Hack Into Your Compiler!

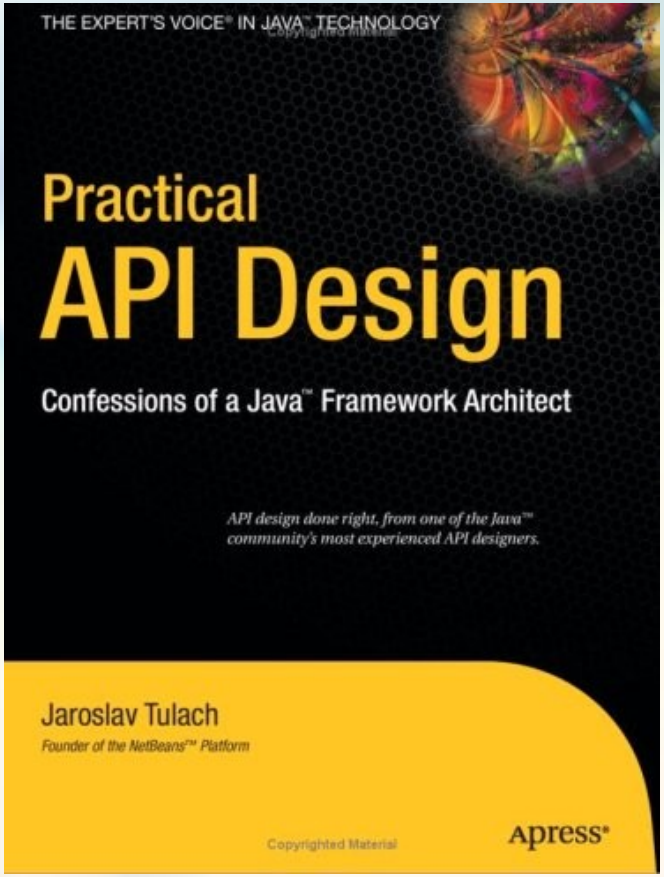
Compile Time Annotations

Jaroslav Tulach
Oracle

NetBeans

Agenda

- Modularity vs. Performance
- Domain Specific Languages
  - > In Java
- API Design Example
  - > Scalable & flexible URLs
- NetBeans IDE & Platform
  - > Common Registration Style
- Q&A



Demo

Set Up NetBeans 7.2 and Sources

Modular Registrations

- Registration takes time
  - `MyClass.registerProvider(...)`
- Unregistration often forgotten
  - `MyClass.unregisterProvider(...)`
- Lookup on Demand
  - “The most important functionality”
  - scalability
- Classloading takes time & memory

Scalable Registrations

- Declare registration
  - > Special language – not in Java
  - > Handled by framework
  - > Unregistration for free
- @ServiceProvider
  - > META-INF/services/pkg.name.Intrfce
- Pull vs. push
  - > Lookup.getDefault().lookupAll(...)
  - > Spring & other DI
- Loads all implementations
  - > Enhance the registration language!

What is a DSL?

A *programming* language or *specification* language dedicated to

- a particular **problem domain**,
- a particular **problem representation technique**, and/or
- a particular **solution technique**.

--wikipedia (http://en.wikipedia.org/wiki/Domain_Specific_Language)

DSL vs. Library API Shootout

Rich Unger
Salesforce.com

Jaroslav Tulach
Oracle

JavaOne
September 2010

DSL Classification

- **Processing style**
  - Own parser – *External*
  - XML based
  - Embedded in other language - *Internal*
- **Computational power**
  - Declarative programming
  - Turing complete
- **Tooling**
  - Need to extend IDEs to support the DSL
  - Tooling standardized for all IDEs

DSL Examples

- LOGO (or Karel)
- SQL
- ZIL (Zork Implementation Language)
- Postscript
- TeX
- CSS
- BNF Grammars (YACC, Antlr, etc)
- Apex
- XML variants (Ant, VoiceXML, XSLT, Docbook, SVG)
- Embedded/Internal (in Haskell, Scala, Java6)

It's Okay to Use XML?

- Quick to develop
- Free lexing
- Lots of existing libraries to manipulate it
- Standard syntax for AST representation
- Poor performance
- Unreadable to humans

```
Michael | Yuriko | Mary
 | Duke | $otherNames

/10/ small | /2/ medium | large
```

```
<one-of>
  <item>Michael</item>
  <item>Yuriko</item>
  <item>Mary</item>
  <item>Duke</item>
  <item>
    <ruleref uri="#otherNames"/>
  </item>
</one-of>

<one-of>
  <item weight="10">small</item>
  <item weight="2">medium</item>
  <item>large</item>
</one-of>
```

NetBeans Intermezzo: XML Layer

```
    <folder name="Shortcuts">
        <file name="OS-ENTER.shadow">
    <attr name="originalFile" stringvalue="Actions/Window/org-netbeans-core-windows-actions-
ToggleFullScreenAction.instance"/>
        </file>
    </folder>


    <folder name="Menu">
        <folder name="View">
            <attr name="position" intvalue="300"/>
            <file name="Separator1.instance">
        <attr name="instanceClass" stringvalue="javax.swing.JSeparator"/>
        <attr name="position" intvalue="300"/>
            </file>
```

Libraries and Embedded DSLs

- Lexing automated
- Free interpretation
- Targeting wide audience of developers
- Bound to syntax of the language
  - Not a real problem for functional languages
  - People like Java
- Creates de-facto new language
  - Reading on paper?

```
expr   ::= expr ’+’ term | term
term   ::= term ’*’ factor | factor
factor ::= ’(’ expr ’)’ | digit+
digit  ::= ’0’ | ’1’ | ... | ’9’
```

```
object ArithmeticParser extends StdTokenParsers {

  type Tokens = StdLexical ; val lexical = new StdLexical
  lexical.delimiters ++= List("(", ")", "+", "*")


  lazy val expr =   term*("+" ^^^ {(x: int, y: int) => x + y} )
  lazy val term = factor*("*" ^^^ {(x: int, y: int) => x * y} )
  lazy val factor: Parser[int] = "(" ~> expr <~ ")" | numericLit ^^ (_.toInt)
}
```

Declarative Java

- Developers love Java
  - Industry is conservative
- Can Java be DSL meta language?
- Better options than external DSL?
  - Annotations
- Hook into Java compiler
  - Annotation Processors

Demo

Register URL Schemas

URL Schema Registration

- Declarative
  - Home made format
- Extensible & Scalable
  - getResource(“META-INF/url/prefix”);
  - Only needed instances created
- Error Detection
  - One cannot type wrong class name
  - Still errors can happen

Demo

Check public constructor

Processor Validation

- Emit errors and warnings
  - Wrong access modifiers
  - Wrong method parameters
  - Supertype and interfaces
- Information about source structure
  - Classes
  - Methods
  - Fields
- Not about method bodies
  - Not Turing complete

Demo

Simplify Source Code

Influence Code

- Cannot change existing sources
  - Unlike Lombok
  - Not flexible
  - Still powerful
- Can generate new
  - Including those to “extend”
- Extend “non-existent” pattern
  - Close relationship between subclass/superclass

Demo

Generate boilerplate code

Generated Sources

- Virtual sources
  - Nobody seen them before
- IDEs
  - Generate on fly
  - Not playing catch-up
  - Navigator & Code Completion
- Incremental vs. clean build
  - API not ready for incremental
  - Many files contribute to single file
  - Tough to make it work right

Demo

In place localization with @Messages

Demo

Code Completion via JSR-296

Back to DSLs

- DSL to access a database
- Ruby on Rails approach
  - Column <=> field
  - Table <=> list of types
- Java too verbose
  - @Entity
  - Out of sync
- Java on Rails!
  - Proof of concept

Demo

LiveDB in Java

NetBeans Platform

- Generate layer.xml
  - Extend `LayerGeneratingProcessor`
  - Use File builder
  - No need to deal with XML
  - Merge automatic
- Unreadable layer.xml, but:
  - Good for automatic processing
  - Remains compatible
  - Easy to cache and store effectively
  - Unified storage

Versioning of Annotation Processors

- Compile time
- Complete control on generated code
- Annotations support default values
- Adding new annotations

```
/version 1.0
@ActionRegistration
class MyAction {
}
```

```
// version 1.1
@ActionRegistration(asynchronous=true)
class MyAction {
}
```

```
// alternative 1.1
@ActionRegistration
@ActionAsynchronous
class MyAction {
}
```

Versioning of Compiler

- Compilers evolve
  - Processor workaround bugs
  - Language incorporates features
- Compiler and libraries independent
  - Compile old library with new compiler
- Processor need dependencies
  - And version info

Extensibility of Annotations

- No inheritance
- Composition of many
  - One master annotation (`@ActionID`)
- Need many processors to “talk”
  - The master one leads the processing
  - Annotate with meta annotations

```
@ActionID(id=”my.action”)
@ActionRegistration
@ActionAsynchronous
@ActionReference
class MyAction {
}
```

```
@ActionID.Generator(generator=MyProcessor.class)
@interface @ActionRegistration {
}
```

Summary

- Languages/Protocols Happen
  - > Modular registrations
- Easy to parse => hard to write
  - > Simplify writing using @annotations
- Conversion during compilation
  - > Easy to read structures
  - > Easy to use generated classes
- NetBeans Platform
  - > General purpose filesystem like storage

Remaining

Q&A

Get the code at https://github.com/jtulach/Annotations