Meta programming

• Programming **about** programs
Lisp

• Metaprogramming originated with Lisp

• “Lisp is a programmable programming language.”
  –John Foderaro
# Code is data

<table>
<thead>
<tr>
<th>(lambda (x) x)</th>
<th>the identify function</th>
</tr>
</thead>
<tbody>
<tr>
<td>'(a b c)</td>
<td>a list of symbols a, b, and c</td>
</tr>
<tr>
<td>'(lambda (x) x)</td>
<td>a list of the symbol lambda, a list of x, and the symbol x</td>
</tr>
</tbody>
</table>
Data is code

<table>
<thead>
<tr>
<th>(+ 1 2 3)</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘(+ 1 2 3)</td>
<td>a list of integers 1, 2, and 3</td>
</tr>
<tr>
<td>(eval ‘(+ 1 2 3))</td>
<td>6</td>
</tr>
</tbody>
</table>
Metaprogramming

• Can build a data structure representing a program

• Can manipulate the data structure
  • => program transformations

• Can evaluate the data structure as a program
Eval in JavaScript

Can eval strings.

```
x = 2
alert(eval(‘x + 2’));    // 4
```
Java reflection

• Java supports metaprogramming through *reflection*

• Can ask an object for its class:

```java
java.lang.Class c = o.getClass();
```

• Or get it directly:

```java
Class c = String.class;
```

• Or get it by name:

```java
Class c = Class.forName("java.lang.String");
```
Creating a new instance

Class c = ...;
Object p = c.newInstance();
import java.lang.reflect.*; ...

Class c = ...;
Method[] ms = c.getDeclaredMethods();
System.out.println(c.getName()
    + " declares " + ms.length
    + " methods");
Querying a method

class C { int m(int x) { return x+1; } }

Class c = C.class;
Method m = c.getDeclaredMethods()[0];
m.getName()               // "m"
m.getDeclaringClass()     // C.class
m.getParameterTypes()     // { int.class }
m.getReturnType()        // int.class
Invoking a method

class C { int m(int x) { return x+1; } } 

Class c = C.class;
Method m = c.getDeclaredMethods()[0];

Object result = m.invoke(new C(), 3);
int v = (Integer) result;
assert v == 4;
class C { String s; }

C x = new C();
Class c = x.getClass();
Field[] fs = c.getDeclaredFields();

Object o = fs[0].getValue(x);
String str = (String) o;

fs[0].setValue(x, “I’ve changed.”);
Supertypes

Class c = ...;
Class sup = c.getSuperclass();
Class[] interfaces = c.getInterfaces();
What’s reflection used for?

- Enterprise Java Beans (EJB)
  - get an object from over the network
  - can query the object to see what methods it supports and then invoke the methods
What’s reflection used for?

- Object-relational mappings
  - Map objects onto database rows
  - Map field names to database columns
- => Can access a database as if a Java program
- Reflection code in the library handles all the translation to and from SQL
What’s reflection used for?

- Dynamic proxies

- Can create a **proxy object** that changes how method dispatch works

- Method invocations on the proxy object get routed to an **invoke** method
Ruby

- OO programming language derived from Smalltalk
- Developed by Yukihiro “Matz” Matsumoto 1990s
- Dynamically typed
- Ruby on Rails – a very popular web framework
  - used to run Twitter (which now uses Scala)
Running ruby

• **irb** - interactive Ruby shell

  ```ruby
  irb> puts "Hello World!"
  ```

• **ruby** - batch interpreter
OO in Ruby

- Class-based language
- Everything is an object
  \[-199.\text{abs} \rightarrow 199\]
• Can eval a named method:

```ruby
meth = 'size'
[1,2,3,4].send(meth)
# returns 4
```

• equivalent to:

```ruby
[1,2,3,4].size
```
Metaprogramming in Ruby

- Can add methods to classes:

```ruby
Fixnum.module_eval do
  def to_word
    if (0..3).include? self
      ['none', 'one', 'a couple', 'a few'][self]
    elsif self > 3
      'many'
    elsif self < 0
      'negative'
    end
  end
end
```

Add `to_word` method to class `Fixnum`

```ruby
1.to_word  # returns ‘one’
7.to_word  # returns ‘many’
```
Metaprogramming in Ruby

• This feature is called **open classes**.

```ruby
Fixnum.module_eval do
  def to_word
    ...
  end
end
```

• `module_eval` reopens an existing class and appends a new method.
Metaprogramming in Ruby

• Could also just write:

```ruby
class Fixnum do
  def to_word
    ...
  end
end
```
Metaprogramming in Ruby

• But, can also evaluate code in a `module_eval`

```ruby
Fixnum.module_eval do
  def self.create_multiplier(name, num)
    module_eval "def #{name}; self * #{num}; end"
  end
end

Fixnum.create_multiplier('multiply_by_pi', Math::PI)
4.multiply_by_pi  # returns 12.5663706143592
```

• Extended the `Fixnum` class object with a method `create_multiplier` that adds a method to every instance of `Fixnum`
Method missing hook

- `method_missing` is invoked when a method that does not exist

```ruby
class Fixnum
  def method_missing(meth)
    method_name = meth.id2name
    if method_name =~ /^multiply_by_(\d+)$/
      self * $1.to_i
    else
      raise NoMethodError, "undefined method"
    end
  end
end

16.multiply_by_64  # returns 1024
16.foo # NoMethodError
```
Metaprogramming in C#

- C# has **expression trees**:

    static ParameterExpression parm =
        Expression.Parameter(typeof(double), "radius");

    static Expression<Func<double, double>> circleAreaExpr =
        Expression.Lambda<Func<double, double>>(
            Expression.Multiply(
                Expression.Constant(Math.PI),
                Expression.Multiply(parm, parm)),
            parm);

    Func<double, double> area = circleAreaExpr.Compile();

    area(5); // 78.5398163397448
Summary

• Metaprogramming powerful means of manipulating programs from within the language

• Java reflection is a limited metaprogramming facility

• eval in Lisp, JavaScript and other languages let programmer build a program in memory and then run it

• Ruby metaprogramming lets programmer modify behavior of classes, add methods, ...