Advanced Java™ Programming Language Refactoring: Pushing the Envelope

Tom Ball

Technical Director, Developer Products Group
Sun Microsystems, Inc.
http://www.netbeans.org/

TS-9861

Goal of This Talk

What you will gain

Learn how automated Java™ programming language refactoring technology is improving, how it works, and how your project can benefit.

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

Software Economics 101

- Software has an intrinsic value
- Software projects have unique half-lives
- Developers add value by changing software
  - Adding features
  - Improving quality, performance
- Changes have a cost
  - Time to implement, deploy
  - Developers are expensive
  - Lost opportunities
- Changes must add more value than their cost

The Bad News

- The pace of software change is increasing:
  - Release early and often
  - Continuous betas
  - New technology half-lives shrinking
- Developer costs are increasing:
  - Project complexity increasing
  - Baby boomers approaching retirement
  - Yet CS majors continue to decline
- Existing methodologies aren't scaling

The Good News

- Agile methodologies are gaining acceptance
- New frameworks are reducing costs of client-server applications
  - JavaServer™ Faces Technology
  - Ruby on Rails™
  - AJAX toolkits
- Development tools are becoming more capable
  - IDEs all integrating refactoring
  - Java technology defect analysis tools becoming mainstream

Refactoring 1.0 Definition

"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure."

Refactoring: Improving the Design of Existing Code
Martin Fowler, 1999

1.0 Definition Limits

- A refactoring had to be in “the book”
  - IDE menu entries reflected this:
    - “Introduce Parameter Object”
    - “Form Template Method”
  - Terminology scares away new users
- Refactoring is done “in the small”
  - Code hand-selected
  - Refactoring invoked via menus
  - Preview detailing every change
  - Developer should review every change before commit

Refactoring 2.0

- Continually integrated into workflow
- “Just do it” smart editing
- Tools share rich model of projects
- Integrated defect detection
  - Coupled with defect correction (when possible)
- Automated re-engineering
  - Large-scale application of small refactorings
  - API migration

Refactoring 2.0 Definition?

"Refactoring is the process of **continually updating** a software system in such a way that it **adds significant value without incurring major cost or risk.**"

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

Java Technology Model-Driven Editing

- All IDEs now use models of Java technology projects
  - Aids navigation
    - Jump to definitions, parent classes and methods
    - Find element references accurately
    - Editor navigation aligns to developer's map
  - Error detection
    - Type conflicts
    - Invalid overriding, method and parameter names
  - Richer models
    - Lexical ➔ Syntactic ➔ Semantic
    - More accurate model allows more safe capabilities

Java Technology Compiler IDE Integration

- IDEs now use Java technology compiler-generated ASTs
  - Java programming language semantic modeling very difficult
    - Accurate type relationships hard
    - Generic types much harder
    - Let the compiler gurus figure it all out
  - Error reporting fully synced to build messages
  - New Java programming language features quickly supported
- All Java IDEs will soon leverage compiler

Editor Tips

- Editor tips show suggested changes to source
  - Based on problems found while editing
    - Background compilation errors
    - Independent model checks
  - Many problems have direct solutions
  - Tip suggests one or more refactoring solutions
    - Accepting tip applies immediate refactoring
  - Eliminates hand selection, manual approval of change
- Refactoring now rarely a separate activity
- Soon tests will be configurable, expandable

DEMO

Java Programming Language Editor Tips

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

What is Reengineering?

- Refactoring is not supposed to change behavior

but...

Sometimes program behavior needs changing

- Examples:
  - Anti-pattern detection and correction
  - Global refactoring and re-design
  - API Migration

Anti-Patterns

- Best practices often have inverse anti-patterns
- Security anti-pattern examples
  - Non-final public static variables
  - Incorrect privileged code declaration and use
- Concurrency anti-pattern examples
  - Overly broad synchronization
  - Incorrect lock ordering
- Static analysis can detect and often fix errors

Jackpot Reengineering

- Jackpot executes custom queries
  - Full access to complete Java language semantic model
  - Scales to multiple, large projects
- Transformers can modify query matches
  - Not limited to classic refactoring
  - The only restriction is that result must compile
- Rule language
  - Transforms statements and expressions
  - Jackpot API for other transformations
- Engine integrated in NetBeans™ 6.0 IDE

DEMO

Jackpot Quick Look

Reengineering Uses

- API evolution
  - Update genericized class clients
  - Convert deprecated references
- API migration
  - Service replacement
  - Framework switching
- Evolve projects to meet current best practices
  - Eliminate micro-optimizations
  - Add annotations

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

Model-Driven Tool Ingredients

- Project definition
  - Complete source file list
  - Libraries, dependencies
  - Build settings

  } analyze only
- Parse trees
- Type hierarchy
- Element (symbol) hierarchy
- Model-based source rewriter

Useful Java Platform, Standard Edition (Java SE Platform) v.6 API

- javac Compiler API
  - com.sun.source.*
  - Parse tree classes, utilities
- Java Specification Request (JSR) 269
  - javax.lang.model.*
  - Type and element classes, utilities
- JSR 199
  - javax.tools.*
  - Compiler invocation

Useful NetBeans 6.0 IDE API

- Java Source API
  - org.netbeans.api.java.source.*
  - Compiler integration
  - References database
  - Source code editing
- Jackpot API
  - org.netbeans.api.jackpot.*
  - Reengineering command, UI support
- Other modules
  - Lots of reference code for different tasks

Anatomy of a Model-Driven Task

- Create Java Source instance
  - Source file list
  - Sourcepath, classpath and boot classpath
  - Build settings
  - Class, package indices
- Execute one or more tasks with it
  - runUserActionTask() for queries
  - runModificationTask() for refactorings
- Commit ModificationResult to change source

Task Skeleton

```
// define visitor to be executed
TreeScanner scanner = new TreeScanner() { ... };

// define task to execute
    CancellableTask task =
new CancellableTask<WorkingCopy>(){
public void run(WorkingCopy wc) {
    wc.toPhase(Phase.RESOLVED);
                wc.getCompilationUnit().accept(visitor);
    }
    public void cancel() { ... }
}
// execute task and commit its modifications
    javaSource.runModificationTask(task).commit();
```

Basic Concepts

- Access parse trees using visitor pattern
  - Don’t assume parent/child node types
  - Parse trees may change
  - Visitors make for tighter code
    - Fewer corner cases to worry about
- Parse trees only persist per source file
  - Do not hold references
- Parse trees, types, elements are immutable
  - Submit replacement tree to rewrite

Example: Add @Override Annotation

```
// create Override annotation tree
TypeElement override =
    elements.getTypeElement("java.lang.Override");
AnnotationTree ann = make.Annotation(
    make.QualIdent(override), Collections.emptyList());

// create new modifiers tree with annotation
List<AnnotationTree> newAnns =
    new ArrayList<AnnotationTree>();
newAnns.addAll(mods.getAnnotations());
newAnns.add(ann);
ModifiersTree newMods = make.Modifiers(mods, newAnns);

// tell Java Source to substitute old modifiers for new
workingCopy.rewrite(mods, newMods);
```

What This Talk Will Cover

Refactoring 2.0?

Integrated Refactoring

Java Technology Reengineering

How It's Done, How to Do It Yourself

Getting Radical: What's Coming Soon

Upgrade to New Language Features

- Generify references
  - Inspects client usage of generic classes
  - Determines closest type parameters
  - Reports suspicious use
- Convert for statements to enhanced for
- Convert interface constants to Enum
- Eliminate primitive wrappers for collections
- Convert parameter array to varargs
- Add @Overrides annotations

Concurrency Anti-Patterns

- Find unsafe construction
  - Report ways partial objects can be published
- Wrap lock use in try/finally block
- Transform double-checked locking
  - Replace with static lazy initialization
- Convert synchronized variable to atomic
- Narrow lock scope
- Remove unnecessary synchronization

Metrics-Driven Refactoring

- Metrics are related to code value
  - Complexity measures
  - Class coupling
- There are no absolute metric values, but…
- Lowering metrics increase value
  - Reduced complexity/coupling == lower maintenance
- Move method to best place
  - Moves method to class with highest coupling
- Split large method

Profiler-Driven Refactoring

- Static analysis cannot analyze all problems
- So add execution data to model
- Replace collection type
  - Monitor collection use
  - Match use to best algorithm
- Exclusive lock to ReadWriteLock
  - Used for frequent, mostly read access
- Reduce lock granularity
  - Introduce lock splitting, striping

Summary

- Java technology is taking lead in refactoring technology
- Refactoring is now deeply integrated into IDEs
- Public APIs allow any developers to define new refactorings
- Powerful Java technology refactorings are just beginning to emerge

For More Information

Projects

- NetBeans 6.0 IDE: new model-driven Java code editor
  - http://www.netbeans.info/downloads/dev.php
- Locksmith: concurrency refactorings for IntelliJ IDEA
  - http://www.sixthandredriver.com/locksmith.html
- Sorcerer: AST-based source code references browser
  - https://sorcerer.dev.java.net/

URLs

- Jackpot: http://jackpot.netbeans.org

Q&A

Advanced Java™ Programming Language Refactoring: Pushing the Envelope

Tom Ball

Technical Director, Developer Products Group
Sun Microsystems, Inc.
http://www.netbeans.org/

TS-9861