Groovy Basics

Sang Shin
JPassion.com
“Learn with Passion!”

Topics

- What is and Why Groovy?
- Groovy Syntax
- Differences from Java
- Refactoring Java code into Groovy code
- Inter-operating with Java
- Groovy Ecosystem

What is & Why Groovy?

What is Groovy?

- Dynamic, objected oriented, scripting language for JVM
- Seamless integration with Java
  - Designed with Java in mind from the beginning (unlike other scripting languages)
  - Easy to learn for Java programmers
- Borrowed language features from Ruby, Python, Smalltalk

Why Groovy over Other Scripting Languages?

- Groovy is a dynamic language “specifically” designed for Java platform
  - Leverage the benefit of JVM
- Groovy provides an effortless transition from Java
  - Groovy code, when compiled, generated Java bytecode
  - Existing Java code works in Groovy environment “as it is” basis
  - Incremental change is possible, in fact, recommended (when you plan to migrate existing Java code to Groovy)

Groovy IS Java (or BETTER Java)

- Provides syntactic sugar
  - > Easy and fun to code (like Ruby)
- Provides new language features over Java
  - > Closure (Java 8 now supports closure through Lambda)
  - > Meta-programming
- Provides easier development environment
  - > Scripting
  - > Combines compilation and execution into a single step
  - > Shell interpreter

Lab

Exercise 0: Install Groovy
Exercise 1: Groovy is Java
5610_groovy_basics.zip

Why Groovy (or Scala)?
What is “State of Java” (Language & JVM)?

Java as a Programming Language

- Java programming language has been a huge success but it is showing its age
  - Java programming language has not evolved significantly since Java SE 5 (2004) until Java SE 8
- Java programming language syntax is verbose, complex
  - Compared to other modern languages
- Java programming language, until Java SE 8, lacks modern language features
  - Closure, Meta-programming, DSL, Functional-programming, Operator overloading, Regular expression as a first class citizen, etc
  - Java 8 now provides some of these features (Closure, Functional-programming through “Lambda”)

JVM as a Run-time platform

- JVM is proven to be a great run-time platform, however
  - Secure, highly performing, mature, etc
- There are large number “ready to use” Java libraries over JVM
  - Commercial and open-sourced
- So we need a better programming language leveraging the current JVM
  - More productive, more fun, less verbose syntax
  - With modern language features
  - Seamless interoperability with Java programs
- Viable Choices
  - Groovy, Scala, JRuby, Clojure

Groovy Tools

- Groovy Shell
  - Interactive command-line application which allows easy access to evaluate Groovy expressions, define classes and run simple experiments
  - groovysh.bat (Windows), groovysh (Mac OS/Linux)
- Groovy Console
  - GUI version of Groovy Shell
  - Lets create, save, load, and runs Groovy code
  - groovyConsole.bat (Windows), groovyConsole (Mac OS/Linux)

Groovy Syntax

Define Variables with “def”

- “def" is a replacement for a type in variable definitions
  - “def” is used to indicate that you don't care about the type
  - You can also think of "def" as an alias of "Object"

```
def dynamic  =  1
println dynamic           // 1
println dynamic.class   // java.lang.Integer

dynamic = "I am a String stored in a variable of dynamic type"
println dynamic           // I am a String stored in a variable of dynamic type
println dynamic.class   // java.lang.String

int typed = 2
println typed
//typed = "I am a String stored in a variable of type int??"  // throws ClassCastException
```

Define Methods in a Class

```
class Calculator {
    // Use “def” to replace return type
    def add (x, y) {
       x+y       // No return statement required in Groovy
    }
    def subtract (x, y) {
       x-y
    }
}

result1 = new Calculator().add(13,4)
result2 = new Calculator().subtract(13,4)
result3 = new Calculator().add("sang", "shin")
result4 = new Calculator().subtract("sangshin", "sang")

println result1    // 17
println result2    // 9
println result3    // sangshin
println result4    // shin
```

Define Methods in a Script

```
def add (x, y) {
   x+y
}
def subtract (x, y) {
   x-y
}

result1 = add(13,4)
result2 = subtract(13,4)
result3 = add("sang", "shin")
result4 = subtract("sangshin", "sang")

println result1    // 17
println result2    // 9
println result3    // sangshin
println result4    // shin
```

List

```
// Each list expression creates an implementation of java.util.List
def list = [5, 6, 7, 8]
println list.get(2) // 7
println list[2] // 7
println list  instanceof  java.util.List // true

// Create an empty list
def emptyList = []
println emptyList.size() // 0
emptyList.add(5)
println emptyList.size() // 1
emptyList<<6
println emptyList.size() // 2
```

Range

- Range can be used as Lists since Range extends java.util.List.

```
// an inclusive range
def range = 5..8
assert range.size() == 4
assert range.get(2) == 7
assert range[2] == 7
assert range instanceof java.util.List
assert range.contains(5)
assert range.contains(8)

// lets use a half-open range
range = 5..<8
assert range.size() == 3
assert range.get(2) == 7
assert range[2] == 7
assert range instanceof java.util.List
assert range.contains(5)
assert ! range.contains(8)
```

```
// get the end points of the range without using indexes
range = 1..10
assert range.from == 1
assert range.to == 10
```

Map

- Map keys are strings by default: [a:1] is equivalent to ["a":1]

```
def map = [name:"Gromit", likes:"cheese", id:1234]
assert map.get("name") == "Gromit"
assert map.get("id") == 1234
assert map["name"] == "Gromit"
assert map['id'] == 1234
assert map instanceof java.util.Map

def emptyMap = [:]
assert emptyMap.size() == 0
emptyMap.put("foo", 5)
assert emptyMap.size() == 1
assert emptyMap.get("foo") == 5
```

String can be defined in 3 ways

```
// Double quotes  – String interpolation is supported (GString)
def name = "Sang Shin"
def name1 = "Hello, ${name}"  // => Hello, Sang Shin
println name1 + ", " + name1.class.name

// Single quotes – String interpolation is not supported (No GString)
def name2 =  'Hello, ${name}'  // => Hello, ${name}
println name2 + ", " + name2.class.name

// Slashes – String interpolation is supported (GString)
def name3 = /Hello, ${name}/  // => Hello, Sang Shin
println name3 + ", " + name3.class.name
```

Using Slashes for Defining a String

- Using slashes for defining a string has a benefit of not requiring an extra backslash for escaping special characters
- Handy with regular expressions or Windows file/directory path names.

```
// Compile error if you do not use backslash for escaping backslash
//def windowPathWithQuotes1 = 'C:\Windows\System32'
def windowPathWithQuotes2 = 'C:\\Windows\\System32'   // single quote '
println windowPathWithQuotes2
def windowPathWithQuotes3 = "C:\\Windows\\System32"  // double quote “
println windowPathWithQuotes3

// No need to use an extra backslash
def windowPathWithSlashes = /C:\Windows\System32/
println windowPathWithSlashes
```

GString (String Interpolation)

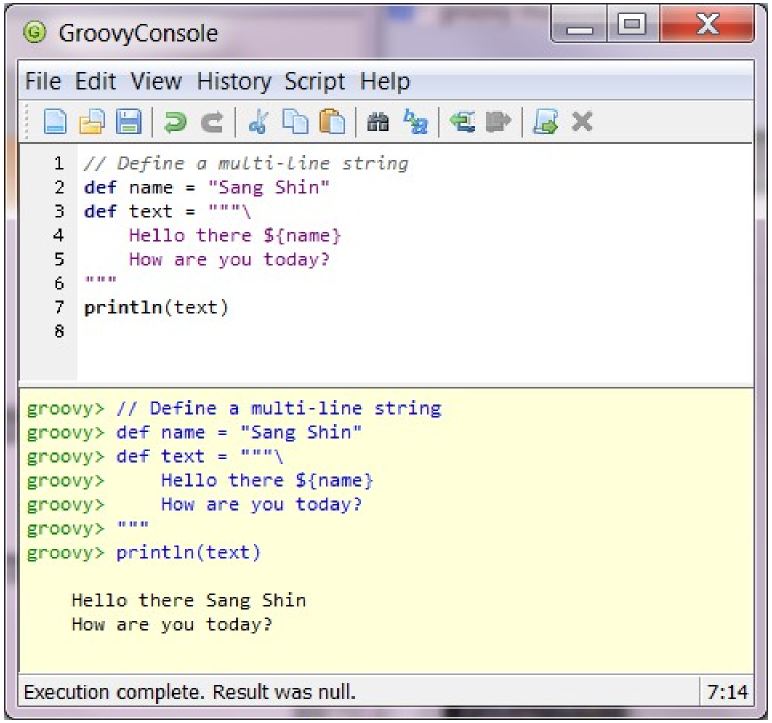
- Groovy creates a GString object when it sees a String defined with double-quote or slash with embedded ${expression}
- The expression gets evaluated in lazy fashion (meaning the evaluation happens only when the string is accessed)

```
foxtype = 'quick'
foxcolor = ['b', 'r', 'o', 'w', 'n']
result = "The $foxtype ${foxcolor.join()} fox"
println result // => The quick brown fox
println result.class.name // => org.codehaus.groovy.runtime.GStringImpl
```

Multi-line Strings

- A multi-line string is defined by three double quotes or three single quotes
- Multi-line string can be used to define an embedded template – XML, HTML, Email, SQL, etc)

```
// This is a compile error
// def foo = "hello

// Define a multi-line string
def name = "Sang Shin"
def text = """\
    Hello there ${name}
    How are you today?
"""
println(text)
```



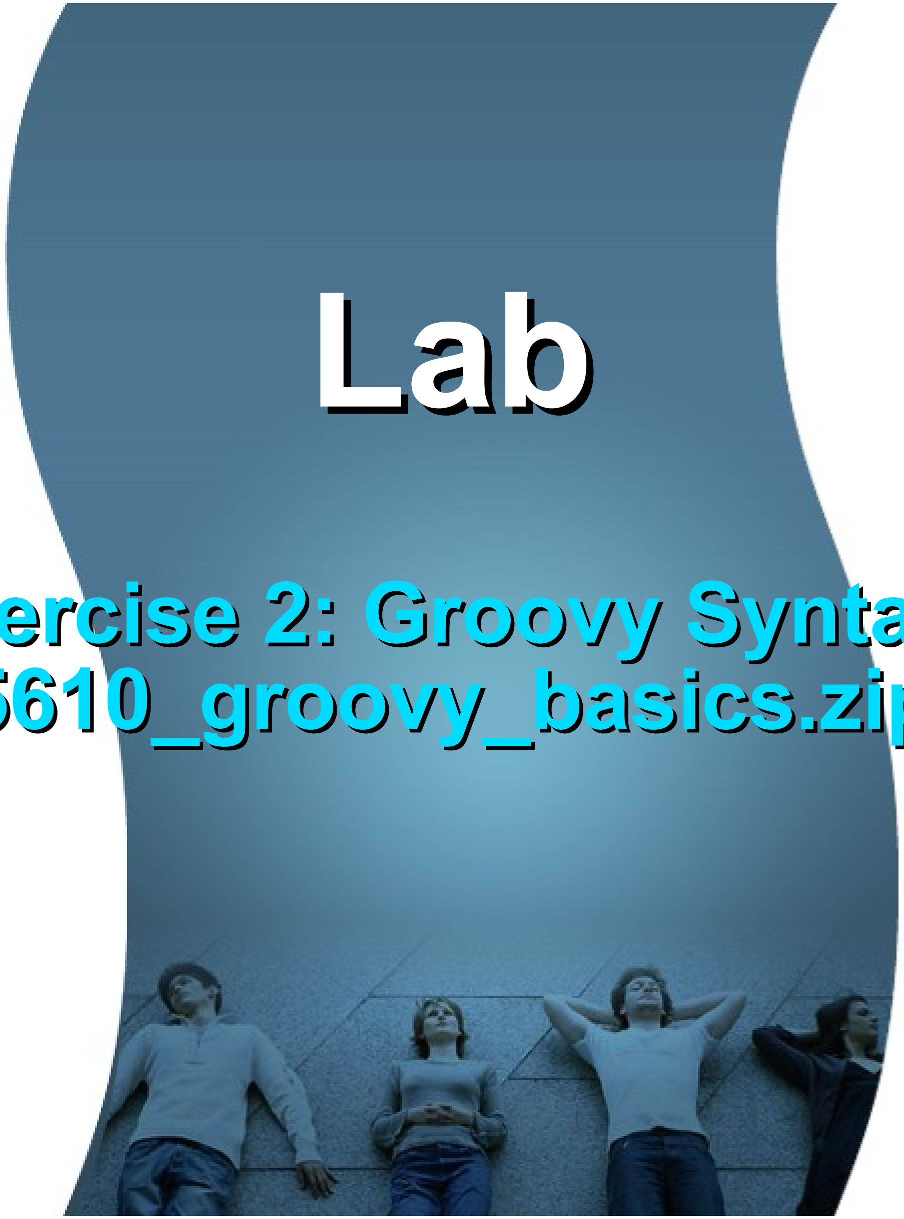
```
// Define a multi-line string
def name = "Sang Shin"
def text = """\
    Hello there ${name}
    How are you today?
"""
println(text)
```

```
groovy> // Define a multi-line string
groovy> def name = "Sang Shin"
groovy> def text = """\
groovy>     Hello there ${name}
groovy>     How are you today?
groovy> """
groovy> println(text)

    Hello there Sang Shin
    How are you today?
```

Execution complete. Result was null. 7:14

Lab

Exercise 2: Groovy Syntax I
5610_groovy_basics.zip



Regular Expression Support

- Groovy supported Regular Expression operators
  - Match operator (==~)
  - Create Matcher operator (=~)
  - Create Pattern operator (~pattern)

RegExp: Match Operator (==~)

- Match operator (==~) returns true if the regular expression matches the string

```
println "something" ==~ /something/   // => true
println "something" ==~ /.*g$/   // => true (ending char is g)
println "something" ==~ '.*g$'   // => true  (ending char is g)
println "something" ==~ '.*s$'   // => false (ending char is not s)
println "something" ==~ '^s.*$'   // => true (starting char is s)

println "something" ==~ /\D*/   //=> true (non digital characters)
println "something" ==~ '\\D*'   // => true (non digital characters)
//println "something" ==~ '\D*'   // compile error
```

RegExp: Create a Matcher Operator (=~)

- Create Matcher operator (=~) returns Matcher object if the matcher has any match results
- You can then use various methods of the Matcher object

```
// Create a Matcher
def myMatcher = "cheesecheesecheese" =~ /chee/
println myMatcher instanceof java.util.regex.Matcher // => true

// Call some methods of Matcher object
println myMatcher.size()  // => 3
println myMatcher[0]  // => chee

// Do some replacement
println myMatcher.replaceFirst("nice")  // => nicesecheesecheese
println myMatcher.replaceAll("good")  // => goodsegoodsegoodse
```

RegExp: Create a Pattern Operator (~String)

- Create Pattern operator (~String) returns Pattern object from the String

```
// ~String creates a Pattern from String
def pattern = ~/foo/
// Perform a matching through the Pattern object
println pattern instanceof java.util.regex.Pattern // => true
println pattern.matcher("foo").matches()    // => true
println pattern.matcher("foobar").matches() // => false

// ~String creates a Pattern from String
def pattern2 = ~/f.*/
// Perform a matching through the Pattern object
println pattern2 instanceof java.util.regex.Pattern // => true
println pattern2.matcher("foo").matches()    // => true
println pattern2.matcher("foobar").matches() // => true
```

Operator Overloading

- Groovy supports operator overloading which makes working with Numbers, Collections, Maps and various other data structures easier to use
- Various operators in Groovy are mapped onto regular Java method calls on objects
- This allows you the developer to provide your own Java or Groovy objects which can take advantage of operator overloading

Operator Overloading

- Operators and the methods they map to
  - a + b   a.plus(b)
  - a – b   a.minus(b)
  - a * b   a.multiply(b)
  - a ** b   a.power(b)
  - ....
- For complete list, go to http://groovy.codehaus.org/Operator+Overloading

```
println 7 + 4       // => 11
println 7.plus(4)   // => 11
println 7 * 4           // => 28
println 7.multiply(4)    // => 28
println 'Sang' + 'Shin'    // SangShin
println 'Sang'.plus('Shin')    // SangShin
```

Special Operators

- Spread operator (*.)
- Elvis operator (?:)
- Safe navigation/Dereference operator (?.)
- Field operator (.@)
- Method closure operator (We will cover this in “Groovy Closure” presentation)

Spread operator (*.) for Collection Object

- Used to invoke a method on all members of a Collection object
- The result of using the spread operator is another Collection object

```
class Language {
    String lang
    def speak() { "$lang speaks." }
}

// Create a list with 3 objects. Each object has a lang property and a speak() method.
def list = [
    new Language(lang: 'Groovy'),
    new Language(lang: 'Java'),
    new Language(lang: 'Scala')
]

// Use the spread operator to invoke the speak() method.
assert list*.speak() == ['Groovy speaks.', 'Java speaks.', 'Scala speaks.']
assert list.collect{ it.speak() }  == ['Groovy speaks.', 'Java speaks.', 'Scala speaks.']

// We can also use the spread operator to access properties, but we don't need to,
// because Groovy allows direct property access on list members.
assert list*.lang  == ['Groovy', 'Java', 'Scala']
assert list.lang == ['Groovy', 'Java', 'Scala']
```

Elvis operator (?:)

- Used to shorten the ternary operator
- Useful in providing default value if it has not been set already

```
def testText1 = null
// Normal ternary operator.
def ternaryResult = (testText1 != null) ? testText1 : 'Hello Groovy1!'
println ternaryResult  // => Hello Groovy1

def testText2 = null
// The Elvis operator
def elvisResult2 = testText2 ?: 'Hello Groovy2!'
println elvisResult2 // => Hello Groovy2!

def testText3 = 'Sang Shin'
// The Elvis operator
def elvisResult3 = testText3 ?: 'Hello Groovy3!'
println elvisResult3 // => Sang Shin
```

Safe Navigation operator (?.)

- Used to avoid NullPointerException

```
class Person {
    String name
    int age
}

Person person  // person is null

// Java way of checking null vaiue
if (person != null){
    println "Name of the person is ${person.name}"
}

// Groovy way using Safe navigation operator
println "Name of the person is ${person?.name}"
```

Lab

Exercise 3: Groovy Syntax II
5610_groovy_basics.zip

Differences from Java

Differences from Java (1)

- Semicolons are optional
  - Use them if you like (though you must use them to put several statements on one line even in Groovy).
- The *return* keyword is optional
  - The result of last statement's evaluation gets returned
- You can use the *this* keyword inside static methods (which refers to this class)
- Methods and classes are public by default
- Attributes are private by default
- Inner classes are not supported
  - In most cases you can use closures instead

Differences from Java (2)

- The *throws* clause in a method signature is not checked by the Groovy compiler
  - Because there is no difference between checked and unchecked exceptions in Groovy
- You will not get compile errors like you would in Java for using undefined members or passing arguments of the wrong type
  - Because properties and methods can be dynamically added
- Basic packages are imported by default
  - No import statements are needed for these packages

Basic Packages that are imported

- java.io.*
- java.lang.*
- java.math.BigDecimal
- java.math.BigInteger
- java.net.*
- java.util.*

- groovy.lang.*
- groovy.util.*

New Features Added to Groovy

- Closures (Now Java 8 Lambda supports this)
- Native syntax for lists and maps
- GroovyMarkup and GPath support
  - GroovyMarkup enables building XML, HTML, SAX, W3C DOM, etc
  - GPath is a path expression language, which allows parts of nested structured data to be identified
- Native support for regular expressions
- Dynamic and static typing is supported - so you can omit the type declarations on methods, fields and variables
- You can embed expressions inside strings
- Lots of new helper methods added to the JDK
- Special operators

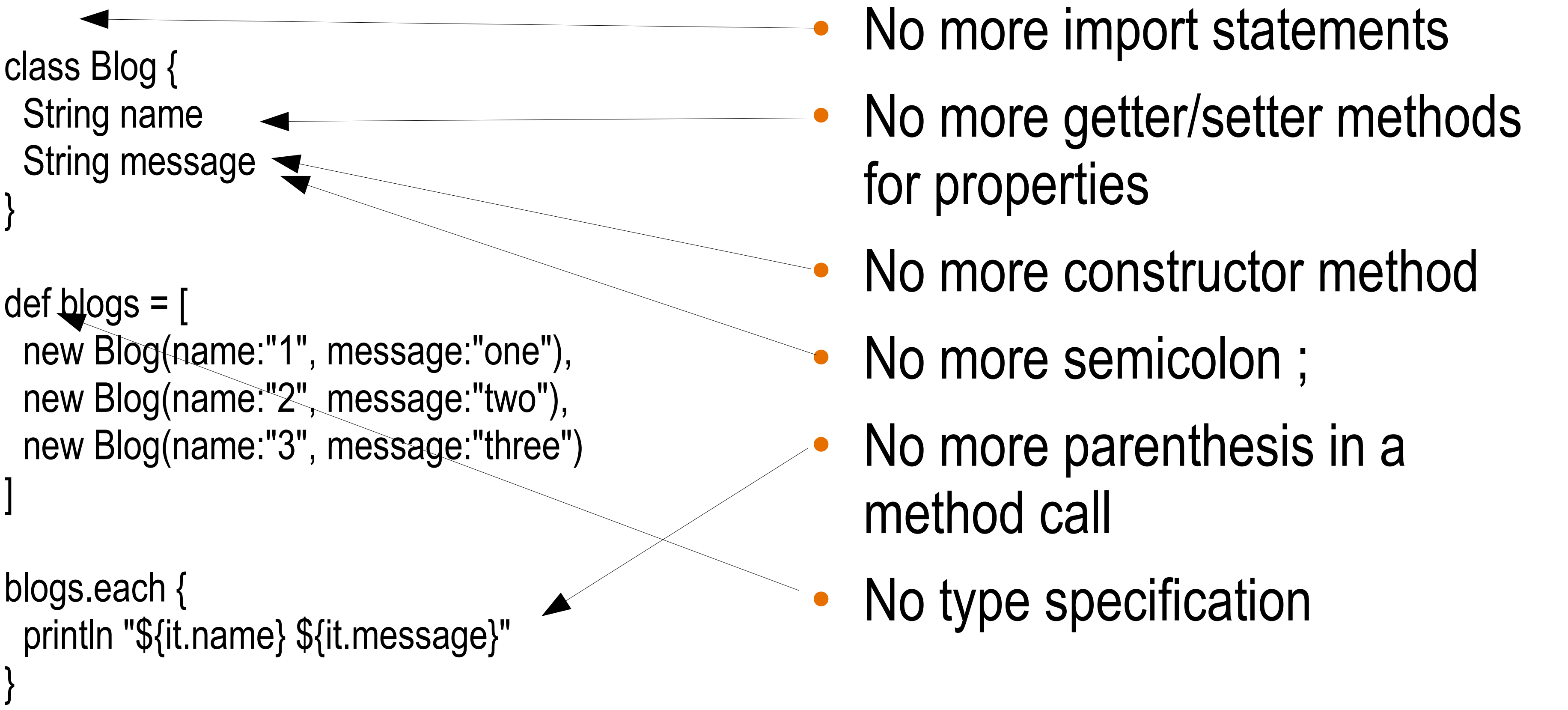
Refactoring Java Code into Groovy Code

Example Java Code - POJO

```
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

public class Blog {
  private String name;
  private String message;

  public Blog() {}

  public Blog(String name, String Message) {
    this.name = name;
    this.message = Message;
  }

    public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getMessage() {
    return message;

  }
```

```
  public void setMessage(String Message) {
    this.message = Message;
  }


  public static void main(String[] args) {
    List Blogs = new ArrayList();
    Blogs.add(new Blog("1", "one"));
    Blogs.add(new Blog("2", "two"));
    Blogs.add(new Blog("3","three"));


    for(Iterator iter = Blogs.iterator();iter.hasNext();) {
      Blog Blog = (Blog)iter.next();
        System.out.println(Blog.getName() + " " + Blog.getMessage());
    }
  }
}
```

Groovy Code – POGO (1)



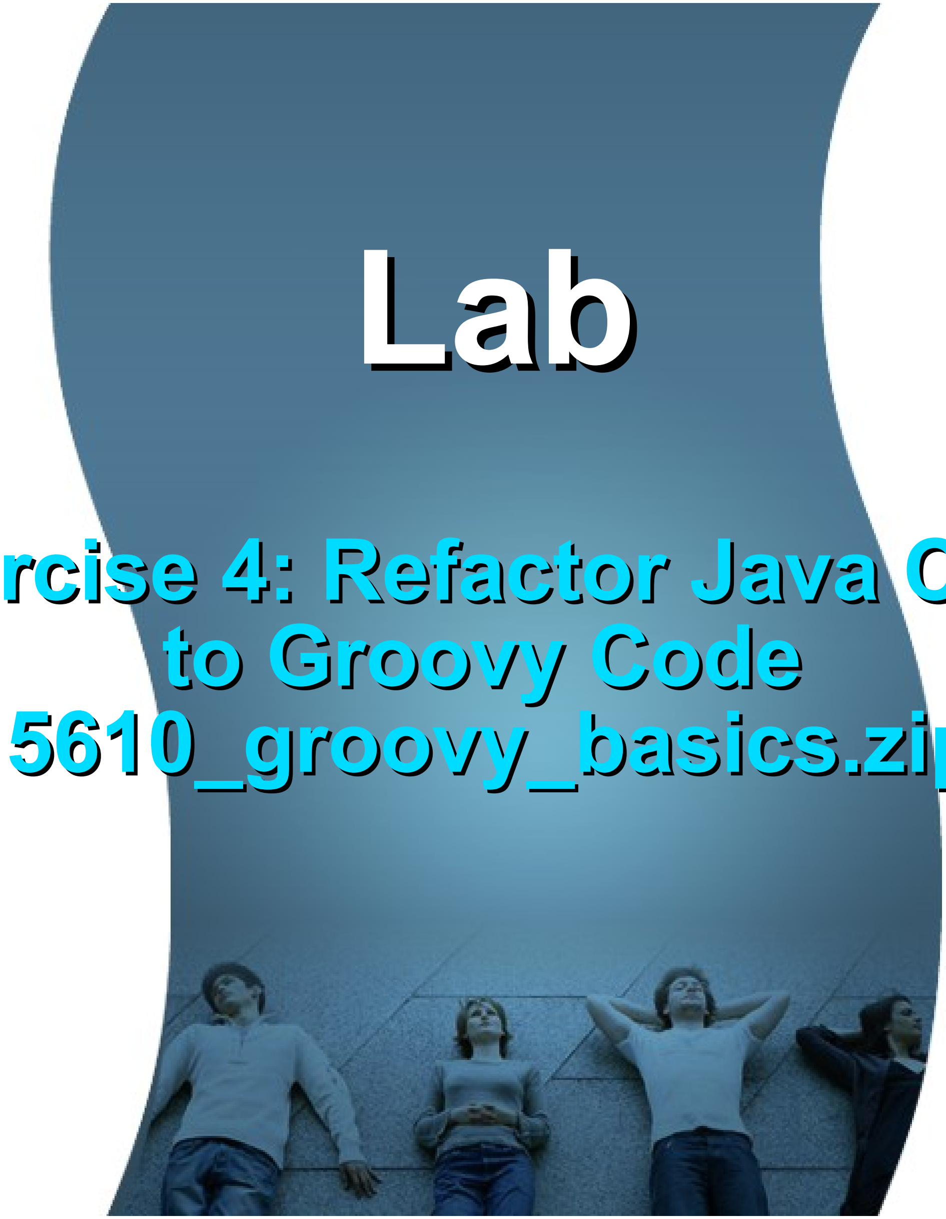
```
class Blog {
  String name
  String message
}

def blogs = [
  new Blog(name:"1", message:"one"),
  new Blog(name:"2", message:"two"),
  new Blog(name:"3", message:"three")
]

blogs.each {
  println "${it.name} ${it.message}"
}
```

- No more import statements
- No more getter/setter methods for properties
- No more constructor method
- No more semicolon ;
- No more parenthesis in a method call
- No type specification

Groovy Code – POGO (2)

```
class Blog {
  String name
  String message
}

def blogs = [
  new Blog(name:"1", message:"one"),
  new Blog(name:"2", message:"two"),
  new Blog(name:"3", message:"three")
]

blogs.each {
  println "${it.name} ${it.message}"
}
```

- No more ArrayList class
  - Use [..] notation
- No more “for” loop
  - Use closure instead
- No more System.out.println()
  - Use println
- No more main() method
  - main() method gets added by Groovy



Lab

Exercise 4: Refactor Java Code to Groovy Code 5610_groovy_basics.zip

Groovy and Java Interoperability

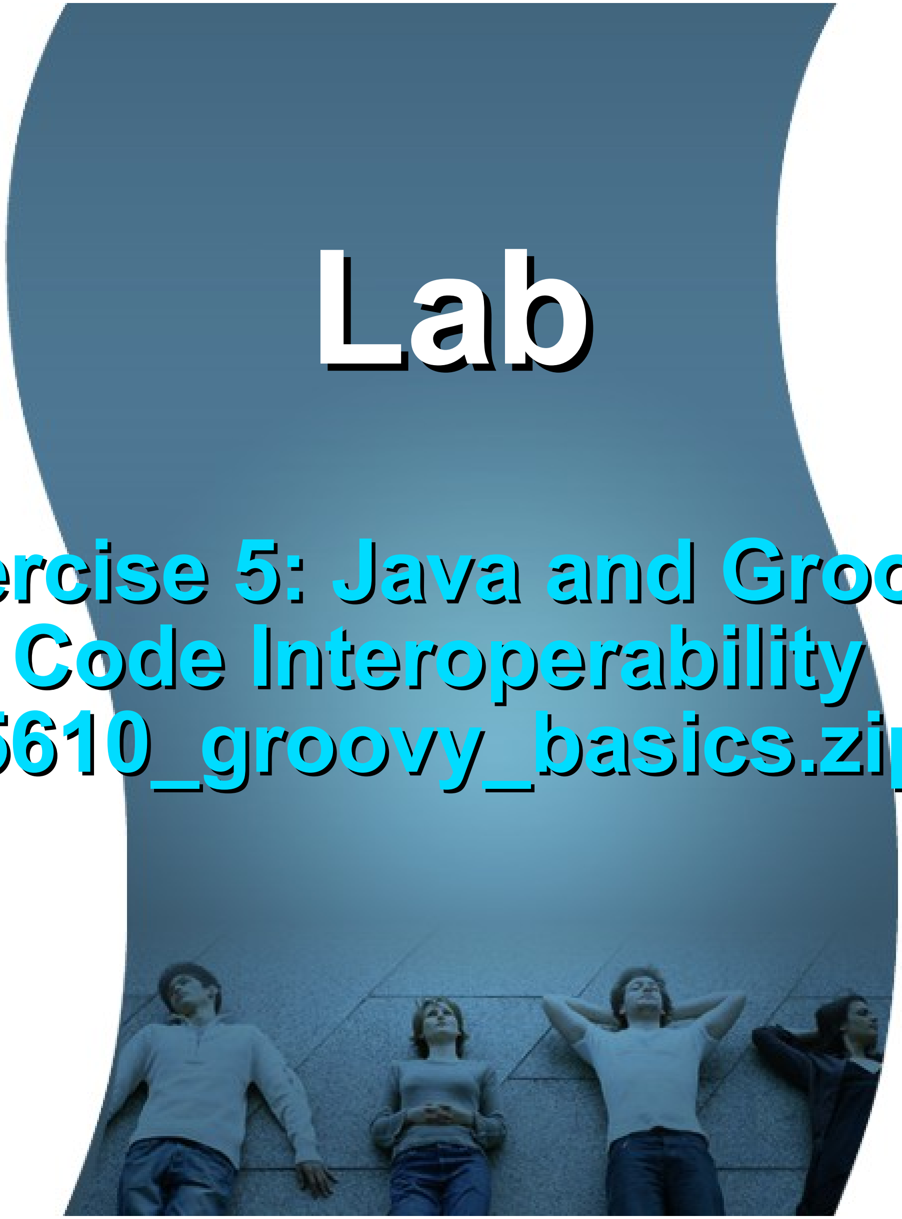
Interoperability with Java

- Groovy code can call Java code
- Java code can call Groovy code

JavaBean is used in Groovy Code

```
// This is JavaBean written in Java

public class Blog {
  private String name;
  private String message;

  public Blog() {}

   public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getMessage() {
    return message;

  }
  public void setMessage(String message) {

    this.message = message;

  }
}
```

```
// Groovy code that uses JavaBean
// In Groovy, every object has “metaClass” property
Blog.metaClass.sayHello = {
    println "Hello"
}


def myBlog = new Blog(name:"4", message:"four")
myBlog.sayHello()
```



Lab

Exercise 5: Java and Groovy Code Interoperability 5610_groovy_basics.zip

Groovy Ecosystem

Groovy Ecosystem

- Frameworks
  - Grails - Web application framework
  - Griffon - MVC Desktop application framework
- Build system
  - Gant - Ant scripting language
  - Gradle – Build automation
- Testing
  - Spock - Testing framework
  - Geb – Functional testing
- Code quality
  - CodeNarc - Groovy code analyzer
- Many more

Learn with Passion!
JPassion.com