Practically Groovy:

Getting Groovy with enterprise development

#practicallygroovy

gr8conf – July 28, 2014

Rocky DeVries

Target.com Search Product

Twitter: @rockympls

Github: http://github.com/rockympls

Why use Groovy?

It looks and works (mostly) like Java!

This works in Groovy:

```
public class Foobar {
  public static void main(String[] args) {
    System.out.println("This IS groovy!");
  }
}
```

It adds some functional flavor to the strictly OO nature of Java

This also works:

```
def closure = { println "This IS groovy!" }
closure.call()
```

It's completely interoperable with Java and its vast array of 3rd party libraries

This Spring MVC method works too:

```
@RequestMapping('/myPage/{name}')
 void renderPage(@PathVariable String name, HttpServletRequest request,
        HttpServletResponse response) {
   renderWithName(name, request, response)
 }
```

A practical example:

With its simplified syntax and standard libraries, Groovy makes everything you'd do in Java easier

Let's sort a map by value in descending order:

First, make a comparator. One that has a reference to the existing map.

```
public class HighestToLowestComparator implements Comparator<String> {
  Map<String, Integer> disordered;

  public HighestToLowestComparator(Map<String, Integer> map) {
    this.disordered = map;
  }

  @Override
  public int compare(String string1, String string2) {
    int compare = disordered.get(string1).compareTo(disordered.get(string2));
    return compare * -1;
  }
}
```

Java woes, cont'd

Then, put the old map into a new one:

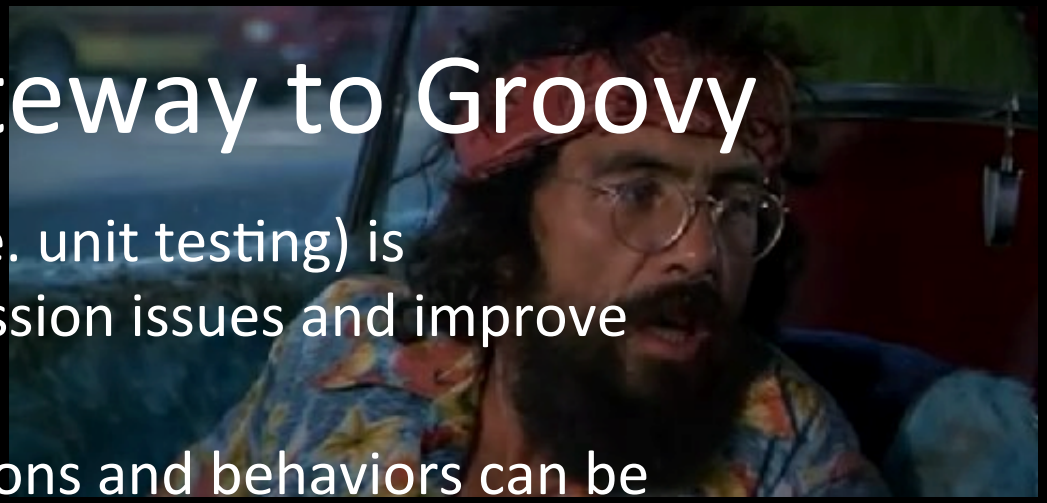
```
Map<String, Integer> sortedMap =
 new TreeMap<>(new HighestToLowestComparator());
sortedMap.putAll(aNormalHashMap);
for (String key : sortedMap.keySet()) {
  System.out.println(key + ": " + frequencies.get(key));
}
```

OK then, how about in Groovy?

```
Map<String, Integer> stuff = [foo: 3, bar: 1, baz: 9]
stuff.sort { it.value * -1 }.each { println "${it.key}: ${it.value}" }
```

That's it. 👌

Testing: the gateway to Groovy



Fact: automated testing (i.e. unit testing) is necessary to prevent regression issues and improve quality

Fact: Mocking out interactions and behaviors can be cumbersome and time-consuming

Sad Fact 😭**:** Interactions and expectations often go untested

So what's the solution?

The spock framework!
A Groovy-based testing framework that addresses these gripes beautifully.

First: a “simple” unit test in Java

Libraries: Spring, JUnit , Mockito

Purpose: Verify that an endpoint is called

The code under test:

```
public class EndpointCallerJ {

  @Value("${app.endpoint.url}")
  private String endpointUrl;
  @Autowired
  private RestTemplate requester;
  @Value("${app.api.key}")
  private String apiKey;

  public String readData() {
    Map<String, String> options = new HashMap<String, String>();
    options.put("apiKey", apiKey);
    options.put("service", "data");
    return requester.getForObject(endpointUrl, String.class, options);
  }

}
```

… And the test!


```
@RunWith(MockitoJUnitRunner.class)
public class EndpointCallerJTest {

  @Mock
  RestTemplate mockGatherer;
  @InjectMocks
  EndpointCallerJ gatherer = new EndpointCallerJ();

  @Test
  public void callsEndpointWithCorrectParameters() {
    String url = "http://test.target.com/blah";
    ReflectionTestUtils.setField(gatherer, "apiKey", "test key");
    ReflectionTestUtils.setField(gatherer, "endpointUrl", url);

    when(mockGatherer.getForObject(eq(url), eq(String.class), (Map) argThat(
            allOf(
                    hasEntry("apiKey", "test key"),
                    hasEntry("service", "data")
            )
    ))).thenReturn("{\"results\": true}");
```

```
 Assert.assertEquals("{\"results\": true}", gatherer.readData());

    verify(mockGatherer, times(1)).getForObject(eq(url), eq(String.class),
            (Map) argThat(
                    allOf(
                            hasEntry("apiKey", "test key"),
                            hasEntry("service", "data")
                    )
            )
    );
  }
}
```

Items of note:

- Lots of parentheses
- Lots of method calls
- **LOTS OF CODE**
- Difficult to understand, *especially* after not looking at it for a while.

OK. How does it look in Groovy?

```
class EndpointCallerSpec extends Specification {

  EndpointCaller gatherer = new EndpointCaller()

  def ”Calls endpoint and parses response"() {
    setup:
    gatherer.endpointUrl = 'http://services.target.com/ds'
    gatherer.requester = Mock(RestTemplate)
    gatherer.apiKey = "myKey"

    when:
    String output = gatherer.readData()

    then:
    1 * gatherer.requester.getForObject(gatherer.endpointUrl,
         String, [apiKey: "myKey", service: "data"]) >> '{"response": true}'
    new JsonSlurper().parseText(output).response == true
  }

}
```

That’s it. 😯

The power of Groovy

Type inference/loose Typing:

```
def foo = "bar"
def oldPeople = listOfPeople.findAll { person ->
 person.age > 30 }
RestTemplate template = Mock()
```

List/Map initialization shorthand:

```
Map<String, Integer> namesAndAges = [Johan: 38, Jenn: 27, Suresh: 19]
List<String> flavors = ["Vanilla", "Cherry", "Root Beer"]
```

Closure power!

```
List<Account> pastDue = accounts.findAll { !it.paid && it.dueDate < new
Date() }
Owner richGuy = nbaOwners.max { it.netWorth }
```

Metaclass/expando metaclass:

```
String.metaClass.l33t =
{ delegate.toLowerCase().replaceAll('[aA]','4').replaceAll('[eE]',
        '3').replaceAll('[iI]','1').replaceAll('[oO]','0') }
```

JSON and XML

Groovy takes the heavy lifting out of parsing JSON and XML

XMLSlurper**: an expressive way to traverse XML structures**

Imagine you have the following XML:

```
<funInfo>
    <terribleTeam name="Detroit Lions”>0-16</terribleTeam>
    <terribleTeam name="Minnesota Twins">Worst ever!</terribleTeam>
    <terribleTeam name="Minnesota Vikings">4 Super Bowls.  0 Wins</terribleTeam>
    <terribleTeam name="Green Bay Packers">Cheeseheads. ‘Nuff said.</terribleTeam>
</funInfo>
```

Here’s how you read it:

```
def tree = new XmlSlurper().parseText(XML)
   tree.terribleTeam.each { team ->
     println "${team.@name} -> ${team.text()}"
   }
```

JSON and XML

What about JSON?

It's even easier.
Let's try a similar data structure in JSON:

```
{
  "worstTeams":
  [
  {"name": "Detroit Lions", "reason": "0-16" },
  {"name": "Minnesota Twins", "reason": "Worst ever!" },
  {"name": "Minnesota Vikings", "reason": "4 Super Bowls.  0 Wins" },
  {"name": "Green Bay Packers", "reason": "Cheeseheads. 'Nuff said." }
  ]
}
```

And reading it?

```
new JsonSlurper().parseText(JSON).worstTeams.each { team ->
     println "${team.name} -> ${team.reason}"
}
```

Gettin' Groovy

Let's look at an example implementation of a ReST service layer

100% Groovy

Questions?

Contact me **@rockympls**
Check out all of the code at
http://github.com/rockympls

Start getting groovy!

References:

"They are who we thought they were" (2006) – Dennis Green et al
"Boilerplate" (2000) – Guinan
"The Wizard" (1989) – Holland, Savage, Edward, Lewis et al
"Friday the 13th" (1980) – Cunningham et al
"Up in Smoke" (1978) – Adler, Chong, Marin et al
"Low Rider" (1975) – War (Allen, Brown, Dickerson, Jordan, Miller, Oskar, Scott)
"Woodstock" (1969) – Lang, Roberts, Rosenman, Kornfeld
"Star Trek" (1966) – Rodenberry, Shatner, Nimoy et al