Meta-Programming in Java: Reflection

Dr. Giuseppe Di Fatta
Associate Professor of Computer Science
Web: http://www.personal.reading.ac.uk/~sis06gd/
Email: G.DiFatta@reading.ac.uk
Director of the MSc Advanced Computer Science
http://www.reading.ac.uk/sse/pg-taught/sse-mscadvancedcomputerscience.aspx

These lecture slides are available at:
http://www.personal.reading.ac.uk/~sis06gd/resources.html
Overview

- Introduction to Meta-Programming
  - programs can represent and manipulate other programs or even themselves

- Meta-Programming mechanisms in Java
  - Generics
  - Reflection

- Applications of Java Reflection
- KNIME: an Eclipse plugin

(original slides from: Kai Koskimies, Rajini Sivaram, Mika Haapakorpi, Giuseppe Di Fatta)
Generics

• Abstraction over types

• Generics are one of the new language features in J2SE 1.5

• Resembles the template mechanism in C++
  – But Generics are NOT templates
Generics

• The generics can be used in classes, interfaces, methods and constructors.

• Two new types:
  – Parameterized types
  – Type variables

• A type variable is an unqualified identifier.

• Class and interface declarations can have type arguments (type variables)
  – Metalevel: Class implements GenericDeclaration

• Method and constructors definitions can have type arguments (type variables)
  – Metalevel: Method, Constructor implements GenericDeclaration
Interface `<Type>` and class `<Class>`

```
«interface»
Annotation

*   1

«interface»
AnnotatedElement
+getAnnotation(in annotation Type : Class<T>)
+getAnnotations()
+getDeclaredAnnotations()
+isAnnotationPresent()

«interface»
Type

«interface»
GenericType
+getGenericComponentType() : Type

«interface»
GenericArrayType
+getGenericComponentType() : Type

«interface»
GenericDeclaration
+getTypeParameters() : TypeVariable[]

«interface»
TypeVariable
+getBounds() : Type[]
+getGenericDeclaration() : GenericDeclaration
+getName() : String

«interface»
WildcardType
+getLowerBounds() : Type[]
+getUpperBounds() : Type

«interface»
ParameterizedType
+getActualTypeArguments() : Type
+getOwnerType() : Type
+getRawType() : Type
```

Java, Dr. Giuseppe Di Fatta, 2007-2013
Examples

- Interface
- A Class as type argument

\[ \text{List<String>} \ anExample = \text{new ArrayList<String>()} \]

- A parametrized type

- Class
- A Class as type argument

- A constructor of a parametrized type

\[ \text{Type aType} = \text{anExample.getClass();} \]
\[ \text{if( aType == ArrayList.getClass() ) ...} \]

- A type variable
- A type constant
Generic Methods: Example

static <T> void fromArrayToCollection(T[] anArray, Collection<T> aColl) {
    for (T anElement : anArray)
        aColl.add(anElement);
}

Object[] objArray = new Object[100];
Collection<Object> objColl = new ArrayList<Object>();

Number[] numArray = new Number[100];
Collection<Number> numColl = new ArrayList<Number>();

String[] stringArray = new String[100];
Collection<String> stringColl = new ArrayList<String>();

fromArrayToCollection(objArray, objColl);   // T inferred to be Object
fromArrayToCollection(numArray, objColl);   // T inferred to be Object
fromArrayToCollection(numArray, stringArray); // compile-time error
Metaclasses

• The Class<T> metaclass is parameterized over the generic TypeVariable T.
  – T represents any class or interface type i.e. the actual class, which is the instance of the metaclass Class<T>.
    • E.g. T cast(Object o)

• Class<T> itself is a represented as a class and also has a corresponding metaclass Class<Class>
How to Reference a Class

• The JRE allows 4 ways to reference a class
  – The class’ class definition
  – Class literals (“String.class”)
  – The `instanceof` keyword
  – Reflection

• Reflection is the only pure runtime way:
  – Provides full access to the object’s capabilities
  – Provides runtime capabilities not otherwise available
  – Improves the quality of an application
  – Extends the power of the classic object-oriented design patterns
Reflection

- Reflection
  - The ability of a program to examine itself and modify its structure or behaviour at run-time

- Two types of reflection
  - **Introspection**
    - Ability to examine meta-level information about the program structure itself at runtime
  - **Intercession**
    - Mechanisms to change the program interpretation or meaning at runtime

- What for?
  - To write flexible software that can adapt to changing requirements
Reflection

• A system is *reflective* if it can inspect part of its execution state while it is running.
  – *Introspection* only reads internal state, without modifying it
  – *Intercession* enables modifying execution state, and thereby changing system semantics
Java Reflection

• Limited dynamic reflection
  – Java implements a mild form of reflection

• Introspection
  – discover information about loaded classes and use them:
    • Class (retrieve or load)
    • Interfaces (list or load)
    • Methods and Constructors (list or invoke)
      – construct new class instances and arrays
    • Fields (list or get/set), access and modify elements of arrays
    • Generics information
    • Metadata annotations
    • Call stack (retrieve indirectly through a Throwable)

• Flexible, but secure (using Java Security Manager)
Java Reflection API

• The Java Reflection API consists of:
  – The class `java.lang.Class`
  – The interface `java.lang.reflect.Member`
  – The class `java.lang.reflect.Field`
  – The class `java.lang.reflect.Method`
  – The class `java.lang.reflect.Constructor`
  – The class `java.lang.reflect.Array`
  – The class `java.lang.reflect.Modifier`
  – The class `java.lang.reflect.InvocationTargetException`
Java Reflection

- Applications getting run-time information about objects, use:
  - `getField[s]`
  - `getMethod[s]`
  - `getConstructor[s]`

- Applications getting compile-time information about objects (at the level provided by `.class` files), use:
  - `getDeclaredField[s]`
  - `getDeclaredMethod[s]`
  - `getDeclaredConstructor[s]`
import java.lang.reflect.*;

// Obtain information about an object or class
Class c = obj.getClass();
Class superClass = c.getSuperclass();
Class[] interfaces = c.getInterfaces();
Field[] fields = c.getFields();
Method[] methods = c.getMethods();

// Create an object of a named class (eg. if classname not known till runtime)
Class cls = Class.forName("example.shapesRectangle");
Object r = cls.newInstance();

// Retrieve or set a field
Field widthField = cls.getField("width");
widthField.set(r, 200);
System.out.println(widthField.get(r));

r = new Rectangle();
r.width = 200;
System.out.println(r.width);
Example: method invocation

Method method = cls.getMethod("area");
long area = (long) (Long)method.invoke(r);
Applications of Java Reflection

- Annotations
- JUnit Tests
- Serialization
- Design Patterns
- Plugins (e.g., Eclipse Plugins)
Annotations

- Annotations
  - Developers can define custom annotation types
  - Using these types developers can annotate
    - fields,
    - methods,
    - classes,
    - and other program elements.

- Development tools can read these annotations (from source files or class files) and generate new artifacts accordingly
  - Source files
  - Configuration files
  - XML documents

- JavaDoc is based on a previous annotation-like mechanism (e.g. @author, @deprecated, @version)
JUnit Tests

- JUnit defines hierarchy of test suites
- Developer tests (White-box tests) – typically run regularly as part of build
- Eclipse Plugin
  - New -> JUnit Test Case
  - Run As -> JUnit Test

- Automated test harness executes all test methods in test suite
- Test harness uses reflection to find and execute test methods
- Based on `@Test` annotations (earlier versions used method names starting with test).
JUnit Test - example

import org.junit.*;

public class QueueTest {
    private Queue queue;

    @Before public void setUp() throws Exception {
        queue = new Queue();
    }
    @After public void tearDown() throws Exception {
    }

    @Test public void testPush() {
        String testString = "testString";
        queue.push(testString);
        Assert.assertEquals(queue.pop(), testString);
    }

    @Test public void testPop() {
    }
    @Test public void testEmpty() {
    }
    @Test public void testFull() {
    }
}

Java, Dr. Giuseppe Di Fatta, 2007-2013
Serialization

Store and retrieve Java objects in a serialized form

• java.io.Serializable interface

```java
FileOutputStream fileOut = new FileOutputStream("traceFile");
ObjectOutputStream out = new ObjectOutputStream(fileOut);
out.writeObject(new Date());

....

FileInputStream fileIn = new FileInputStream("traceFile");
ObjectInputStream in = new ObjectInputStream(fileIn);
Date date = (Date)in.readObject();
```

• By default, all non-transient, non-static fields of objects are serialized
• Default serialization and deserialization use reflection to recursively create object tree
Design Patterns

• Proxy
• Factory
  – Create objects without hardcoding concrete classes
• Delegation, Facade
• JavaBean
  – A Java Bean is a reusable software component that can be manipulated visually in a builder tool.
  – Supports introspection, customization, events, properties and persistence
Example: Factory without Reflection

```java
public static Student getStudent (String studentType) {
    Student theStudent;

    if(studentType.equals("Student"))
        theStudent = new Student();
    else if(studentType.equals("GradStudent"))
        theStudent = new GradStudent();
    else if(studentType.equals("UGradStudent"))
        theStudent = new UGradStudent();
    //else if... add more when necessary
    else ... //not supported studentType

    return theStudent;
}
```
Example: Factory with Reflection

```java
class Student {
    // Implementation of Student class
}

public static Student getStudent(String studentType) {
    Student theStudent;

    try {
        theStudent = (Student) Class.forName(studentType).newInstance();
    }
    catch (Exception e) {
        // not supported studentType
    }

    return theStudent;
}
```
Plugins (eg. Eclipse Plugins)

- Plugin architecture
- Plugins are loaded when required
- Plugin manifest file (plugin.xml)
- Class loader, dynamic class loading, reflection
Reflection - Summary

• Reflection allows to access and manipulate the representation and the state of the program in the Java Virtual Machine at runtime

• Code based on reflection
  – is more complex
  – is harder to debug
  – may have a performance impact

• Used extensively in componentised software
  – Many programming toolsets for Java rely on Reflection

• Software becomes “softer”
  – flexible
  – extensible
  – pluggable
KNIME – an Eclipse Plugin

- Introduction to the KDD process and tools
- Flow-based programming
- KNIME
KDD Development Environments

- Increasing demand for integrated environments to facilitate the Knowledge Discovery in Databases (KDD) process
  - Data Analytics and Data Mining

- **Workflow management tools** that integrates analytical data mining methods for prediction, discovery, classification, etc., with data management and information visualization.

An Overview of the Steps That Compose the KDD Process

[Diagram showing the steps of the KDD Process]

Java, Dr. Giuseppe Di Fatta, 2007-2013

28
Data analysis/mining tools popularity

Data mining/analytic tools reported in use on Rexer Analytics survey during 2009

Results of the 2011 KDnuggets poll on data mining software

Which data mining/analytic tools you used in the past 12 months for a real project (not just evaluation) [1103 voters]

<table>
<thead>
<tr>
<th>Tool</th>
<th>% users in 2011</th>
<th>% users in 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>RapidMiner (305)</td>
<td>27.7%</td>
<td>37.8%</td>
</tr>
<tr>
<td>R (257)</td>
<td>23.3%</td>
<td>29.8%</td>
</tr>
<tr>
<td>Excel (240)</td>
<td>21.8%</td>
<td>24.3%</td>
</tr>
<tr>
<td>SAS (150)</td>
<td>13.6%</td>
<td>12.1%</td>
</tr>
<tr>
<td>Your own code (134)</td>
<td>12.1%</td>
<td>18.4%</td>
</tr>
<tr>
<td>KNIME (134)</td>
<td>12.1%</td>
<td>19.2%</td>
</tr>
<tr>
<td>Weka (Pentaho) (130)</td>
<td>11.8%</td>
<td>14.4%</td>
</tr>
<tr>
<td>Salford (117)</td>
<td>10.6%</td>
<td>1.6%</td>
</tr>
<tr>
<td>Statistica (94)</td>
<td>8.5%</td>
<td>6.3%</td>
</tr>
<tr>
<td>IBM SPSS Modeler (91)</td>
<td>8.3%</td>
<td>7.3%</td>
</tr>
<tr>
<td>MATLAB (79)</td>
<td>7.2%</td>
<td>9.2%</td>
</tr>
<tr>
<td>IBM SPSS Statistics (79)</td>
<td>7.2%</td>
<td>7.9%</td>
</tr>
<tr>
<td>SAS Enterprise Miner (78)</td>
<td>7.1%</td>
<td>5.5%</td>
</tr>
<tr>
<td>JMP (63)</td>
<td>5.7%</td>
<td>5.7%</td>
</tr>
</tbody>
</table>

Java, Dr. Giuseppe
Flow-based Programming

Flow-based Programming (FBP) is a programming paradigm that defines applications as networks of "black box" processes, which exchange data across predefined connections by message passing, where the connections are specified externally to the processes. These black box processes can be reconnected endlessly to form different applications without having to be changed internally. FBP is thus naturally component-oriented.
• Developed at the **ALTANA-Chair for Bioinformatics and Information Mining**, Department of Computer and Information Science, University of Konstanz, Germany

• Under continuous evolution and extension
  – April 2006: 1\textsuperscript{st} release
  – March 2014: Version 2.9.2
  – Community contributions and meetups

First publication on KNIME in 2006:

An Eclipse Plugin: KNIME

KNIME: Interactive Data Exploration

Features:
- Modular Data Pipeline Environment
- Large collection of Data Mining techniques
- Data and Model Visualizations
- Interactive Views on Data and Models
- Java Code Base as Open Source Project
- Seamless Integration: R Library, Weka, etc.
- Based on the Eclipse Plug-in technology

Easy extendibility
New nodes via open API and integrated wizard
Node Model
Conclusions on KNIME

• KNIME, since 2006:
  – Still open source, enhanced GUI, many more modules and features
  – Commercial extensions: e.g., server-based version

• Modularity and extendibility
  – General and extendible data structure (DataTable and DataCell)
  – Nodes encapsulate computational processing tasks (algorithms)
  – Extensions based on Eclipse plugin framework (in Java): new KNIME node wizard helps in the task.

• A workflow management system
  – directed edges connects nodes to create data pipelines
  – a workflow is, in general, a directed acyclic graph
  – multi-threading
  – Meta-nodes (nested workflows)
Masters module dedicated to KNIME and R:
- “Data Analytics and Mining” (SEMDM13) - MSc Advanced Computer Science
  http://www.reading.ac.uk/sse/pg-taught/sse-mscadvancedcomputerscience.aspx

KNIME user → KNIME desktop version (only KNIME):
- http://www.knime.org/knime-desktop
- http://www.knime.org/downloads/datasets
- http://www.knime.org/introduction/examples

KNIME developer → KNIME SDK version (an Eclipse distro):
- http://tech.knime.org/developer-guide
- http://tech.knime.org/developer/example
- API: for example see the DataTable interface in
  http://tech.knime.org/docs/api/org/knime/core/data/package-summary.html

Dr. Rosaria Silipo’s blog with lots of resources on KNIME:
- http://www.dataminingreporting.com/