Groovy - What, Why and How?

What is it?

- dynamic language running on the JVM
- compiles to Java byte-code
- interoperable with Java
- JSR 241 from 2004
- largely adopted along with Grails

Why use it?

- low learning curve if you know Java
- DSL fluency and Builders
- shell+scripting = command line power
- Tooling:
  - groovysh
  - groovyConsole
  - SwingPad
  - Maven
  - IntelliJ(with Eclipse not far behind)

Builders

Builders are a way to express tree data structures in a natural and fluent way.

MarkupBuilder and SwingBuilder are excellent examples.

```
def writer = new StringWriter()
def builder = new MarkupBuilder(writer)
builder.string(){
    value('hello world')
}
writer.toString()
```

...

```
<string>
    <value>hello world</value>
</string>
```

Builders

SwingBuilder provides easy ways to define UI

and

SwingPad is a nice tool for quick prototyping.

```
import java.awt.BorderLayout
panel {
  borderLayout()
  textField("Change Me!", id:'tf', constraints: BorderLayout.NORTH )
  button("change me")
}
```

Groovy Demo Time

Syntactic Sugar

Groovy adds a lot of nice stuff to the existing JDK.

- Collection: each(), eachWithIndex(), collect(), flatten(), combinations(), intersection(), disjoint(), sort()
- List: first(), last(), pop(), push(), asImmutable(), asSynchronized()
- Map: groupBy(), subMap(), toMapString()
- File: eachDirMatch(), eachDirRecurse(), eachFileMatch(), withReader(), withWriter()
- String: eachLine(), eachMatch(), findAll(), count(), is"XYZ"(), toURI(), toURL(), execute()
- And so on.....

Groovy Demo Time

MOP

MetaObjectProgramming allows you to affect the runtime behaviour of classes using various techniques.

- map coercion: useful for mocks, alternative to inner classes
- AST transforms
  - @Delegate
  - @Bindable
  - @Immutable
  - @Singleton
- ExpandoMetaClass: class level or per instance dynamic methods
- @Category and @Mixin

Groovy Demo Time