Reflection
(in fact, Java introspection)

Prof. Dr. Ralf Lämmel
Universität Koblenz-Landau
Software Languages Team

Elevator speech

So *programs are programs* and *data is data*.

However, programs can be represented as data*!*

Programs may process data representing programs*!*

Those are *meta*programs.

The activity is called *metaprogramming*.

Reflection is an important form of *metaprogramming:* a program examines or modifies its own structure and behavior at runtime.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Regular construction and invocation

```
package helloworld;
Exemplar obj = new Exemplar();
obj.run();
```

Reflection (introspection) would mean that:

- ‘Exemplar’ is encoded as String (i.e., ‘data’).
- ‘run’ is encoded as String (i.e., ‘data’).
- Both of them would need to be ‘looked up’.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Introspection-based construction and invocation

```
Class clss1 = Class.forName("helloworld.Exemplar");
Constructor cons1 = clss1.getConstructor();
Object obj1 = cons1.newInstance();
Method meth = clss1.getMethod("run");
meth.invoke(obj1);
```

See package **helloworld** of
https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javaReflectionSamples

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

See package **helloworld** of
https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javaReflectionSamples

‘Hello world*!*’ with reflection

DEMO

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Examples of metaprograms

- A pretty printer for Java.
- A Java compiler because:
  - it inspects Java code, and
  - it produces byte code.
- The JUnit framework because it examines a Java program to locate and execute test methods.This is a case of reflection (introspection).
- …

Summary of terminology

- A ***metaprogram*** **i**s a program that manipulates programs to which we refer as **object programs**. That is, object programs correspond to data of metaprograms.
- The language in which metaprograms are written is called the ***metalanguage***. The language of object programs is called the ***object language***.
- If a program examines or modifies its own structure and behavior at runtime, then it is a **reflective** program, i.e., object program and metaprogram coincide.
- If reflection is limited to “read access”, then this is called ***introspection***. This is the Java situation.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Java’s reflection API

See the Eclipse project
https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javaReflectionSamples

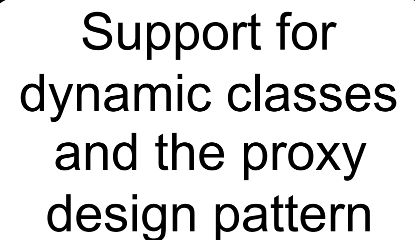
Java's reflection API

Types covered today

- java.lang.Class
- java.lang.reflect.Field
- java.lang.reflect.Method
- java.lang.reflect.Constructor

Additional type

- java.lang.reflect.Proxy



(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

The class of **classes**

The class of **constructors**

Class **lookup**

```
Class clss1 = Class.forName("helloworld.Exemplar");
Constructor cons1 = clss1.getConstructor();
Object obj1 = cons1.newInstance();
Method meth = clss1.getMethod("run");
meth.invoke(obj1);
```

Constructor **lookup**

The class of **methods**

Method **invocation**

Constructor **invocation**

Method **lookup**

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

java.lang.Class

- Represents classes (and interfaces) in a Java app.
- Primitive types have an associated class.
- Even void has an associated class.
- Class *cannot be regularly instantiated*.
- Instead the class loader uses defineClass:
  - Input: byte code
  - Output: an instance of type Class
- Each object knows of its class: **`obj.getClass`**()
- Each class knows of its class: **String.class**

To treat argument and result types homogeneously.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

java.lang.Class interface

- Map strings to class objects (forName)
- **Getters**
  - Name
  - Constructors
  - Fields
  - Methods
  - Interfaces
  - Annotations
  - Package
  - Superclass
  - Enclosing class
- **NO SETTERS**
- Various other inspectors ("tests")
- Instantiation (but see constructors)
- Cast operations
- …

Everything you ever wanted to know about a class.

We cannot change classes. Such limited reflection is called introspection.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

java.lang.reflect.Method

- Represents methods
  - … of classes and interfaces
  - static, initialization, instance, and abstract methods
- Constructors are treated very similar; see:
  - java.lang.reflect.Constructor
- Does not provide byte-code access.
- Returned by getters on java.lang.Class.
- **NO METHOD BODIES**

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

java.lang.reflect.Method interface

- **Getters**
  - Name
  - Parameter types
  - Return type
  - Type parameters
  - Class
  - Modifiers
  - Annotations
  - Parameter annotations
  - …
- ***Invocation***

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

java.lang.reflect.Field

- Provides information about fields.
- Provides dynamic access to fields.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

java.lang.reflect.Field interface

- Getters
  - Name
  - ***Value*** (for a given object)
  - Modifiers
  - Annotations
  - …
- Setters
  - ***Value*** (for a given object)

Reflection examples

See the Eclipse project

https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javaReflectionSamples

Object dumping

```
public class Tree {
        public int info;
        public Tree left, right;
}

Tree tree = new Tree();
tree.info = 1;
tree.left = new Tree();
tree.left.info = 2;
tree.right = new Tree();
tree.right.info = 3;
tree.right.left = new Tree();
tree.right.left.info = 4;
Dumper.dump(tree);
```

```
dumper.Tree
-int info = 1
-dumper.Tree left =
--int info = 2
--dumper.Tree left = NULL
--dumper.Tree right = NULL
-dumper.Tree right =
--int info = 3
--dumper.Tree left =
---int info = 4
---dumper.Tree left = NULL
---dumper.Tree right = NULL
--dumper.Tree right = NULL
```

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

See package **dumper** of
https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javaReflectionSamples

Object dumping

DEMO

Scenario

- Given an object of an arbitrary type.
- Provide a deep dump on the state of the object.
- That is, go over the fields recursively.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Object walking

```
Company c = ...;
double before = total(c);
cut(c);
double after = total(c);
assertEquals(before / 2.0d, after, 0);
```

Scenario

- We want to **total** without writing specific code walking the object.
- Ditto for **cutting** salaries.
- Thus, introspection provides generic walking.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

http://101companies.org/wiki/Contribution:javaReflection

Object walking

DEMO

Please note the conciseness of the code compared to javaComposition, for example.

A challenge for (Java) reflection

Given a class, what are the subclasses of it?
Alas, there is no such member on “Class”.

The set of classes known to the system is simply the set of classes loaded so far. One could assume a scan over the full classpath to get to know more classes, but automatically loading all these classes is expensive and may not be intended.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

See package **packagelist** of
https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javaReflectionSamples

```
/**
* Find (and load) all classes in a given package.
* The search relies on a directory listing.
*/
public static Iterable<Class<?>> getClassesInPackage(Package p) {
   LinkedList<Class<?>> l = new LinkedList<Class<?>>();
   String name = p.getName();
   name = name.replace('.',File.separatorChar);
   File dir = new File(name);
   for (String f : dir.list())
      if (f.endsWith(".class")) {
         String classname = f.substring(0,f.length()-6);
         try {
            Class<?> clss = Class.forName(p.getName() + "." + classname);
            l.add(clss);
         }
         catch (ClassNotFoundException e) { ... }
      }
   return l;
}
```

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Metadata

Metaprograms vs. metadata

- ***Metaprograms*** are programs that generate, analyze, or control other programs.
- ***Metadata*** (in the narrow context of metaprogramming) is <u>data that is attached to programs to control metaprograms</u>.
- In Java, we use ***annotations*** for metadata.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Sample annotation

```
@Test
public void testTotal() throws Exception {
   Document doc = DOMUtilities.loadDocument(sampleCompany);
   double total = total(doc);
   assertEquals(399747, total, 0);
}
```

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Annotations

Data associated with methods, fields, class, etc. that does not affect the semantics of a program, but controls metaprograms over the annotated object programs.

Scenarios:

- Identify test methods and control execution (JUnit)
- Control serialization of objects as XML (JAXB)
- ... many other frameworks ...

.NET uses the term “custom attributes” instead.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Sample annotation

```
@RequestForEnhancement(
   id = 2868724,
   synopsis = "Enable time-travel",
   engineer = "Mr. Peabody",
   date = "4/1/3007"
)

public void travelThroughTime(Date destination) { ...
}
```

“=“ notation can be omitted if there is only a single component of name ‘value’.

Source: http://docs.oracle.com/javase/7/docs/technotes/guides/language/annotations.html

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Annotation declaration for sample annotation

```
/**
 * Describes a Request-For-Enhancement (RFE)
 * for the API element at hand
 */

public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date() default "[unimplemented]";
}
```

Source: http://docs.oracle.com/javase/7/docs/technotes/guides/language/annotations.html

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Summary of rules for annotation declarations

- An annotation declaration takes the form of a special interface declaration:
  - it starts with an 'at' sign @;
  - method declarations must not have any parameters;
  - … and must not have any throws clauses;
  - return types of the method must be one of the following:
    - primitive types;
    - String;
    - Class;
    - Enum;
    - array types of the above types;
  - there may be a default for each method.

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Meta-annotations

(Annotations for annotation declarations)

Declaration of a JUnit-like @Test annotation
with meta-annotations

```
/**
* An annotation to tag methods as test methods.
* The annotations of the annotation carry the following meaning:
* @Retention implies that the annotation has to be maintained until runtime.
* @Target means that the annotation is admitted on methods only.
*/
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test { }
```

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

See package **junitlight** of
https://github.com/101companies/101repo/tree/master/technologies/Java_platform/samples/javaReflectionSamples

A tiny approximation of JUnit

DEMO

Scenario
- Take a class name as argument.
- Find all “test” methods in the class.
- Execute them.
- Keep a record of success and failure.

More metaprogramming power

- Byte-code engineering with ASM:
  - http://asm.ow2.org/
- Aspect-oriented programming with AspectJ:
  - https://eclipse.org/aspectj/
- Parsing Java with javaParser:
  - http://javaparser.org/
- Reflection in Python:
  - https://en.wikibooks.org/wiki/Python_Programming/Reflection

(C) 2010-2017 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)