



## Reflection

### *Java™ Technology's Secret Weapon*

**Steve Odendahl**  
Member of Technical Staff  
Global eServices Engineering  
Sun Microsystems, Inc.

Session 2289

## Goal

Learn the What, How, When, Why and Where of the Java™ Reflection API



2 | Session 2289

## Learning Objectives

When we are done, you will be able to:

- Use the Reflection API in your own code
- Understand the engineering tradeoffs in using Reflection
- Apply some common Reflection Patterns
- Avoid inappropriate use of Reflection



3 | Session 2289

## Speaker's Qualifications

Steve Odendahl has been employed as a Java™ technology applications programmer since the exciting early days of the JDK™ 1.02 release

As a member of the Global eServices Engineering team in Enterprise Services, developing Java technology-based solutions for service and support, he takes an interest in writing effective, efficient, and maintainable Java code



4 | Session 2289

## What Is Reflection?

Reflection is Java™ technology's **crowbar**—a blunt instrument for dirty jobs that can't be done any other way.



5 | Session 2289

## Agenda

- Introduction to the Reflection API
- Engineering tradeoffs
- Reflection patterns
- Real World Reflection
- When not to use Reflection



6 | Session 2289

## Introduction to the Reflection API



Session 2289

## Introduction to the Reflection API

What can Reflection do?  
The classes in the Reflection API  
Sample code using the API



8 | Session 2289

## What Can Reflection Do?

Provide runtime information on the fields and methods of a class



9 | Session 2289

## What Can Reflection Do?

Provide runtime information on the fields and methods of a class

Instantiate objects and arrays given the class name



10 | Session 2289

## What Can Reflection Do?

Provide runtime information on the fields and methods of a class

Instantiate objects and arrays given the class name

Invoke static and instance methods given the method name



11 | Session 2289

## What Can Reflection Do?

Provide runtime information on the fields and methods of a class

Instantiate objects and arrays given the class name

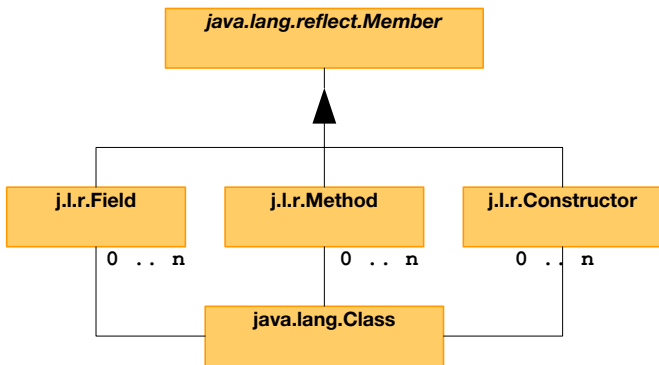
Invoke static and instance methods given the method name

Create a class at runtime that implements one or more interfaces



12 | Session 2289

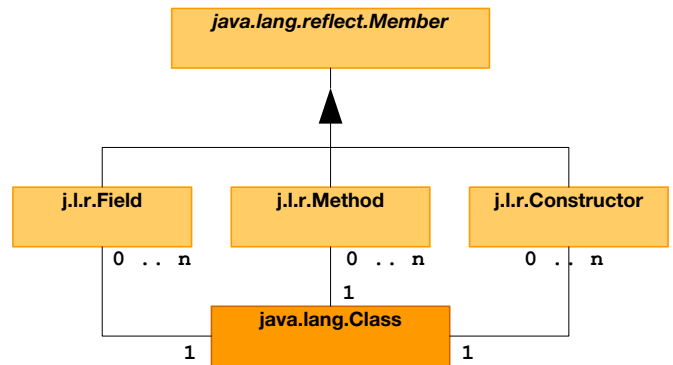
## The Reflection API – Major Players



13 | Session 2289



## The Reflection API – Class

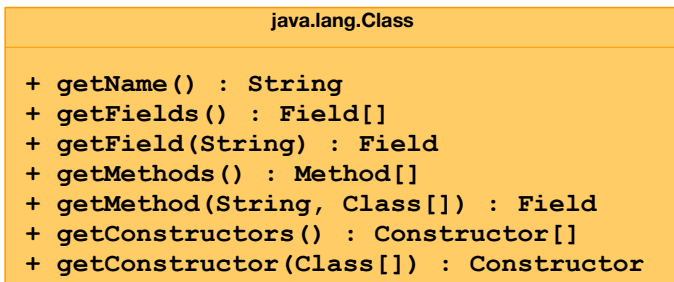


14 | Session 2289



## The Reflection API – Class

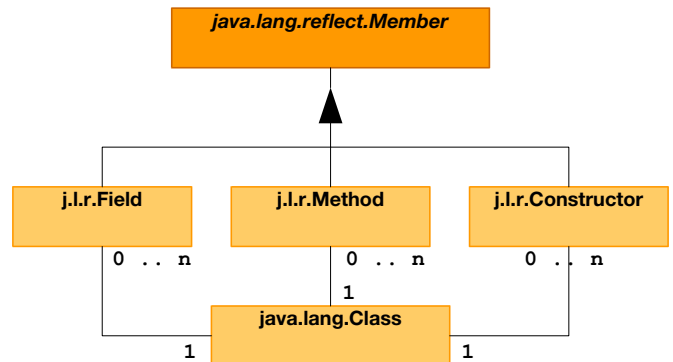
Entry point to the Reflection API



15 | Session 2289



## The Reflection API – Member

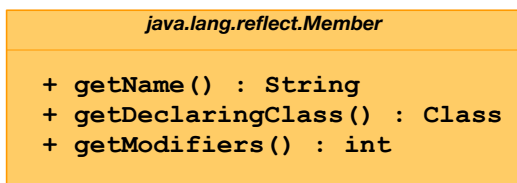


16 | Session 2289



## The Reflection API – Member

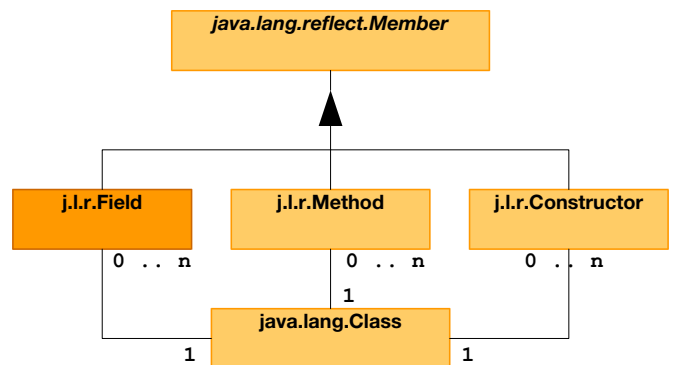
Name (of the field or method)  
 Modifiers (public, static, transient...)



17 | Session 2289



## The Reflection API – Field



18 | Session 2289



## The Reflection API—Field

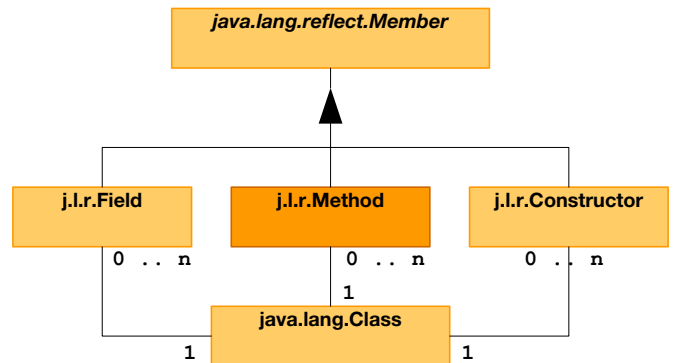
Get and set the value of the field

```
java.lang.reflect.Field  
  
+ get(Object) : Object  
+ set(Object, Object)  
+ getInt(Object) : int  
+ setInt(Object, int)  
...  

```



## The Reflection API—Method



## The Reflection API—Method

Call the method—`invoke(target, args)`

Static method—`invoke(null, args)`

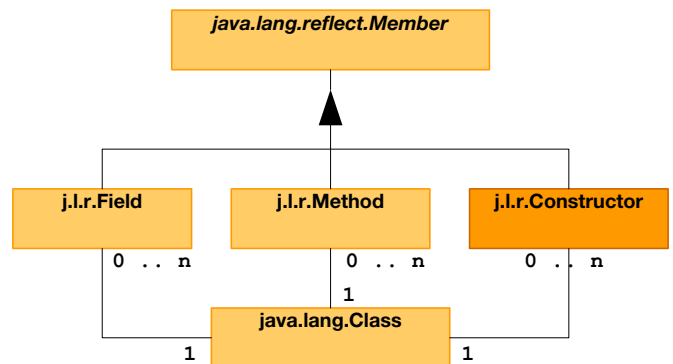
No arguments—`invoke(target, null)`

```
java.lang.reflect.Method  
  
+ invoke(Object, Object[]) : Object  
+ getParameterTypes() : Class[]  
+ getExceptionTypes() : Class[]  

```



## The Reflection API—Constructor



## The Reflection API—Constructor

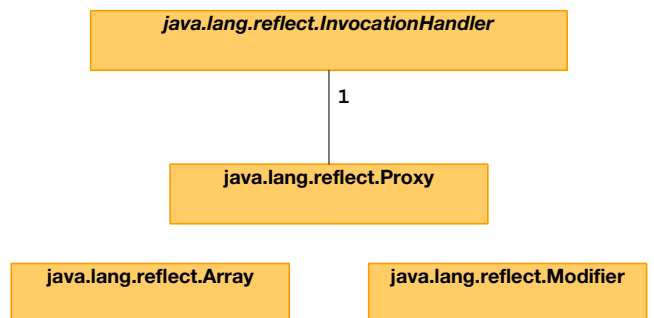
Create an instance—`newInstance(args)`

```
java.lang.reflect.Constructor  
  
+ newInstance(Object[]) : Object  
+ getParameterTypes() : Class[]  
+ getExceptionTypes() : Class[]  

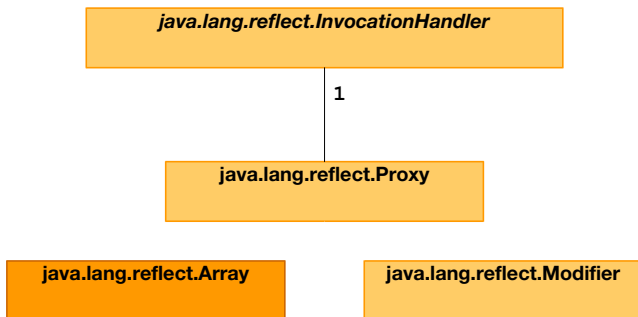
```



## The Reflection API—Minor Players



## The Reflection API—Array



25

Session 2289



## The Reflection API—Array

Create an array

Get/set items in the array

```
java.lang.reflect.Array

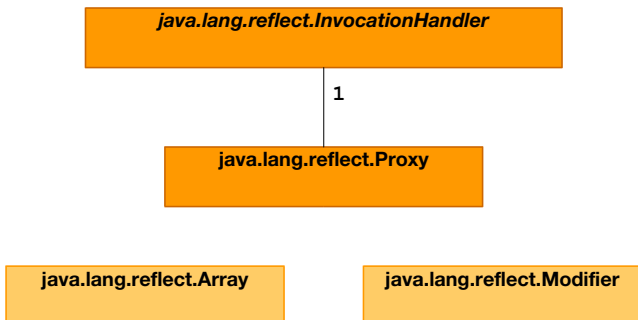
+ newInstance(Class, int) : Object[]
+ newInstance(Class, int[]) : Object[]
+ get(Object, int) : Object
+ set(Object, int, Object)
+ getInt(Object, int) : int
...
```

26

Session 2289



## The Reflection API—Proxy



27

Session 2289



## The Reflection API—Proxy

Create a **dynamic proxy**—an object that implements one or more interfaces

```
java.lang.reflect.Proxy

+ newInstance(ClassLoader,
              Class[], InvocationHandler) : Object

java.lang.reflect.InvocationHandler

+ invoke(Object, Method, Object[]) : Object
```

28

Session 2289



## It All Begins With Class

An instance of `java.lang.Class` for every reference type (classes, interfaces, and arrays)

An instance of `java.lang.Class` for every primitive type

Most reflective techniques start with retrieval of a `Class` instance

29

Session 2289



## How to...Get an Instance of Class

Use the class name

```
try {
    Class strcl = Class.forName(
        "java.lang.String");
    Class strarray = Class.forName(
        "[Ljava.lang.String;");
} catch (ClassNotFoundException cnfex) {
    // handle it
}
```

30

Session 2289



## How to...Get an Instance of Class

Use the class name

Use a class literal

```
Class strcl = String.class;
Class intcl = int.class;
Class strarray = String[].class;
```



31 | Session 2289

## How to...Get an Instance of Class

Use the class name

Use a class literal

Retrieve it from an instance

```
Class strcl = "hello".getClass ();
Class strarray =
    new String[] {}.getClass ();
```



32 | Session 2289

## How to...Get an Instance of Class

Use the class name

Use a class literal

Retrieve it from an instance

Use the constant defined in the wrapper class

```
Class intcl = Integer.TYPE;
```



33 | Session 2289

## How to...Create an Object

Use the Class object

```
try {
    Class foocl = Class.forName ("Foo");
    Foo f = (Foo) foocl.newInstance ();
} catch (ClassNotFoundException cnfex) {
    // handle it
} catch (InstantiationException iex) {
    // handle it
} catch (IllegalAccessException iex) {
    // handle it
}
```



34 | Session 2289

## Interlude—What Do I Do All About All Those !@#\$\$%^& Exceptions?

ClassNotFoundException occurs for two reasons

Misspelled the class name

In source code

In configuration file

Programmer error—rethrow as RuntimeException

```
catch (ClassNotFoundException cnfex) {
    throw new RuntimeException (cnfex);
}
```



35 | Session 2289

## Interlude—What Do I Do All About All Those !@#\$\$%^& Exceptions?

ClassNotFoundException occurs for two reasons

Misspelled the class name

In source code

In configuration file

System loads classes from variable locations

Treat like an IOException

Don't catch the exception—force higher-level code to handle it



36 | Session 2289

## How to...Create an Object

Use the Class object

Use a Constructor

```
try {
    Constructor c =
        foocl.getConstructor (null);
    Foo f = (Foo) c.newInstance (null);
} // catch blocks omitted
```



37 | Session 2289

## How to...Create an Array

Use the Array.newInstance method

```
try {
    Foo[] foos = (Foo[])
        Array.newInstance (foocl, 100);
} // catch blocks omitted
```



38 | Session 2289

## How to...Get a Method

Use the method name and parameters to retrieve it from the Class instance

```
Class[] paramTypes = { String.class };
Class cl = java.io.PrintWriter.class;
try {
    Method m = cl.getMethod
        ("println", paramTypes);
} // catch blocks omitted
```



39 | Session 2289

## How to...Get a Field

Use the field name

```
Class cl = System.class;
try {
    Field f = cl.getField ("out");
} // catch blocks omitted
```



40 | Session 2289

## How to...Get the Value of a Field

Use the Field instance

```
try {
    java.io.PrintWriter out =
        (java.io.PrintWriter) f.get (null);
} // catch blocks omitted
```



41 | Session 2289

## How to...Invoke an Instance Method

Use the Method object

```
Object[] params = { "Hello, world!" };
try {
    m.invoke (out, params);
} // catch blocks omitted
```



42 | Session 2289

## Putting It All Together...

### A familiar program

```
public static void main (String[] args)
    throws Exception {
    Field f = System.class.getField ("out");
    PrintStream out = (PrintStream) f.get (null);
    Class[] paramTypes = { String.class };
    Method m = PrintStream.class.getMethod
        ("println", paramTypes);
    String[] params = (String[]) Array.newInstance
        (String.class, 1);
    Array.set (params, 0, "Hello, world!");
    m.invoke (out, params);
}
```



43 | Session 2289

## Engineering Tradeoffs



Session 2289

## Engineering Tradeoffs

Readability  
Performance  
Code size



45 | Session 2289

## Readability

Obviously suffers in most cases

### Direct Code

```
public static void main (String[] args) {
    System.out.println ("Hello, world!");
}
```

### Reflective Code

```
public static void main (String[] args)
    throws Exception {
    Field f = System.class.getField ("out");
    PrintStream out = (PrintStream) f.get (null);
    Class[] paramTypes = { String.class };
    Method m = PrintStream.class.getMethod
        ("println", paramTypes);
    String[] params = (String[]) Array.newInstance
        (String.class, 1);
    Array.set (params, 0, "Hello, world!");
    m.invoke (out, params);
}
```



46 | Session 2289

## Readability

Obviously suffers in most cases

Reason enough to **avoid** use of Reflection in all but exceptional circumstances



47 | Session 2289

## Readability

Obviously suffers in most cases

Reason enough to **avoid** use of Reflection in all but exceptional circumstances

Alleviated by general utility methods

```
public static Method getMethod(Object obj, String str) {
    try {
        Class cl = obj.getClass ();
        return cl.getMethod (str, null);
    } catch (NoSuchMethodException nsmex) {
        throw new RuntimeException (nsmex);
    }
}
```



48 | Session 2289

## Readability

Obviously suffers in most cases

Reason enough to **avoid** use of Reflection in all but exceptional circumstances

Can be alleviated by utility methods

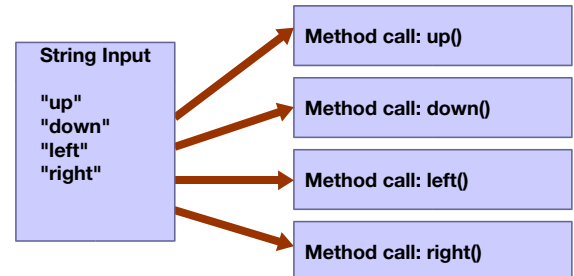
In certain circumstances, code employing Reflection can be **more readable** than non-reflective code



49 | Session 2289

## Readability – Example

Take various actions based on a String



50 | Session 2289

## Readability – Example

Take various actions based on a String

A straight-forward implementation

```
public void takeAction (String str) {
    if (str.equals ("up")) {
        up ();
    } else if (str.equals ("down")) {
        down ();
    } else if (str.equals ("left")) {
        left ();
    } else if (str.equals ("right")) {
        right ();
    }
}
```



51 | Session 2289

## Readability – Example

Difficult to read – difficult to extend

A reflective implementation

```
public void takeAction (String str) {
    Util.invoke (str, this, null);
}
```

Concise, easy to extend

Use utility methods to improve readability



52 | Session 2289

## Performance

Comparison of 2 scenarios...

Reflective

Cached reflective

On 3 platforms...

Solaris™ 8

Windows 2000

Windows CE

Using different Java™ virtual machines



53 | Session 2289

## Performance

Direct method invocation

```
for (int i = 0; i < ITERATIONS; i++) {
    incrementable.increment ();
}
```



54 | Session 2289

## Performance

Direct method invocation

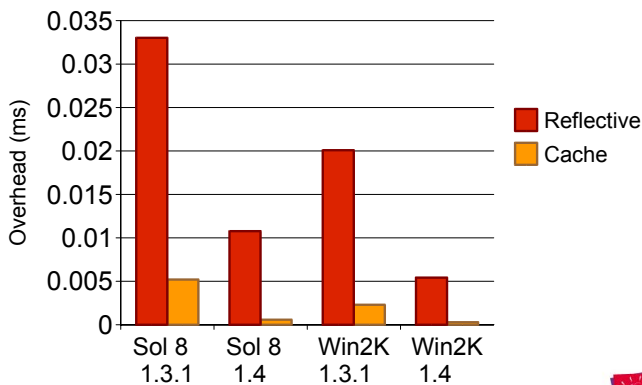
Method lookup and invocation

```
for (int i = 0; i < ITERATIONS; i++) {  
    Method m = cl.getMethod  
        ("increment", null);  
    m.invoke (incrementable, null);  
}
```



55 | Session 2289

## Performance—Results



57 | Session 2289

## Performance—Conclusion

Cache method and invoke **very competitive**, especially with 1.4

Method lookup and invoke has greater overhead, but can still be useful



59 | Session 2289

## Performance

Direct method invocation

Method lookup and invocation

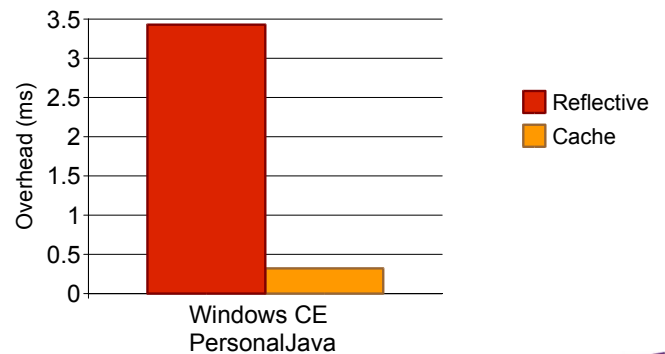
Cache method lookup and invoke

```
Method m = cl.getMethod ("increment", null);  
for (int i = 0; i < ITERATIONS; i++) {  
    m.invoke (incrementable, null);  
}
```



56 | Session 2289

## Performance—Results



58 | Session 2289

## Code Size

Example—AWT and JFC/Swing API development

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99



60 | Session 2289

## Code Size

Example—AWT and JFC/Swing API development

Implementation of ActionListener interfaces

```
public interface ActionListener {
    public void actionPerformed
        (ActionEvent event);
}
```



61 | Session 2289

## Code Size

Example—AWT and JFC/Swing API development

Implementation of ActionListener interfaces

Traditionally done through inner classes

```
button.addActionListener
    (new ActionListener () {
        public void actionPerformed
            (ActionEvent event)
        { doAction (); }
    });
```



62 | Session 2289

## Code Size

Example—AWT and JFC/Swing API development

Implementation of ActionListener interfaces

Traditionally done through inner classes

Results in many small classes



63 | Session 2289

## Code Size—Solution

Use a reflective adapter with a cached Method

```
public class ReflectiveAdapter
    implements ActionListener {

    private Object target;
    private Method method; // cached Method

    public ReflectiveAdapter
        (Object target, String action) {
        this.target = target;
        method = getMethod (target, action);
    }
    ...
}
```



64 | Session 2289

## Code Size—Solution

Implement the actionPerformed method

```
public class ReflectiveAdapter
    implements ActionListener {
    ...
    public void actionPerformed (ActionEvent evt) {
        try {
            method.invoke (target, null);
        } catch (Exception ex) {
            throw new RuntimeException (ex);
        }
    }
}
```

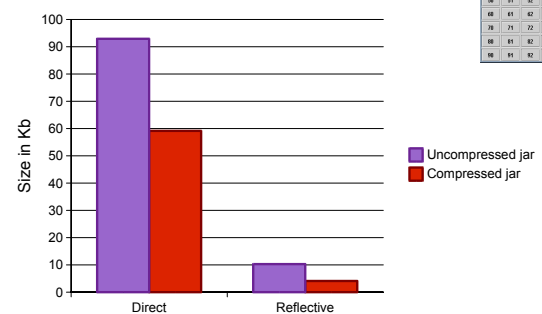
Result—one class, many instances



65 | Session 2289

## Code Size—Results

Simple GUI with 100 JButton



File	1	2	3	4	5	6	7	8	9
18	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99



66 | Session 2289

## Code Size – Conclusions

For a Java technology GUI, reflective adapters can result in

- Smaller jar files

- Faster startup

Could be useful on small devices



67 | Session 2289

## Reflection Patterns



Session 2289

## Reflection Patterns

- Factory Method

- Interpreter

- Double Dispatch

- Interposition



69 | Session 2289

## Factory Method

“Define an interface for creating an object, but let subclasses decide which class to instantiate” — **Design Patterns** — GOF

What if the creational method is static?



70 | Session 2289

## Factory Method – Example

Define an abstract base class Enum for defining type-safe enumeration classes

```
public abstract class Enum
    implements java.io.Serializable {

    public int getIndex () ...
    public String getName () ...
}
```



71 | Session 2289

## Factory Method – Example

Define an abstract base class Enum for defining type-safe enumeration classes

Convenient to have a bulk-creation static method

```
public static Enum[] create (String[] names)
```



72 | Session 2289

## Factory Method – Example

Define an abstract base class Enum for defining type-safe enumeration classes

Convenient to have a bulk-creation static method

Subclasses will be easy to define and populate

```
public class Direction extends Enum {
    public static final Direction[]
        DIRS = (Direction[]) create
            (new String[]
             {"Left", "Right", "Up", "Down"});
}
```



73 | Session 2289

## Factory Method – Solution

Pass in a reference to the Class instance for the subclass

```
public static Enum[]
    create (Class cl, String[] names)
```



75 | Session 2289

## Factory Method – Result

An Enum base class that makes it easy to define type-safe enumerated types

```
public class Direction extends Enum {
    public static final Direction[]
        DIRS = (Direction[]) create
            (Direction.class, new String[]
             {"Left", "Right", "Up", "Down"});
}
```



77 | Session 2289

## Factory Method – Problem

Static methods bound at compile-time

The create method does not know how to create instances of subclasses



74 | Session 2289

## Factory Method – Implementation

*(Exception handling omitted)*

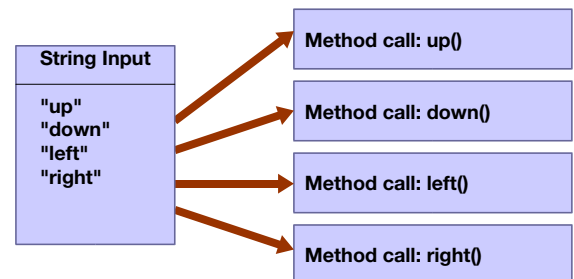
```
public static Enum[] create (Class cl,
    String[] names) {
    Enum[] enums = Array.newInstance
        (cl, names.length);
    for (int i = 0; i < enums.length; i++)
    {
        enums[i] = cl.newInstance ();
        enums[i].setName (names[i]);
        enums[i].setIndex (i);
    }
    return enums;
}
```



76 | Session 2289

## Interpreters

Recall Readability example



78 | Session 2289

## Interpreters – Example

Use as a basis for a simple graphics language

```
down 6
on
down 2
right 4
up 6
off
```



79 | Session 2289

## Interpreters – Example

Read and parse lines into { String, int[] }

```
down 4  → "down"  {4}
on      → "on"     {}
down 2  → "down"  {2}
right 4 → "right"  {4}
up 6    → "up"    {6}
off     → "off"   {}
```



80 | Session 2289

## Interpreters – Example

Look up method and invoke

```
down 4  → "down"  {4} → down (4) ;
on      → "on"     {} → on () ;
down 2  → "down"  {2} → down (2) ;
right 4 → "right"  {4} → right (4) ;
up 6    → "up"    {6} → up (6) ;
off     → "off"   {} → off () ;
```

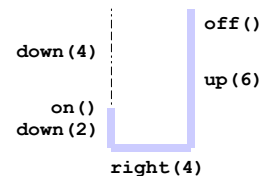


81 | Session 2289

## Interpreters – Example

Drawing is performed

```
down 4  → "down"  {4} → down (4) ;
on      → "on"     {} → on () ;
down 2  → "down"  {2} → down (2) ;
right 4 → "right"  {4} → right (4) ;
up 6    → "up"    {6} → up (6) ;
off     → "off"   {} → off () ;
```

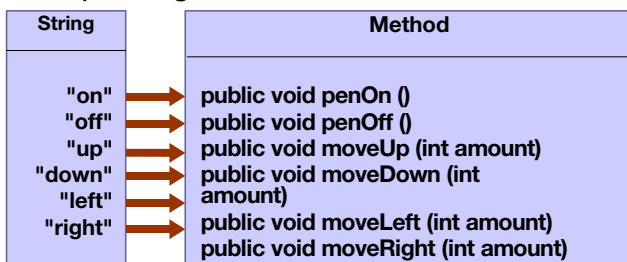


82 | Session 2289

## Interpreters – Refinement

String need not match method name

Map String => Method



83 | Session 2289

## Interpreters – Conclusion

See **Languages for the Java™ VM**

160 languages for the JVM™

Many use Reflection for implementation

<http://grunge.cs.tuberlin.de/~talk/vmlanguages.html>



84 | Session 2289

## Double Dispatch

Suppose we want a logging interface that will support various types of input

```
public interface Logger {
    public void log (Object obj);
}
```



85 | Session 2289

## Double Dispatch

A possible implementation—

```
public class LoggerImpl implements Logger {
    public void log (Object obj) ...
    private void log (String str) ...
    private void log (Date date) ...
    private void log (Exception ex) ...
}
```



86 | Session 2289

## Double Dispatch—Problem

Leads to chains of if ... else if

```
public void log (Object obj) {
    if (obj instanceof String) {
        log ((String) obj);
    } else if (obj instanceof Date) {
        log ((Date) obj);
    } else if (obj instanceof Exception) {
        log ((Exception) obj);
    }
}
```



87 | Session 2289

## Double Dispatch—Classic Solution

Define a loggable interface

```
public interface Loggable {
    public void logThis (Logger logger);
}
```



88 | Session 2289

## Double Dispatch—Classic Solution

Define a loggable interface

Add type-specific methods to the Logger interface

```
public interface Logger {
    public void log (Object obj);
    public void log (String str);
    public void log (Date date);
    public void log (Exception ex);
}
```



89 | Session 2289

## Double Dispatch—Classic Solution

Define a loggable interface

Add type-specific methods to the Logger interface

Implementations of Loggable invoke the appropriate method

```
public class LoggableDate implements Loggable {
    private Date date;
    public void logThis (Logger logger) {
        logger.log (date);
    }
}
```

90 | Session 2289

## Double Dispatch—Classic Solution

Define a loggable interface

Add type-specific methods to the Logger interface

Implementations of Loggable invoke the appropriate method

Implementation of Logger.log

```
public void log (Object obj) {
    ((Loggable) obj).logThis (this);
}
```



91 | Session 2289

## Double Dispatch—Classic Problems

Must define a Loggable adapter or subclass for every type

Some types may be final—java.lang.String

Must add a method to the Logger interface for each new type, as well as an implementation method in LoggerImpl



92 | Session 2289

## Double Dispatch—Reflection

Select method based on argument class

```
private Method getLogMethod (Object obj) {
    try {
        Class cl = obj.getClass ();
        return getClass ().getDeclaredMethod
            ("log", new Class[] { cl });
    } // catch blocks omitted
}
```



93 | Session 2289

## Double Dispatch—Reflection

Logger implementation

```
public void log (Object obj) {
    try {
        getLogMethod (obj).invoke (this,
            new Object[] {obj});
    } // catch blocks omitted
}

private void log (String string) ...
private void log (Date date) ...
private void log (Exception exception) ...
```



94 | Session 2289

## Double Dispatch—Problem

```
private Method getLogMethod (Object obj) {
    Class cl = obj.getClass ();
    return getClass ().getDeclaredMethod
        ("log", new Class[] { cl });
}
```

Fails if log is called with an Exception subclass

Need more sophisticated utility method

Search all methods

isAssignableFrom

Search methods from superclasses



95 | Session 2289

## Double Dispatch—Results

Easy to extend for new types

No need to subclass or wrap the various loggable types

Loss of type safety

“Thinking in Patterns” —Chapter 12—Bruce Eckel

<http://www.mindview.net>

“Reflect on the Visitor design pattern”

—Jeremy Blosser

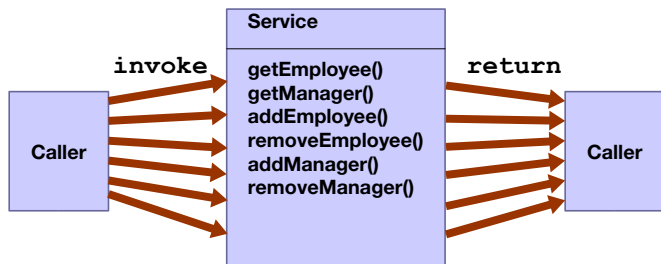
<http://www.javaworld.com/javaworld/javatips/jw-javatip98.html>



96 | Session 2289

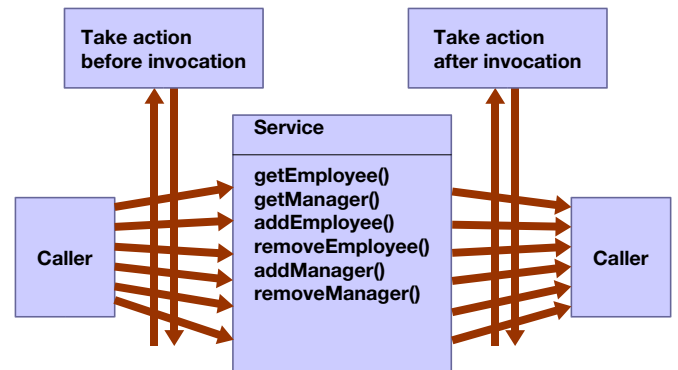
## Interposition

A means for adding to or amending behavior across a group of methods



97 | Session 2289

## Interposition



98 | Session 2289

## Interposition – Implementation

Use a dynamic proxy

Substitutes for the “real” object

Intercepts the method call

Perform action (before)

Invoke method on real object

Perform action (after)



99 | Session 2289

## Interposition – Example

Exception handling Interposition

```
public class ExceptionHandler implements
    InvocationHandler {
    private Object source;
    public ExceptionHandler (Object source) {
        this.source = source;
    }
    public void invoke (Object proxy,
        Method method, Object[] args)
        throws Throwable ...
```



100 | Session 2289

## Interposition – Implementation

```
public void invoke (Object proxy, Method method,
    Object[] args) throws Throwable {
    try {
        method.invoke (source, args);
    } catch (InvocationTargetException itex) {
        log (itex.getCause ());
        throw itex.getCause ();
    } catch (IllegalAccessException iaex) {
        throw new RuntimeException (iaex);
    }
}
```



101 | Session 2289

## Interposition – Sample Usage

RMI version of Service

```
public interface Service extends Remote ...
public class ServiceImpl implements Service ...

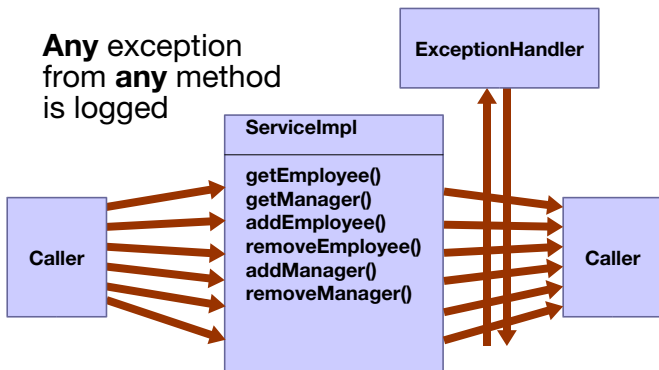
// create a dynamic proxy
Service proxy = (Service) Proxy.newInstance
    (impl.getClass ().getClassLoader (),
        new Class[] { Service.class },
        new ErrorHandler (impl));
Naming.rebind ("//habanero/Service", proxy);
```



102 | Session 2289

## Interposition

Any exception from any method is logged



103 | Session 2289

## Interposition – Another Example

Provides a read-only view of an object

```
public void invoke (Object proxy, Method method,
    Object[] args) throws Throwable {
    try {
        if (method.getName ().startsWith ("set")) {
            throw new SecurityException
                ("not allowed to set");
        }
        method.invoke (source, args);
    } // catch blocks omitted
}
```



104 | Session 2289

## Interposition – Conclusion

Applications

Logging

Persistence

Error handling



105 | Session 2289

## Interposition – Links

“Explore the Dynamic Proxy API”  
– Jeremy Blosser

<http://www.javaworld.com/javaworld/jw-11-2000/jw-1110-proxy.html>

“Using java.lang.reflect. Proxy to Interpose on Java™ Class Methods” – Tom Harpin

<http://developer.java.sun.com/developer/technicalArticles/JavalP/Interposing/>

Aspect Oriented Programming

<http://www.aspectj.org>



106 | Session 2289

## Real World Reflection



Session 2289

## Real World Reflection

Jakarta Tomcat Servlet engine

jEdit text editor

Java™ core libraries



108 | Session 2289

## Real World Reflection – Tomcat

Reference implementation of the Java™ Servlet API and JavaServer Pages™ technology

<http://jakarta.apache.org/tomcat>

Reflection used to configure server from XML files

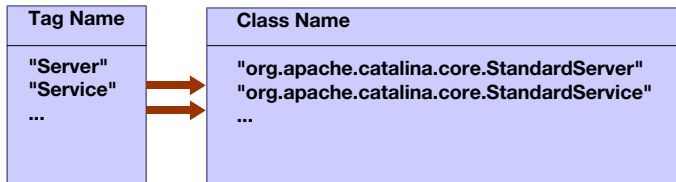


109 | Session 2289

## Tomcat – Example

An internal map associates tag name to class name

```
<Server port="8005" shutdown="SHUTDOWN">
  <Service name="Tomcat-Standalone">
    ...
```



111 | Session 2289

## Tomcat – Example

The attributes of the tag are parsed

Associated **set** method found via Reflection

Look first for set method with a String parameter

Look for set method with int/boolean parameter

Method is invoked on the newly created object

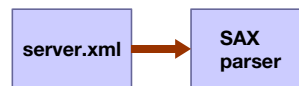
```
<Server port="8005" shutdown="SHUTDOWN">
  standardServer.setPort(8005);
  standardServer.setShutdown("Shutdown");
```

113 | Session 2289

## Tomcat – Example

The **server.xml** file contains basic server configuration information

```
<Server port="8005" shutdown="SHUTDOWN">
  <Service name="Tomcat-Standalone">
    ...
```



110 | Session 2289

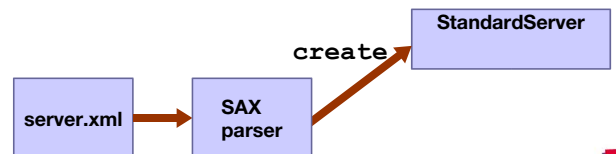
## Tomcat – Example

A tag name is encountered during parsing

The associated class name is retrieved

Class instance retrieved by Class.forName

Instance created by Class.newInstance

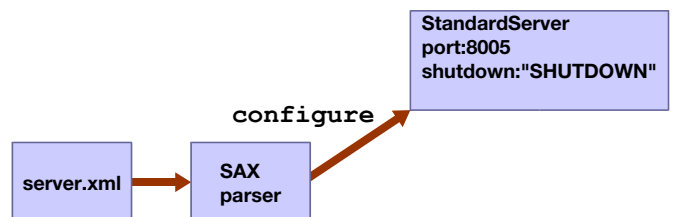


112 | Session 2289

## Tomcat – Example

Object configured from tag attributes

```
<Server port="8005" shutdown="SHUTDOWN">
```



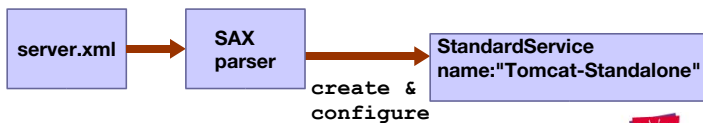
114 | Session 2289

## Tomcat – Example

Process repeated for child tags

```
<Server port="8005" shutdown="SHUTDOWN">  
  <Service name="Tomcat-Standalone">
```

```
StandardServer  
port:8005  
shutdown:"SHUTDOWN"
```

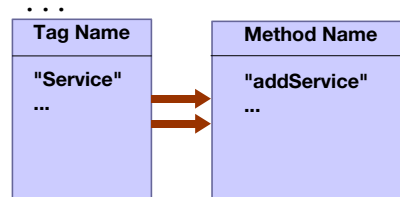


115 | Session 2289

## Tomcat – Example

An internal map also associates the tag name with a method name from the parent

```
<Server port="8005" shutdown="SHUTDOWN"  
debug="0">  
  <Service name="Tomcat-Standalone">
```



116 | Session 2289

## Tomcat – Example

The associated method is retrieved from the parent class (StandardServer)

The method is invoked on the parent object with the child object as the argument

```
<Server port="8005" shutdown="SHUTDOWN"  
debug="0">  
  <Service name="Tomcat-Standalone">
```

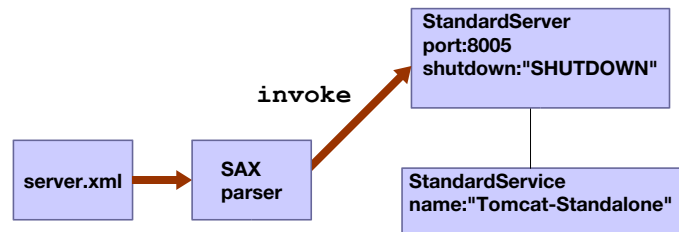
```
...  
standardServer.addService (standardService);
```



117 | Session 2289

## Tomcat – Example

The child object is added to the object graph



118 | Session 2289

## Tomcat – Conclusion

A complex data structure configured from an XML file

Performance not critical—used at startup

Flexibility in configuration is critical

Plug in different service types by tag name

Build structure via various methods



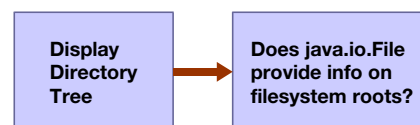
119 | Session 2289

## Real World Reflection – jEdit

Text editor based entirely on Java™ technology

<http://www.jedit.org>

Uses Reflection to discover library capabilities at runtime



120 | Session 2289

## Real World Reflection – jEdit

java.io.File.listRoots method was added in 1.2

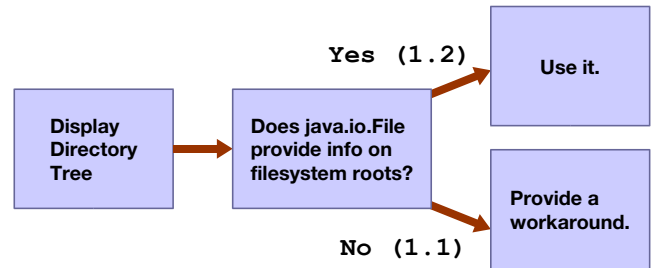
```
// try using Java 2 method first
try{
    method = File.class.getMethod
        ("listRoots",new Class[0]);
} catch(Exception e) {
    fsView = FileSystemView.getFileSystemView();
}
```



121 | Session 2289

## Real World Reflection – jEdit

Reflection used to select workaround for old versions



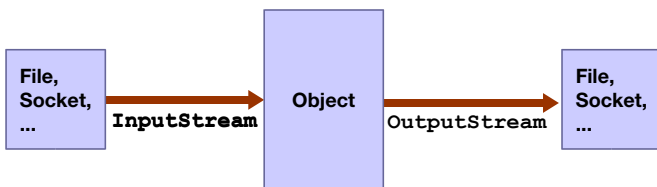
122 | Session 2289

## Real World Reflection – Java™ Core

Object Serialization

Reassemble Object from Stream

Disassemble Object into Stream

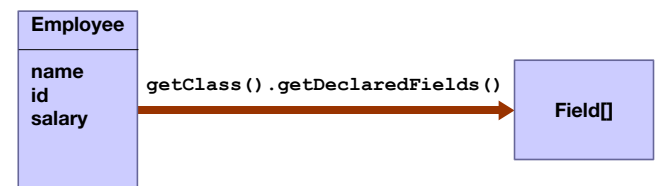


123 | Session 2289

## Object Serialization

Reflection used to determine fields

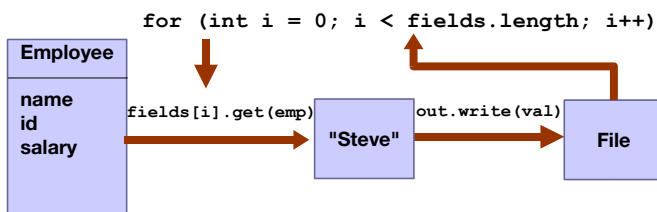
Even private ones!



124 | Session 2289

## Object Serialization

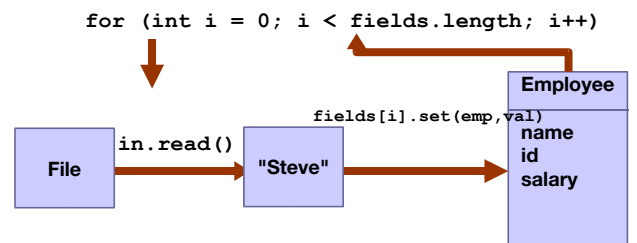
Reflection used to get the value of fields for writing to the stream



125 | Session 2289

## Object Serialization

Reflection used to set the value of fields from data read from the stream



126 | Session 2289

## Object Serialization – Conclusion

Some other applications

- Object-Relational Mapping

- Debugging

Code Generation vs. Reflection

<http://www.javaworld.com/javaworld/jw-11-2001/jw-1102-codegen.html>



127 | Session 2289

## When Not to Use Reflection



Session 2289

## Reflection Considered Harmful?

*Or... “When You Have a Crowbar, Everything Looks Like A Wall”*

Don't use reflection when...



129 | Session 2289

## Reflection Considered Harmful?

*Or... “When You Have a Crowbar, Everything Looks Like A Wall”*

Don't use reflection when...

- Direct invocation does the job

  - More simply

  - More clearly



130 | Session 2289

## Reflection Considered Harmful?

*Or... “When You Have a Crowbar, Everything Looks Like A Wall”*

Don't use reflection when...

- Direct invocation does the job

- Type safety is more important than flexibility

  - Errors in spelling class or method names caught at runtime, not compile-time



131 | Session 2289

## Reflection Considered Harmful?

*Or... “When You Have a Crowbar, Everything Looks Like A Wall”*

Don't use reflection when...

- Direct invocation does the job

- Type safety is more important than flexibility

- Performance is critical

  - Inside loops



132 | Session 2289

## Conclusion



Session 2289

## Summary

Apply the Reflection API to problems direct code can't solve

Know the tradeoffs of using Reflection

Employ the common patterns of Reflection use

Use utility methods to hide some of the complexity of the Reflection API



134 | Session 2289

## And...

Don't fear Reflection—fear indiscriminate use of Reflection.



135 | Session 2289

# Q&A



Session 2289



**BEYOND**  
BOUNDARIES

Session 2289