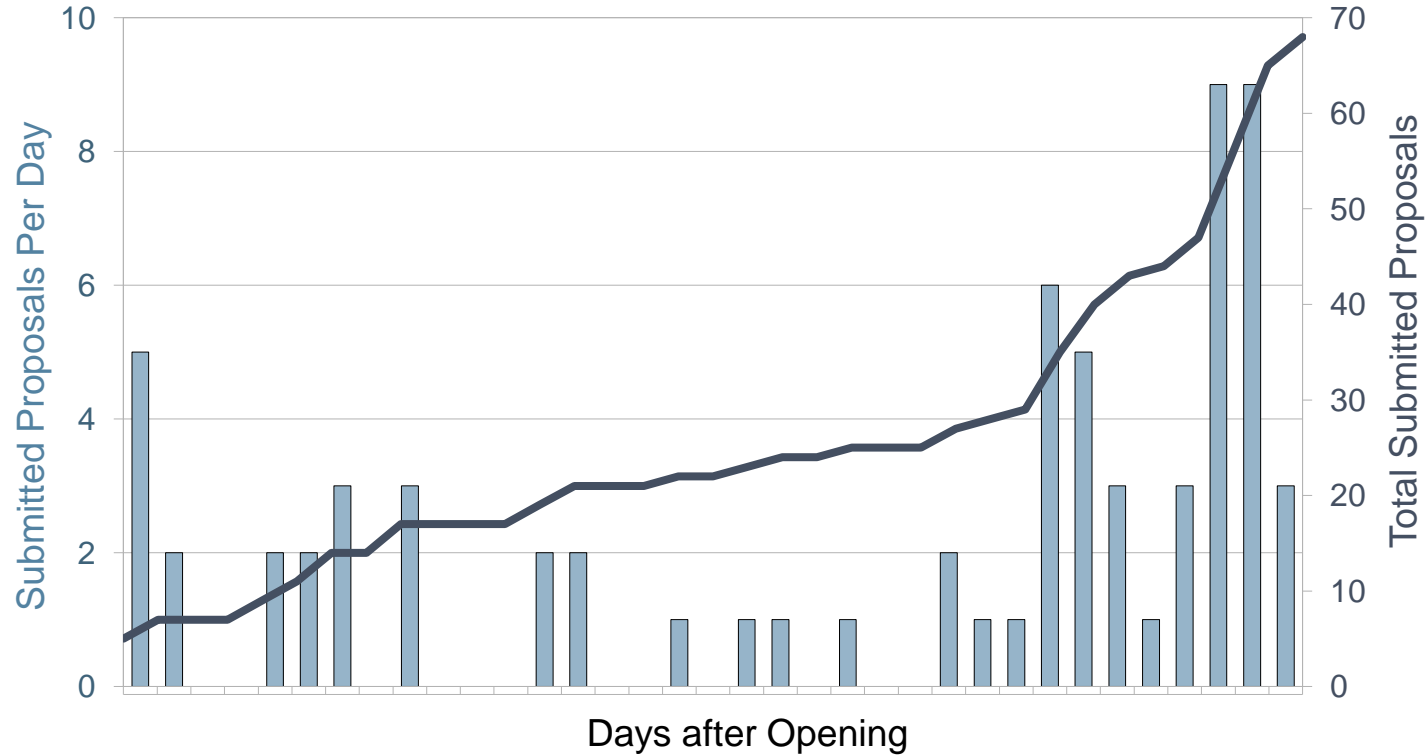
- Easier to use generics
  - Diamond
  - Varargs warnings
- More concise error handling
  - Multi-catch
  - `try-with-resources`
- Consistency and clarity
  - Strings in switch
  - Literal improvements

# Project Coin History

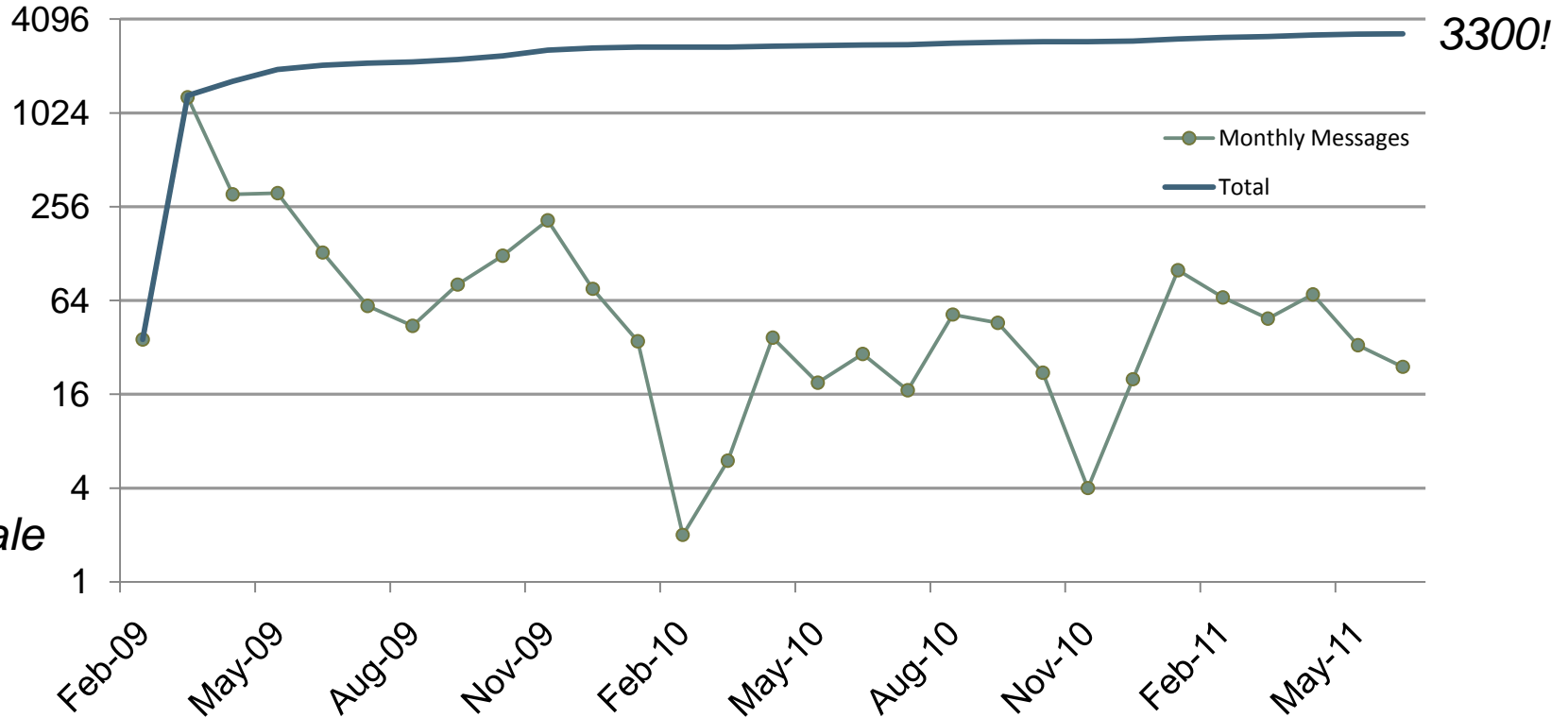
Specification and implementation

- Project Coin in OpenJDK:  
<http://openjdk.java.net/projects/coin/>
- Initial call for proposals to the community
- Implementation done as part of overall JDK 7 project

# Project Coin Proposals



# coin-dev traffic



Note:  
Log scale

# Project Coin Specification

- Specification in the JCP:  
JSR 334: “Small Enhancements to the  
Java™ Programming Language”  
<http://www.jcp.org/en/jsr/detail?id=334>
- Stages
  - Early draft review
  - Public Review
  - Proposed Final Draft
  - *Final approval ballot underway!*

# IDE Support

Here today and more on the way!

- Beta support in Eclipse  
<http://thecoderlounge.blogspot.com/2011/06/java-7-support-in-eclipse-jdt-beta.html>
- IntelliJ IDEA 10.5
- NetBeans 7.0  
<http://netbeans.org/kb/docs/java/javase-jdk7.html>

# *DEMO*

# Project Coin Features

- Binary literals and underscores in literals
- Strings in switch
- Diamond
- Multi-catch and more precise rethrow
- `try-with-resources`
- Varargs warnings

# Varargs warnings

- Summary: no longer receive uninformative unchecked compiler warnings from calling platform library methods:
  - `<T> List<T> Arrays.asList(T... a)`
  - `<T> boolean Collections.addAll(Collection<? super T> c, T... elements)`
  - `<E extends Enum<E>> EnumSet<E> EnumSet.of(E first, E... rest)`
  - `void javax.swing.SwingWorker.publish(V... chunks)`

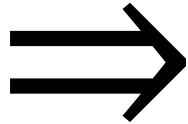
# Coin features easy to use

More work for the compiler!

- Type inference in diamond
- Internal compiler *desugaring*
  - multi-catch
  - strings in switch
  - **try**-with-resources

```
// Sugared
switch(s) {
  case "a":
  case "b":
  case "c":
    return 10;

  case "d":
  case "e":
  case "f":
    return 20;
  ...
}
```



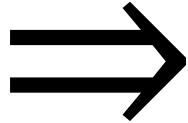
```
// Desugared
int $t = -1;
switch(s.hashCode()) {
  case 0x61: // "a".hashCode()
    if(s.equals("a")) $t = 1; break;
  case 0x62:
    if(s.equals("b")) $t = 2; break;
  ... }

switch($t) {
  case 1:
  case 2:
  case 3:
    return 10;

  case 4:
  case 5:
  case 6:
    return 20;
  ...
}
```

```
// Sugared
switch(s) {
  case "a":
  case "b":
  case "c":
    return 10;

  case "d":
  case "e":
  case "f":
    return 20;
  ...
}
```



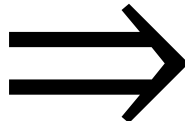
```
// Desugared
int $t = -1;
switch(s.hashCode()) {
  case 0x61: // "a".hashCode()
    if(s.equals("a")) $t = 1; break;
  case 0x62:
    if(s.equals("b")) $t = 2; break;
  ... }

switch($t) {
  case 1:
  case 2:
  case 3:
    return 10;

  case 4:
  case 5:
  case 6:
    return 20;
  ...
}
```

# try-with-resources desugaring

```
try ResourceSpecification  
    Block
```



```
{  
    final VariableModifiers_minus_final R #resource = Expression;  
    Throwable #primaryException = null;  
  
    try ResourceSpecificationtail  
        Block  
    catch (Throwable #t) {  
        #primaryException = t;  
        throw #t;  
    } finally {  
        if (#resource != null) {  
            if (#primaryException != null) {  
                try {  
                    #resource.close();  
                } catch (Throwable #suppressedException) {  
                    #primaryException.addSuppressed(#suppressedException);  
                }  
            } else {  
                #resource.close();  
            }  
        }  
    }  
}
```

# Library support

`try-with-resources` isn't just a language feature

- New superinterface `java.lang.AutoCloseable`
  - All `AutoCloseable` and by extension `java.io.Closeable` types usable with `try-with-resources`
- Retrofit `Closeable/AutoCloseable` to Java SE API
  - JDBC 4.1 retrofitted as `AutoCloseable` too
- *Suppressed exceptions* are recorded for posterity using a new facility `Throwable.addSuppressed`
  - Suppressed exception info included in stack trace, etc.

# Minty fresh libraries in JDK 7

Better JDK coding through “coinification”

- Retrofit Project Coin features in the JDK code base
  - Improve the existing code base
  - Validate the design of features through use
- Detectors / converters applied for
  - Diamond, `try-with-resources`
  - “JDK7, Coin, and Making Libraries Fresh and Minty,”  
<http://stuartmarks.wordpress.com/2010/12/23/jdk7-coin-and-making-libraries-fresh-and-minty/>
- Also used in new library and test code
  - `try-with-resources` in JSR 203/NIO2 utility methods

# A Systematic Update

- Automated code conversion for coinification
- Ran *annotation processors* over the JDK
  - Types to be retrofitted as **Closeable/AutoCloseable**:  
“Project Coin: Bringing it to a Close(able),”  
[http://blogs.sun.com/darcy/entry/project\\_coin\\_bring\\_close/](http://blogs.sun.com/darcy/entry/project_coin_bring_close/)
  - Methods and constructors to be annotated with **@SafeVarargs**  
“Project Coin: Safe Varargs in JDK Libraries,”  
[http://blogs.sun.com/darcy/entry/project\\_coin\\_safe\\_vararg\\_libraries/](http://blogs.sun.com/darcy/entry/project_coin_safe_vararg_libraries/)
- *Quantitative* language design

# Language design for the real world: Diamond

- Two inference algorithms were considered for diamond
  - Differed in how constraints were gathered
- Sometimes the 1<sup>st</sup> algorithm was more useful, *but* other times the 2<sup>nd</sup> algorithm was more useful
- What to do?
  - Is either one any good?
  - How to choose between them?
- Look at relative performance on a body of code

# Diamond outcome

On a body of millions of lines of code...

- Both algorithms were equally effective
  - Type arguments eliminated in 90% of constructor calls
  - A slightly different 90% for each algorithm
- Choose the algorithm with better future evolution properties

# The importance of source compatibility

A migration issue with more precise rethrow?

```
try {  
    throw new DaughterOfFoo();  
} catch (Foo e) {  
    try {  
        throw e; // Used to be treated as throwing Foo,  
                // would now throw DaughterOfFoo  
    } catch (SonOfFoo anotherException) {  
        ; // Reachable?  
    }  
}
```

# Should the feature be explicitly requested?

```
try {
    throw new DaughterOfFoo();
} catch (final Foo e) {
    try {
        throw e; // Used to be treated as throwing Foo,
                // would now throw DaughterOfFoo
    } catch (SonOfFoo anotherException) {
        ; // Reachable?
    }
}
```

# No; problem doesn't occur in practice

More precise analysis by default, easy path to migrate code

```
try {
    throw new DaughterOfFoo();
} catch (Foo e) {
    try {
        throw e; // Used to be treated as throwing Foo,
                // would now throw DaughterOfFoo
    } catch (SonOfFoo anotherException) {
        ; // Reachable?
    }
}
```

# Conclusion

- Project Coin features are easy to use today!
  - Ease writing readable and reliable code
  - Tooling support in IDEs
  - Systematic platform upgrade

# Q&A

[blogs.oracle.com/darcy](https://blogs.oracle.com/darcy)



