

Tipping the scale:

The Ruby features that make the difference

- murphee (Werner Schuster)
- Blog @ <http://jroller.com/page/murphee>



Ruby: first encounter

- OOP
- `5.times { |x| p "Hello" }`
- Dynamic typing



murphee's reaction...




So?



Misleading items

- “Java has OOP too”
 - “\$foo, @bar, etc? Looks like Perl!”
 - “I like my static typing, thank you very much...”
 - “Ducks typing? How do they reach the keyboard?”
-
-

What I missed the first time round

- Blocks
 - Metaprogramming
 - Dynamic typing
 - Mixins
- 

Ruby in 10 seconds

- Matz (Yukihiro Matsumoto)
 - OOP
 - Smalltalk and Lisp
 - Dynamic/Interactive/**Red**/Succinct/Flexible
-
-

The many shapes of Ruby

```
#guess what this does
def three_lingo_hello(name)
  ["Hello", "Seas", "HowdyHo!"].collect{|g|
    "#{g} to Ruby, #{name}! \n"
  }
end

three_lingo_hello("murphee")
```

The many shapes of Ruby

```
# we don't like Make, we got Rake!  
task :default => [:test]  
  
task :test do  
  ruby "test/unittest.rb"  
end
```



The many shapes of Ruby

```
# we don't like Make, we got Rake!  
task :default => [:test]  
  
task :test do  
  ruby "test/unittest.rb"  
end
```

Yep, that's Ruby too

The many shapes of Ruby

```
require 'builder'
x=Builder::XmlMarkup.new(:target => $stdout,
                        :indent => 1)

x.date {
  x.year "2006"
  x.month "01"
  x.day "01"
}
```



Ruby in 10 seconds

```
require 'builder'  
x=Builder::XmlMarkup.new(:target => $stdout,  
                          :indent => 1)  
  
x.date {  
  x.year "2006"  
  x.month "01"  
  x.day "01"  
}
```

```
<date>  
  <year>2006</year>  
  <month>01</month>  
  <day>01</day>  
</date>
```

Blocks: Translation

- In Lisp:
 - Lambda Expressions
 - In Smalltalk:
 - Blocks
 - Generally:
 - Closure
 - Java
 - ~ Anonymous Classes (kind of...)
-
-

Blocks: Intro

- anonymous chunk of code
 - looks like
 - ```
{|param|
 param +1
}
```
  - names between symbols are input arguments
  - code evaluated later
- 
-

# Blocks: Use Case 1: Iterators

```
words = ["foo", "bar", "fubar"]
words.each{|item|
 p item
}
```



# Blocks: Use Case 1: Iterators

```
words = ["foo", "bar", "fubar"]
words.each{|item|
 p item
}
```

## Compare Java version:

```
List words = new ArrayList();
words.addAll(Arrays.asList(
 new String[]{"foo", "bar", "fubar"}));
for (Iterator iter = words.iterator(); iter.hasNext();) {
 String el = (String) iter.next();
 System.out.println(el);
}
```



# Blocks: Use Case 1: Iterators

```
words = ["foo", "bar", "fubar"]
words.each{|item|
 p item
}
```

## Compare Java 1.5 version:

```
List words = new ArrayList();
words.addAll(Arrays.asList(
 new String[]{"foo", "bar", "fubar"}));
for (String el : words) {
 System.out.println(el);
}
```

**But...**



# Blocks: Use Case 2: Map

```
words = ["foo", "bar", "fubar"]
uc = words.collect{|item|
 item.upper
}
```

## Compare Java 1.5 version:

```
List words = new ArrayList();
words.addAll(Arrays.asList(
 new String[]{"foo", "bar", "fubar"}));
List uc = new ArrayList();
for (String el : words) {
 uc.add(el.toUpperCase());
}
```

# Blocks: Use Case 3: Filter/Select

```
words = ["foo", "bar", "fubar"]
l = words.select{|item|
 item.size > 3
}
```

## Compare Java 1.5 version:

```
List words = new ArrayList();
words.addAll(Arrays.asList(
 new String[]{"foo", "bar", "fubar"}));
List l = new ArrayList();
for (String el : words) {
 if(el.length() > 3){
 uc.add(el.toUpperCase());
 }
}
```

# Blocks: Use Case 4: Iterator with index

```
words = ["foo", "bar", "fubar"]
l = words.each_with_index{|i,e|
 p "#{i}. #{e}"
}
```

## Compare Java 1.5 version:

```
List words = new ArrayList();
words.addAll(Arrays.asList(
 new String[]{"foo", "bar", "fubar"}));
int counter = 0;
for (String el : words) {
 System.out.println(counter + ". " + el);
 counter++;
}
```

**etc.**



# Blocks: Use Case 5: Transactions

```
IO.open(IO.sysopen("foo.txt"), "r") { |aFile|
 # do stuff with the file
}
```



# Blocks: Use Case 5: Transactions

- Transaction happens in block
- No need to close stream/file
- Keeps code clear





# Blocks: Implementation

```
def wrap_around(name, &block)
 p "Pre"
 block.call()
 p "Post"
end
```

```
wrap_around("murphee") {
 p "Hello there, block!"
}
```

---

---

# Blocks: Implementation

```
def wrap_around(name, &block)
 p "Pre"
 block.call()
 p "Post"
end
```

```
wrap_around("murphee") {
 p "Hello there, block!"
}
```

```
Pre
Hello there, block!
Post
```

# Metaprogramming: Intro

- Programs that program
- Runtime Code generation
- Specialization



# Metaprogramming: Use Case 1: Proxies

```
class FooProxy
 # eg. remote is an XmlRpc connection

 def method_missing(*syms)
 remote.call(syms[0].to_s)
 end
end

x = FooProxy.new
x.helloWorld
x.fluffy()
x.tryThis!()
```

---

---

# Metaprogramming: Use Case 2: Accessors

```
class Foo
 attr_accessors :hello, :world
end
```

```
f = Foo.new
f.hello = 1
p f.world
...
```



# Metaprogramming: More...

- ActiveRecord
  - creates accessors based on DB schema
- Aspect Oriented Programming?
  - no special tools needed
- [Your ideas here]



# Mixins: Intro

- No multiple inheritance
- Mixins allow to add functionality to a class
- Used to add each/collect/select/...
  -



# Mixin: Use Case 1: Make class iterable

```
class ToTen
 include Enumerable
 def each(&bl)
 (1..10).each{|n|
 yield n
 }
 end
end

x = ToTen.new
x.each {|n| ...}
x.collect {|n| ...}
x.select {|n| ...}
.....
```

---

---



# Misc:

- `ObjectSpace.each_object`
  - Domain Specific Languages
    - Rake, etc.
  - Continuations
  - `irb`
- 
-

**etc.**



# JRuby

- Don't go cold turkey
    - Ruby implementation on the JVM
  - Runs subset of Rails
    - full version end of summer(?)
  - Script Java objects
    - prototyping
    - powerful configuration files
    - end user scripting
    - flexibility
- 
-

# JRuby: EclipseShell

- EclipseShell
  - interactive editor with shell
- modelled after Smalltalk Workspaces
- Future:
  - develop Eclipse RCP apps *in* Eclipse
  - prototype Eclipse plugins
  - explore Eclipse plugins



# People who liked Ruby, also looked at:

- Smalltalk
    - (home of the first refactoring tools)
  - Squeak
    - free Smalltalk written in Smalltalk
  - Croquet
    - immersive 3D collaboration environment,
    - based on Squeak
    - cf. Neal Stephenson's “Metaverse”
- 
-

# Let's wrap up

- Books @ Pragmatic Programmers
    - <http://www.pragmaticprogrammer.com/bookshelf/index.html>
  - why's (poignant) guide to Ruby
    - <http://poignantguide.net/ruby/>
    - Now with 50% more foxes.
  - EclipseShell
    - <http://eclipse-shell.sourceforge.net/>
- 
-