Eclipse and Java™ 8

Daniel Megert

Platform and JDT Lead

Eclipse PMC Member

IBM Rational Zurich Research Lab

Eclipse and Java^{TM} 8

- New Java language features
- Eclipse features for Java 8
- Behind the scenes

New Java Language Features

- **2 JSRs**
  - JSR-335: Lambda Expressions
  - JSR-308: Annotations on Java Types
- **2 JEPs**
  - JEP 118: Method Parameter Reflection
  - JEP 120: Repeating Annotations

JSR-335: Two New Type of Methods

- **Default methods**
  - Previous names:
    - Defender methods
    - Virtual extension methods
- **Static interface methods**
  - No OOP here!
    - Method must be qualified with exact interface type

Default Methods

- **Intention**
  - Allow evolution of interfaces (esp. in libraries)
  - Methods can be added to interface without API breakage
  - Why part of JSR-335?
    - Allows to add new methods that take a lambda expression: java.util.function.Function<T, R>
- **Consequences**
  - Multiple inheritance?
    - Yes, but compiler throws error if same method is inherited
  - Need to resolve manually with new construct: I.super.m()

JSR-335: Lambda Expressions

- **Many names used in the past**
  - Lambda Expressions, Closures, Anonymous Methods
- **Function + “captured state” (can have non-locals)**
- **Paradigm of passing a "code block as data"**
- **Get rid of verbose anonymous class syntax**

Lambda Expressions

- **Scope**
  - Anonymous classes introduce their own scopes
  - Interplay between names in enclosing scope ↔ inherited names
- **Capture**
  - Can capture explicitly final outer locals
  - And now since 1.8: effectively final locals
- **Expressions at the grammar level**
  - Lambda needs a context that provides target type

Lambda Expressions: Functional Interface

- **Lambda needs a context that provides target type**
- **Lambda only allowed for functional interfaces**
  - Interface with a single abstract method
    - Default methods don't count, but can be there
    - Static methods are not allowed, but can be there
    - Methods from Object don't count either
  - Optionally annotated with @FunctionalInterface
- **Lambda object implements a functional interface**

JSR-335: Method References

- **Very similar to lambda expressions**
  - Also require a target type
  - Target type must be a functional interface
  - Serve as instances of the functional interface
  - Don't provide a method body, but instead: **refer to an existing method**
  - void doSort(Integer[] ints) {
    Arrays.sort(ints, **Integer::compare**);
    }

JSR-308: Annotations on Java Types

- **But, couldn't we already do this before Java 8?**
  - void foo(@Foo String s) {}
  - No! The annotation was on the declaration (s)
  - Same here: @Foo String java17() {}
- **So far, only annotations on declarations**
  - ElementType: packages, classes, fields, methods, …
- **Java 8: annotations on types**
  - ElementType.TYPE_PARAMETER
  - ElementType.TYPE_USE

JSR-308: Annotations on Java Types

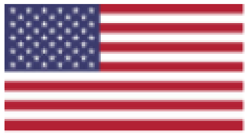
- Allows to add constraints to types anywhere in the code
- Leveraged in Eclipse to improve null analysis

Behind the Scenes

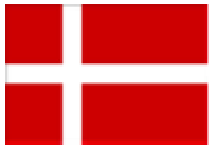
- The Team
- How did we implement the Java 8 specs?
- Java 8 effort by numbers

The Team

- Andy Clement
- Steve Francisco
- Michael Rennie
- Olivier Thomann
- Curtis Windatt

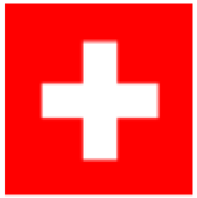


- Walter Harley
- David Williams



- Jesper S. Møller

- Stephan Herrmann



- Markus Keller
- Dani Megert

- Jay Arthanareeswaran
- Deepak Azad
- Shankha Banerjee
- Anirban Chakarborty
- Vikas Chandra
- Noopur Gupta
- Ayushman Jain
- Manju Mathew
- Manoj Palat
- Srikanth Sankaran
- Sarika Sinha

Implementing the Specs

- **Initially: javac defined/drove specs**
- **Eclipse must only use spec, but**
  - Incomplete (April – Sept 2013)
  - Inaccurate or undefined in some parts
- **We participated in the JSR expert groups**
- **Users report differences between ECJ and javac**
  - ECJ? javac bug? JLS bug?
  - Who is the master, JLS or javac?
- **We helped to make the spec more concise!**

JDT Does Not Accept Contributions! Really?

- 2012 starts with a JDT team that has 4 core and 4 UI committers/experts
- Half of the team gone by summer 2012!
- Hard to find new people with compiler know-how
- Backfilled by the end of the year
- BUT: New people had zero knowledge of JDT
- Hard life for existing committers: train new people and make progress on Java 8

JDT Does Not Accept Contributions! Really?

- Not much room/energy to review contributions unrelated to Java 8?
- JDT spent lots of time to review contributions!
- JDT Core: 50 contributions from 20 people
- JDT UI: 47 contributions from 15 people

Java 8 Effort by Numbers

- First commit in May 25, 2012
- 3 big projects tested compiler to build it JDK 8, OpenJFX and Eclipse SDK
- 31 people contributed code
- 800 bugs/enhancements fixed for Java 8
- 1500+ commits