JDT Embraces Java 8

Stephan Herrmann

Committer:
- Object Teams
- JDT

Java 8 ready

Eclipse DemoCamp 2014, Munich, Germany

© 2014 by Stephan Herrmann; made available under the EPL v1.0
THE RETAIL APPLICATION Company
- POS
- Store Integration
- Mobile Devices
- ...

Domain Specific Languages
Java (Server & Client)
Hardware Integration

Eclipse DemoCamp 2014, Munich, Germany
© 2014 by Stephan Herrmann; made available under the EPL v1.0
Java 8 Support: Luna ~ “SR1”

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

- Jesper S. Møller

- Stephan Herrmann

- Walter Harley
- David Williams

- Jay Arthanareeswaran
- Anirban Chakarborty
- Manoj Palat
- Shankha Banerjee
- Manju Mathew
- Noopur Gupta
- Deepak Azad
- Srikanth Sankaran
• Java 8 features supported:
  – JSR 308 - Type Annotations.
  – JEP 120 - Repeating Annotations.
  – JSR 269 - Pluggable Annotation Processor API & javax.lang.model API enhancements for Java 8.
  – JSR 335 – Lambda Expressions
    • Lambda expressions & method/constructor references
    • Support for “code carrying” interface methods
    • Enhanced target typing
    new overload resolution & type inference
Evolution or Disruption

- Assumptions made in the implementation
  - Grammar can be made to conform to LALR(1)
  - Type bindings can be compared with "==" *(interned)*
  - Compilation happens in sequential phases
    - overload resolution is based on types being resolved
    - flow analysis starts after resolution is completed
  - Indexing is word-based (parsing suffices)
  - Code completion can stop parsing at cursor

None of these assumptions are valid for Java 8
Revolution

- Fundamental changes in many places
- Full rewrite of
  - Type Inference
Demo: \(\lambda\)
• **DEMO JDT Support for Lambda**
  
  - Quick assist
    - to / from lambda
    - expression ↔ block
  
  - Hover
    - to see the functional type
    - to see inference results

• **Not shown**
  
  - Search / Type hierarchy / Refactoring
    - are lambda aware
The Compiler

• Reasons for having our own compiler
  – incremental compilation (as you type)
    • good error recovery
  – run even code with errors
  – use compiler information in the UI
    • visualization
    • navigation
    • quick assist & refactoring
  – QUALITY ASSURANCE
abstractly capture the concepts

spec

implement the spec, the full spec, and nothing but the spec

javac

compare behavior

bug?
deviation

bug?

bug?
Areas Covered

- **Connecting objects to a system**
  - nothing new

- **Algorithms**
  - lambda expressions
  - library updates for lambdas

- **Types**
  - type annotations
  - type inference
Object Composition

playedBy Relationship

- **Properties**
  - **Dynamism:** roles can come and go (same base object)
  - **Multiplicities:** one base can play several roles (different/same role types)
Areas Covered

- Connecting objects to a system
  - **Object Teams**
- Algorithms
  - lambda expressions
  - library updates for lambdas
- Types
  - type annotations
  - type inference