The Art of Metaprogramming in Java

Abdelmonaim Remani
@PolymathicCoder
About Me

- Software Architect at Just.me Inc.
- Interested in technology evangelism and enterprise software development and architecture
- Frequent speaker (JavaOne, JAX, OSCON, OREDEV, etc…)
- Open-source advocate
- President and founder of a number of user groups
  - NorCal Java User Group
  - The Silicon Valley Spring User Group
  - The Silicon Valley Dart Meetup
- Bio:  [http://about.me/PolymathicCoder](http://about.me/PolymathicCoder)
- Twitter:  [@PolymathicCoder](https://twitter.com/PolymathicCoder)
- Email:  abdelmonaim.remani@gmail.com
About Me

- Today is my birthday!
License

- Creative Commons Attribution Non-Commercial 3.0 Unported
  - [http://creativecommons.org/licenses/by-nc/3.0](http://creativecommons.org/licenses/by-nc/3.0)

- Disclaimer: The graphics, logos, and trademarks used in this presentation belong to their rightful owners.
Metadata
What Is Metadata?

- The term “Metadata” was coined by Philip Bagley in 1986 in “Extension of Programming Language Concepts”
- Data about data or data that describes other data
  - Structural (Before there is any data)
    - Its type
    - Its valid values
    - How it relates to other data
    - Its purpose
    - Etc…
  - Descriptive (After data is there)
    - How it was created
    - The context within which it exists
    - Etc…
How Is Metadata Expressed?

- Implicit
  - Adhering to a certain convention

- Explicit
  - External to the code
    - DSL (Domain-Specific Language)
    - Markup: XML, RDF, etc…
    - Etc…
  - Internal to the code
    - As comments on code
      - Javadoc: @author, @version, @since, etc…
      - XDoclet
      - Etc…
    - As code itself
      - HTML meta tags: <meta name="author" content="Abdelmonaim Remani">
      - Java Annotations: @Override, @Deprecated, etc…
      - Embedded DSL
      - Etc…
How Is Metadata Being Used?

- **Schema**
  - Data Dictionary in RDBMS
  - Check Constraints (JSR 303 Validation, Etc…)
  - Etc…

- **Semantics**
  - WSDL (Web Services Description Language)
  - RDF (Resource Description Framework)
  - Etc…

- **Data Management**
  - Build/deployment instructions (.svn files, .git, etc…)
  - Etc…

- **Configurations**
- Etc…
Metadata in Java: Annotations
Java Annotations

- JSR 175 (A Metadata Facility for the Java Programming Language)
  - Introduced in J2SE 5.0 (September 30, 2004)
  - Standardized how annotations are declared in Java code
- An alternative way to Javadoc comments, externally as XML, etc…
  - More readable
  - Closer to the code
  - Statically-typed
  - Can be retained until runtime
JDK Annotations

- Java Language Spec Annotations
  - @Override
  - @Deprecated
  - @ SuppressWarning

- JSR 250 (Common Annotations for the Java Platform)
  - @PostConstruct
  - @Resource
  - @DenyAll
  - Etc…
Write Your Own

- You need to tell the compiler how the annotation is to be treated
- In java.lang.annotation package
  - @Target
    - The element type the annotation can be applied to
      - ElementType.ANNOTATION_TYPE
      - ElementType.CONSTRUCTOR
      - ElementType.FIELD
      - ElementType.LOCAL_VARIABLE
      - ElementType.METHOD
      - ElementType.PARAMETER
      - ElementType.TYPE
Write Your Own

- In java.lang.annotation package
  - @Retention
    - RetentionPolicy.SOURCE
      - Discarded by the compiler
    - RetentionPolicy.CLASS
      - Included in the class file but ignored the JVM. This is the default
    - RetentionPolicy.RUNTIME
      - Included in the class file and read by the JVM.
  - @Documented
    - Whether it should be shown in the javadoc or not
  - @Inherited
    - Allowed to be inherited by subclasses
Write Your Own

- Attributes
  - Carry additional metadata details
  - May only be
    - Primitives
    - Enums
    - Java.lang.String
    - Java.lang.Class
    - One-dimensional arrays of the above
  - May have a default value
  - No modifiers or parameters
public class MyClass {

    @CodeReview(authors= {"Abdelmonaim Remani"}, creationDate="07/19/2012")

    @Inherited
    @Target(ElementType.TYPE)
    @Retention(RetentionPolicy.SOURCE)

    public @interface CodeReview {
        String[] authors();
        String creationDate() default "01/01/1970";
        int version() default 1;
        Status status() default Status.NOT_REVIEWED;
    }
}
Annotating a package
- Must create a file named package-info.java in the target package
- Note that “package-info” is not a valid Java identifier
  - Most IDEs you will prevent you from creating a class name “package-info”
  - Make sure you create it as a regular file with .java extension instead
Metaprogramming
What is Metaprogramming?

- Writing programs that write or manipulate other programs or themselves based on some metadata
- Metaprogramming
  - != Generative/Automatic Programming
- Ralph Johnson
  - “It is about programming at the meta level, that is about changing your interpreter or changing how code is compiled”
- Black art and a Big-boy’s toy
- An underused feature
- Supported in many languages and across several platforms
- The backbone of many of the most successful frameworks
How is Metaprogramming Supported?

- Exposing the internals of the runtime/compiler as an API
- Dynamic execution of expressions containing programming commands
  - Code as Strings
  - Code as a series of method calls
- A program transformation system
  - A description gets transformed into a target language
    - The compiler itself
    - YACC takes in grammar, and produces C/C++ code containing `yyparse()`
    - ANTLR (ANother Tool for Language Recognition)
Concepts

- **Metalanguage**
  - The language in which the metaprogram is written in

- **Object Language**
  - The language in which the target (produced) program is written in

- **Reflection or Reflexivity**
  - When metalanguage == object language
  - No translation necessary
Usage in code

- Static data that can be pre-computed or pre-generated at compile time
- Eliminate boiler-plate
  - Code that cannot be abstracted in functions for DRYness sake
    - Think Aspects in AOP
    - Stereotypes in Spring
    - Etc…
  - Code of common methods
    - Getters/setters, toString(), hashCode(), equals(), etc…
- Etc…
Benefits

- In code
  - Performance gain
  - Flexibility
  - Simplicity
  - Etc...

- Development
  - Minimize the LOC to express a solution
  - Productivity gain
  - Reduced development time/efficiency
  - Etc…
How to?

- Many techniques focused on specific aspects of metaprogramming
- No well-defined best practices

This presentation is an attempt to bring in some structure through defining a process
- Defining the metadata
- Processing the metadata
- Metamodel construction
- Validating the metamodel
- Metamodel interpretation
Metadata Processing
Metadata Processing

- Programmatically reading/accessing metadata
  - Parsing raw metadata
  - Call to an API
    - Reflection
    - Query
    - Etc…
  - Tools
Metadata Processing in Java: Annotation Processing
At Runtime: JSR 175

- **JSR 175: A Metadata Facility for the Java Programming Language**
  - Defined the core reflection API for reading/accessing annotations on annotated elements at runtime as long as their retention policy extends to the runtime
  - Reflection
    - Reading annotations is done in reference to the structure of the program

- **Libraries**
  - Etc…
At Runtime: JSR 175

```java
Annotation[] annotations = MyClass.class.getAnnotations();
for (Annotation annotation : annotations) {
    if (annotation instanceof CodeReview) {
        CodeReview codeReviewAnnotation = (CodeReview) annotation;
        System.out.println(codeReviewAnnotation.value());
    }
}
```
At Build Time: JSR 269

- Mirror-Based Reflection
  - Reflective capabilities are encapsulated in intermediary objects called mirrors
  - Annotations are accessible through a processor API
- In J2SE 5.0 didn’t standardize a processing API
  - We used apt, a stand-alone tool, along with the Mirror API (com.sun.mirror)
At Build Time: JSR 269

- In Java SE 6 (December 11, 2006)
  - JSR 269: Pluggable Annotation Processing API
    - http://docs.oracle.com/javase/6/docs/technotes/guides/javac/index.htm
  - Leverages JSR 199 (Java Compiler API)
    - Javax.tools
    - Programmatically invoke javac
    - Implements ServiceLoader interface of SPI (JSR 24 - ServiceProvider API))
    - Provides the DiagnosticListener interface to allow listening for warnings and error by the compiler
  - Extends javac as a plug-in allowing to write custom annotation processors
  - Seamless integration with javac
    - Finds if there is an annotation registered to process a particular annotation
    - Plugs it into the compiler
At Build Time: JSR 269

- **JSR 269 Defines 2 APIs**
  - Writing annotation processors and interacting with the processing environment
    - Javax.annotation.processing
  - Modeling the Java Programming Language
    - Javax.lang.model
      - Type declarations and types (Accommodates Generics)
@SupportedSourceVersion(SourceVersion.RELEASE_6)
@SupportedAnnotationTypes("com.polymathiccoder.talks.metaprogramming.annotation.CodeReview")
public class CodeReviewProcessor extends AbstractProcessor {
    @Override
    public boolean process(Set<? extends TypeElement> annotations, RoundEnvironment roundEnvironment) {
        Messager messager = this.processingEnv.getMessager();
        for (TypeElement typeElement : annotations) {
            for (Element element : roundEnvironment.getElementsAnnotatedWith(typeElement)) {
                messager.printMessage(Diagnostic.Kind.ERROR, "I'm told to log an error even though I already have.");
            }
        }
        return true;
    }
}
Code

- Implement `javax.annotation.processing.Processor` or extend `javax.annotation.processing.AbstractProcessor`
  - Process method
    - Returns whether or not the annotations processed are claimed by the processor. This determines whether other processors can further process them or not
- Configure the processor using annotations
  - `@SupportedAnnotationTypes`
    - Register the annotations to be processed with their full-qualified name. Wildcats are supported as well
  - `@SupportedSourceVersion`
    - The Java version supported
  - `@SupportedOptions`
    - Register supported command-line options
Code

- Annotation processor registration
  - Javac command-line options
    - -processor <processor classes>
    - -proc:none or -proc:only. It is enabled by default
    - -processorpath <annotation path>
  - Leverage JSR 24 (ServiceProvider API) to automatically register processors
    - Compile the processor with –proc:none and package it in a jar
    - Include in META-INF/services a file named javax.annotation.processing.Processor containing a text entry of the full-qualified name of the processor class
    - Optional: Multiple processor can be registered or ordered
Metadata Processing

- Annotated elements might have nested annotated elements and so forth
- Visitor Design Pattern
  - Separation of annotation processing code from the object structure
Metamodell Construction
What is a Metamodel?

- Metadata is processed into a model that can be accessed programmatically
  - Static
  - Dynamic
- A association of the data and its metadata
Static Metamodel

- A metamodel based on
  - One all-knowing “god object” encapsulating all possible metadata values that could be associated with the one annotated element

- Advantages
  - Simple
  - Statically typed

- Disadvantages
  - Nulls all over
Static Metamodel

```java
@Id
@Generated(type=Strategy.SEQUENTIAL)
@Column(name="ID")
private long id;

@Column(name="FULL_NAME")
private String fullName;

@Unique
@Column(name="EMAIL")
private String email;

public class MetaModel {
    private Map<Field, PossibleMetaData> value = new HashMap<Field, PossibleMetaData>;

    public static class PossibleMetaData {
        private boolean isId;
        private String name;
        private boolean isUnique;
        private Strategy type;
    }
}
```
Dynamic Metamodel

- **Ravioli Code**
  - Metamodel is structured in small and loosely-coupled components
- **Decorator Design Pattern**
  - Annotated elements are decorated with annotations that bring in metadata
- **Advantages**
  - Flexible
- **Disadvantages**
  - Complex
Dynamic Metamodel

AnnotatedElement
- annotatedElement : java.lang.reflect.AnnotatedElement
- metamodelData : MultiMap<java.lang.reflect.AnnotatedElement, Object>
<<constructor>> + AnnotatedElement(java.lang.reflect.AnnotatedElement, String) : AnnotatedElement

<<abstract>>
AnnotatedDecorator
- decoratedAnnotated : Annotated
<<constructor>> + AnnotatedDecorator(Annotated) : AnnotatedDecorator

AnnotatedWithSampleAnnotation
<<constructor>> + AnnotatedDecorator(Annotated) : AnnotatedDecorator
Metamodel Validation
Validation?

- Ensuring the validity or correctness of semantics of the metamodel
  - Verification that a set of constraints are satisfied
- Compliance additional consistency constraints outside the Java language specification
  - Java only has @Target
- Example
  - Assuming that you are writing your own JSR 318 (Enterprise javabeans 3.1) implementation
    - You should not allow a POJO to be annotated with both @Statefull and @Stateless
Constraint Satisfaction

- This is anything but simple
  - A Constraint Satisfaction Problem
    - Can be resolved by multiple algorithms (Backtracking, Constrain reparation, and local search)
    - These algorithms are out of the scope of this presentation
Validating the Metamodel

- The imperative way
  - In Java
  - A jungle of conditional statements
Validating the Metamodel

- The logical way (Logic Programming)
  - Semantics are represented declaratively as predicates (Facts and Rules)
  - Procedurally interpreted query resolution
  - Constraint Logic Programming (An extension of Logic Programming)
    - Used for digital circuit verification
- Prolog is King
  - tuProlog: Implementation of the Prolog interpreter in Java
    - http://alice.unibo.it/xwiki/bin/view/Tuprolog/
  - Jlog: Implementation of the Prolog interpreter in Java
    - http://jlogic.sourceforge.net/
  - JPL: Java Interface to SWI Prolog
- Yeah you’re gonna have to learn that
Validating the Metamodel

- **Clojure**
  - Lisp-like language on the JVM
  - core.logic supports logic and constraint programming

- **Other**
  - Prova Rule Language
    - [http://www.prova.ws/](http://www.prova.ws/)
  - Mercury
Validating the Metamodel

- Rules Engine
  - Drools Expert
    - A highly performance optimized rules engine
    - Rules are written in mvel (A powerful expression language) or XML
    - Integrates with Java collaborator POJOs
Interpreting the Metamodel
Transformations

- Changing the structure of existing code
  - AST (Abstract Syntax Tree) rewriting
  - Adding/removing behavior
  - Automatic generation of cookie-cutter code
  - Create complex class hierarchies

- Notable byte code manipulation libraries
  - ASM
  - GCLIB
  - Javassist
  - Etc…
Transformations

- **ASM**
  - Small
  - Fast
  - Low-level
  - Require a good understanding of byte-code and Java Language Spec
  - Good documentation
  - [http://asm.ow2.org/](http://asm.ow2.org/)

- **GCLIB**
  - Built on top of ASM
  - High performance
  - Issues with signed jars
  - Dropped by Hibernate in favor of Javassist for lack of support and active
Transformations

- Javassist
  - Very easy to use
  - Works with signed jars
  - Good documentation
  - Excellent Javadoc
  - Adopted by hibernate
  - Slower than GCLIB
  - [http://www.jboss.org/javassist/](http://www.jboss.org/javassist/)
Weaving in Cross-Cutting Concerns

- OOP creates a hierarchical object model by nature
- Certain things are not necessary a part of the application logic
  - Logging
  - Transactions
  - Security
  - Etc...
- AOP (Aspect Oriented Programming)
  - Code weaving
  - Dynamic Proxies
Mapping

- Map one object to another based on the metamodel
- Example
  - Dozer
    - http://dozer.sourceforge.net/
  - ORM Frameworks
Mapping

Metadata

POJO Class

Metamodel

POJO Instance

Mapper

Row Class

Row Instance

Creates
Other

- Inversion of Control
  - Dependency Injection (JSR 330)
- Instrumentation
  - Monitor and measure performance
  - Diagnostics
  - Trace info
  - Etc…
- Etc…
Metaprogramming in the Wild
Metaprogramming in the Wild

- **Spring**
  - Lightweight POJO-based frameworks
  - Non-invasiveness
    - Stereotyping a class in lieu of inheritance to acquire behavior
    - Arbitrary methods signature instead of overriding inherited ones
  - Declarative configuration
    - Dependency Injection

- **Rails**
  - Rapid Development
  - Convention over configuration
    - Active Record Pattern

- **Hibernate/JPA (JSR 317)**
  - ORM

- **Guice**
- **Spring AOP**
- **Etc…**
Model-Driven Engineering

- Philosophy
  - In an application
    - Artifacts that are good candidates for auto-generation
    - Why?
      - The architecture is defined by
        - Making abstractions guided by the separation of concerns
      - Most deal with the same concerns
        - Avoid rediscovering the best approach every time
    - Code generation, Scaffolding, etc…
Project Averroes
Averroes

- A metaprogramming framework
  - Auto-discovery of annotations
    - @Discoverable
  - Annotation processing
  - Metamodel construction
  - Metamodel validation
    - Annotations of annotations (meta-metadata)
      - @AccompaniedWith
      - @AccompaniedWithAtLeastOne
      - @ForType
      - @ForTypeAnnotatedWith
      - @ForTypeIncluding
      - @NotAccompaniedWith

- A work in progress…
  - https://github.com/PolymathicCoder/Averroes
Questions?
Thank You!

@PolymathicCoder