IDT-GD-200 - Introduction to Groovy Scripting

A comprehensive introduction to Groovy programming language

identitrain

Groovy and Course Overview

identitrain

Course Overview

The course is divided into four 3-hour days - each with 2 labs. There will be approximately 1 hour of instruction, then 30 minutes of lab/interactive questions.

- Throughout the course there will be Checkpoint Questions that will reinforce important concepts.

Overview of Groovy

- Groovy extends the Java Language rather than replacing it.
- Groovy is a powerful, optionally typed and dynamic language for the Java platform aimed at improving developer productivity. The rich ecosystem has existing projects for web development, automation, build and testing tools, code analysis, and GUI development.

Course Concepts

- A few things we will review throughout this course:
- Groovy syntax and language features
- Builders to create object graphs, mark up, or JSON
- Connecting to a database and using Sql libraries
- Compile time and runtime metaprogramming concepts
- Domain Specific Language support

Online Resources

- Groovy Home Page: https://groovy-lang.org/
- Documentation Page: https://groovy-lang.org/documentation.html
- Groovy Source Code: https://github.com/apache/groovy
- GroovyDoc: https://groovy-lang.org/api.html
- Issue Tracker: https://issues.apache.org/jira/projects/GROOVY/issues/
- Slack: https://groovycommunity.com/
- REPL: http://groovyconsole.appspot.com/

Additional Resources

- Sources of Additional Resources Outside of this Course:
- Online Courses
- Social Media
- Slack
- Books
- Groovy in Action, Second Edition By Dierk König, Paul King, Guillaume Laforge, Hamlet D’Arcy, Cédric Champeau, Erik Pragt, and Jon Skeet
- Making Java Groovy
- By Ken Kousen
- Programming Groovy 2
- By Venkat Subramaniam
- Offline as Well as Online

How to Get the Most Out of This Course

- Complete the exercises
- Answer the course checkpoint questions
- Ask your own questions
- Take note of other people's questions
- Use the additional resources provided at the end of the course

Getting Groovy Set Up

identitrain

Getting Started 1

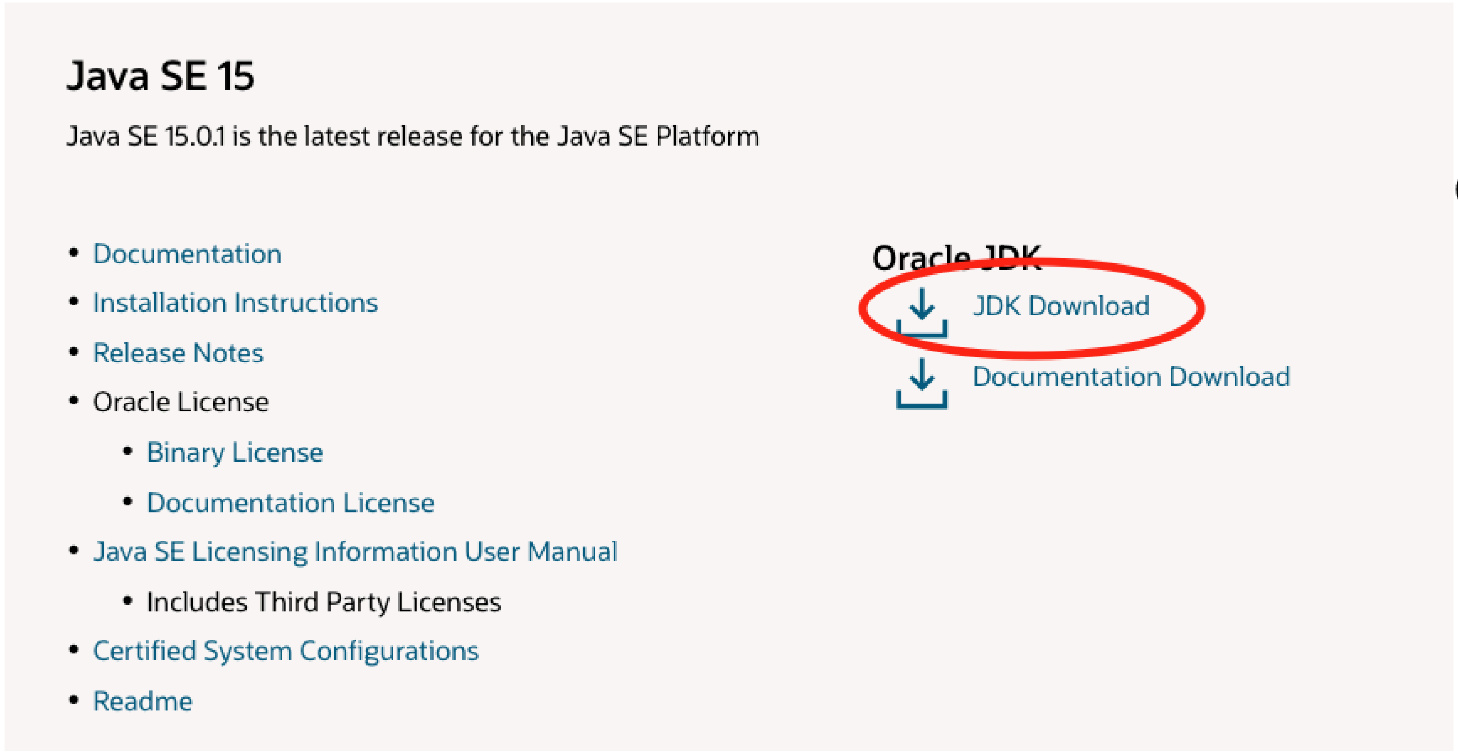
- Complete the following steps:
- Connect to the Internet
- Install Java
- Download Groovy from the Groovy Website
- Install All the Tools We Will Need
- Setting up the Development Environment

Getting Started 2

- Confirm Java Version

Getting Started 3

- Installing JDK https://www.oracle.com/java/technologies/javase-downloads.html



Java SE 15

Java SE 15.0.1 is the latest release for the Java SE Platform

- Documentation
- Installation Instructions
- Release Notes
- Oracle License
  - Binary License
  - Documentation License
- Java SE Licensing Information User Manual
  - Includes Third Party Licenses
- Certified System Configurations
- Readme

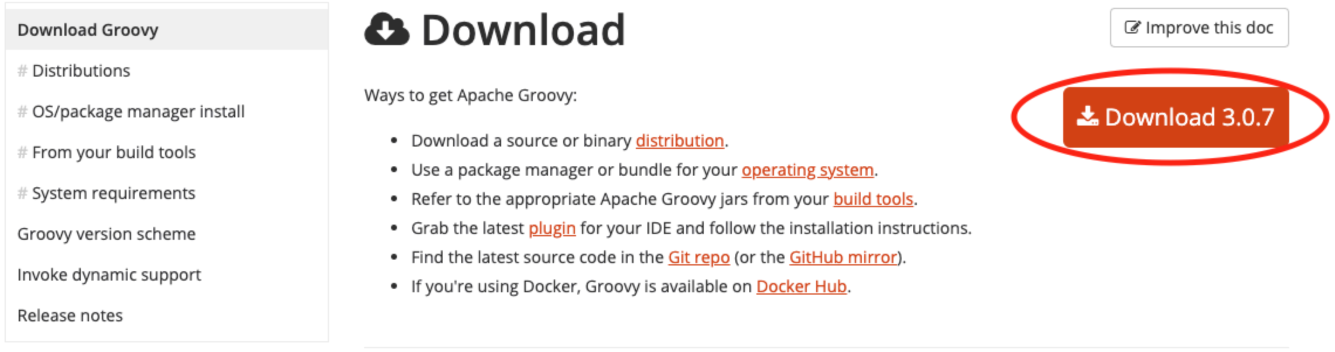
Oracle JDK

JDK Download

Documentation Download

Getting Started 4

https://groovy.apache.org/download.html



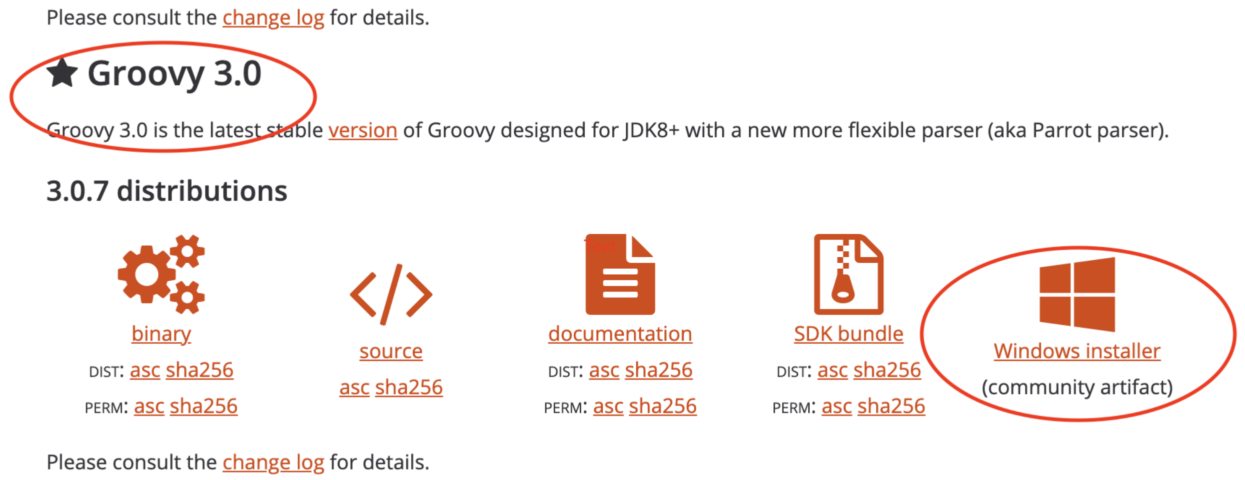
Download

Ways to get Apache Groovy:

- Download a source or binary distribution.
- Use a package manager or bundle for your operating system.
- Refer to the appropriate Apache Groovy jars from your build tools.
- Grab the latest plugin for your IDE and follow the installation instructions.
- Find the latest source code in the Git repo (or the GitHub mirror).
- If you're using Docker, Groovy is available on Docker Hub.

Getting Started 5

- Further Down the Groovy Download Page



Please consult the change log for details.

★ Groovy 3.0

Groovy 3.0 is the latest stable version of Groovy designed for JDK8+ with a new more flexible parser (aka Parrot parser).

3.0.7 distributions

binary
DIST: asc sha256
PERM: asc sha256

source
asc sha256

documentation
DIST: asc sha256
PERM: asc sha256

SDK bundle
DIST: asc sha256
PERM: asc sha256

Windows installer
(community artifact)

Please consult the change log for details.

Getting Started 6

- Groovysh is the shell access to the Groovy compiler.
- It is installed with Groovy by default
- The manual page is at http://groovy-lang.org/groovysh.html
- Demonstration of:
- Single line statements and immediate results
- Multi-line statements and the Groovysh instance retains the context
- Groovysh

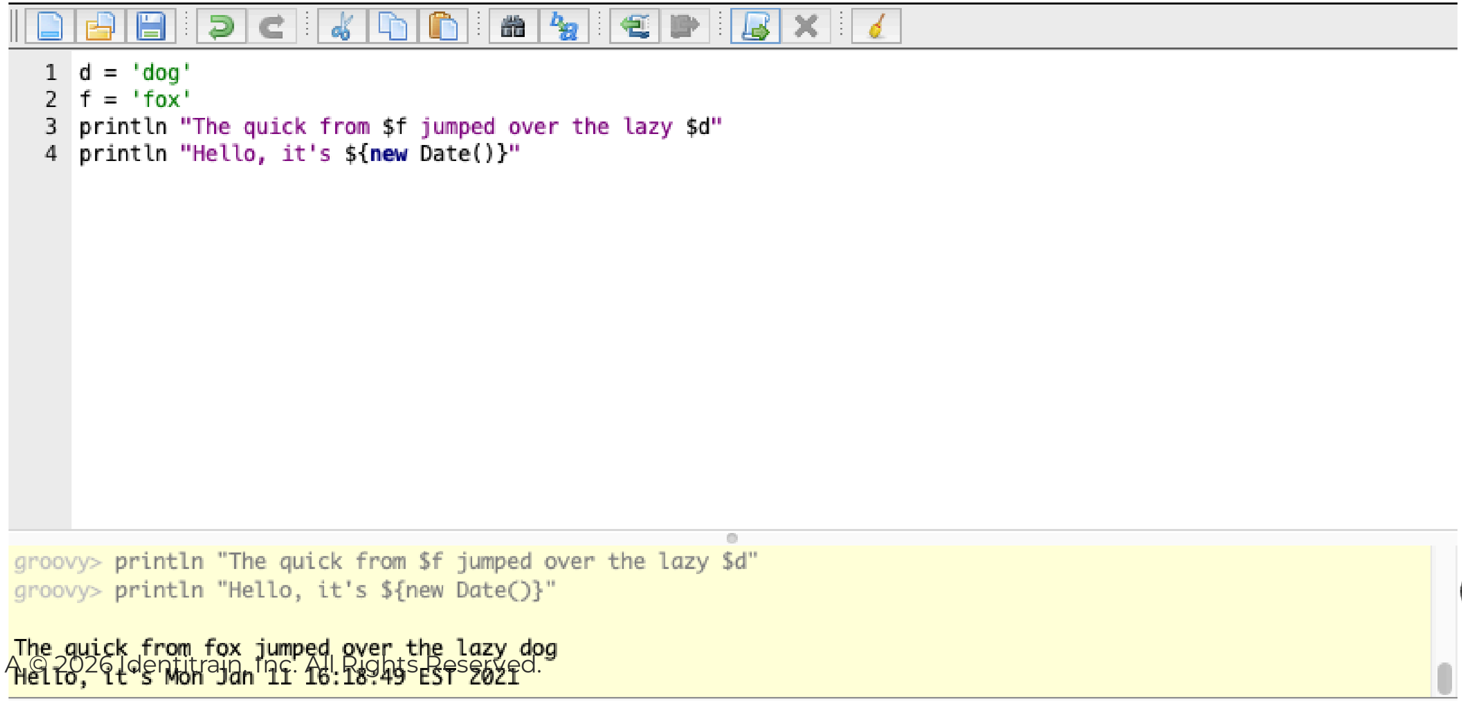
identitrain

Getting Started 7

- Groovyc is the command line compiler for Groovy –
- You don’t have to run it before execution
- The output from Groovyc is Java bytecode
- Normal input is an *.groovy file, output is a corresponding *. Class file
- There is a link to full documentation on the Groovy documentation page
- Current address is http://groovy-lang.org/groovyc.html
- Groovyc

Getting Started 8

- The Groovy Console
- Interactive Text Area
- Syntax highlighting
- Console output when running



```
d = 'dog'
f = 'fox'
println "The quick from $f jumped over the lazy $d"
println "Hello, it's ${new Date()}"
```

```
groovy> println "The quick from $f jumped over the lazy $d"
groovy> println "Hello, it's ${new Date()}"

The quick from fox jumped over the lazy dog
Hello, it's Mon Jan 11 16:18:49 EST 2021
```

Getting Started 9

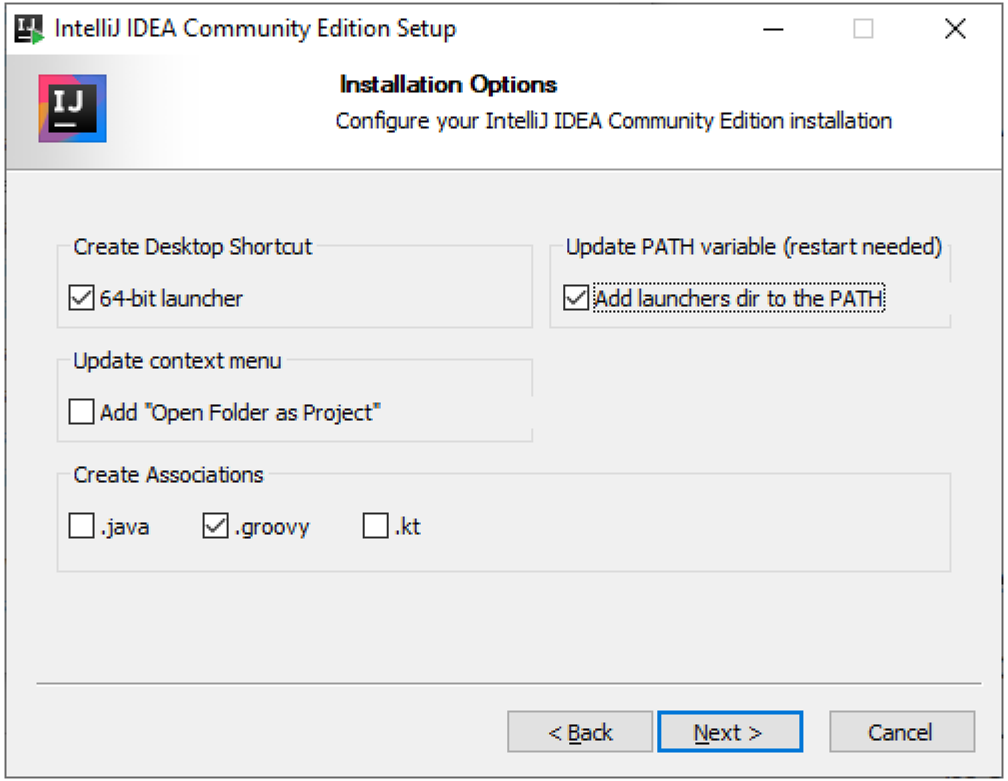
- Script -> Inspect AST
- Notice the dropdown for At end of Phase
- Notice the new constructors and the @Generated annotation
- Notice the added interface
- Userful ones can be found as annotations here: https://docs.groovy-lang.org/latest/html/gapi/groovy/transform/package-summary.html
- The AST Browser (from GroovyConsole)

Getting Started 10

- An IDE for Groovy available in open source (free) or commercial versions
- Intellij Idea

Getting Started 11

- Installing Intellij Idea



IntelliJ IDEA Community Edition Setup

Installation Options

Configure your IntelliJ IDEA Community Edition installation

Create Desktop Shortcut

- [✓] 64-bit launcher

Update PATH variable (restart needed)

- [✓] Add launchers dir to the PATH

Update context menu

- [ ] Add "Open Folder as Project"

Create Associations

- [ ] .java
- [✓] .groovy
- [ ] .kt

Configuring IntelliJ

- Add groovy location to IntelliJ etc.
- Show the project settings and stuff.
- Show how IntelliJ runs groovy files.
- Some extra settings

Checkpoint

Which tool allows you to explore the AST?

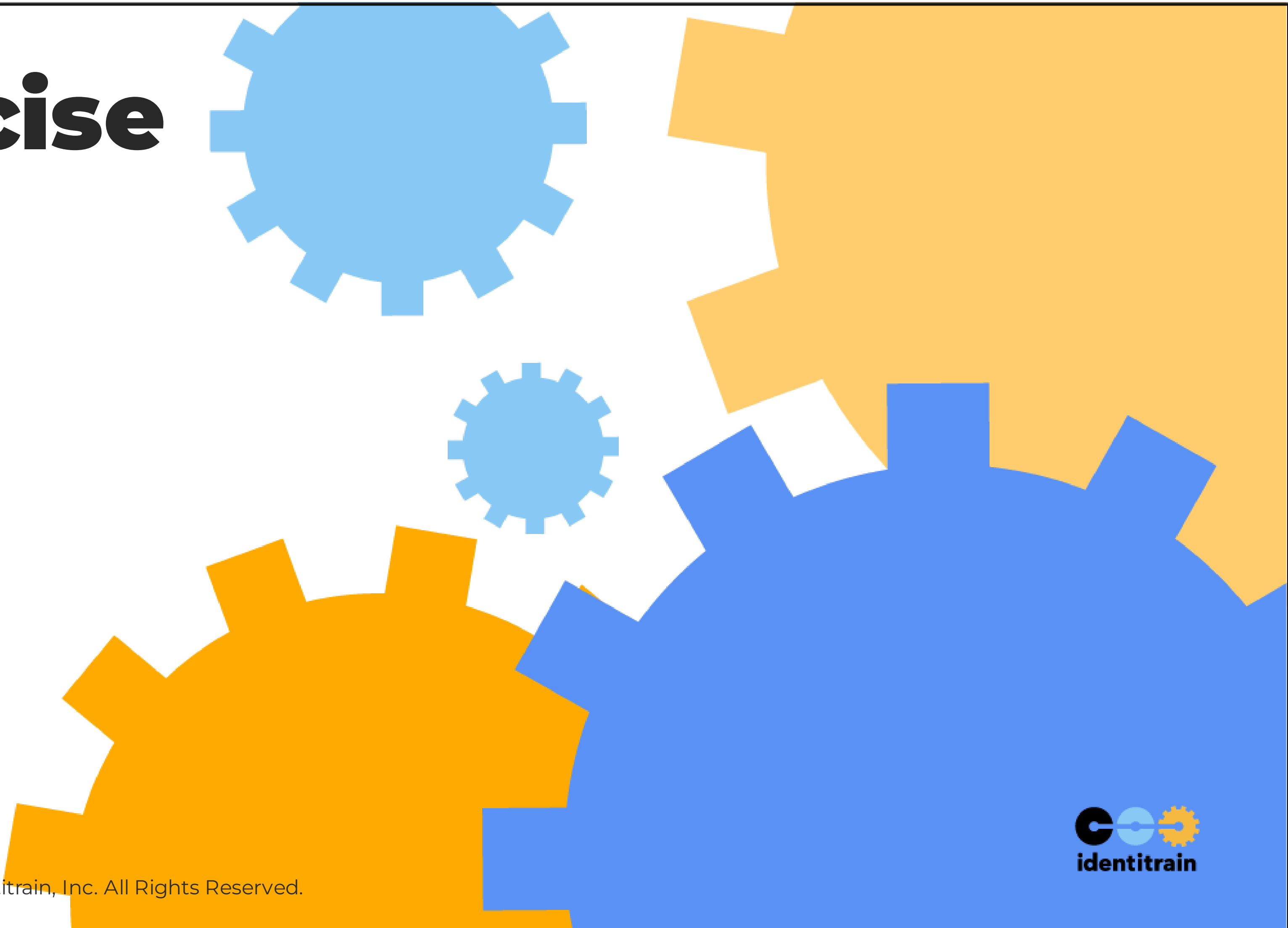
- GroovyConsole

Checkpoint

What is a REPL?

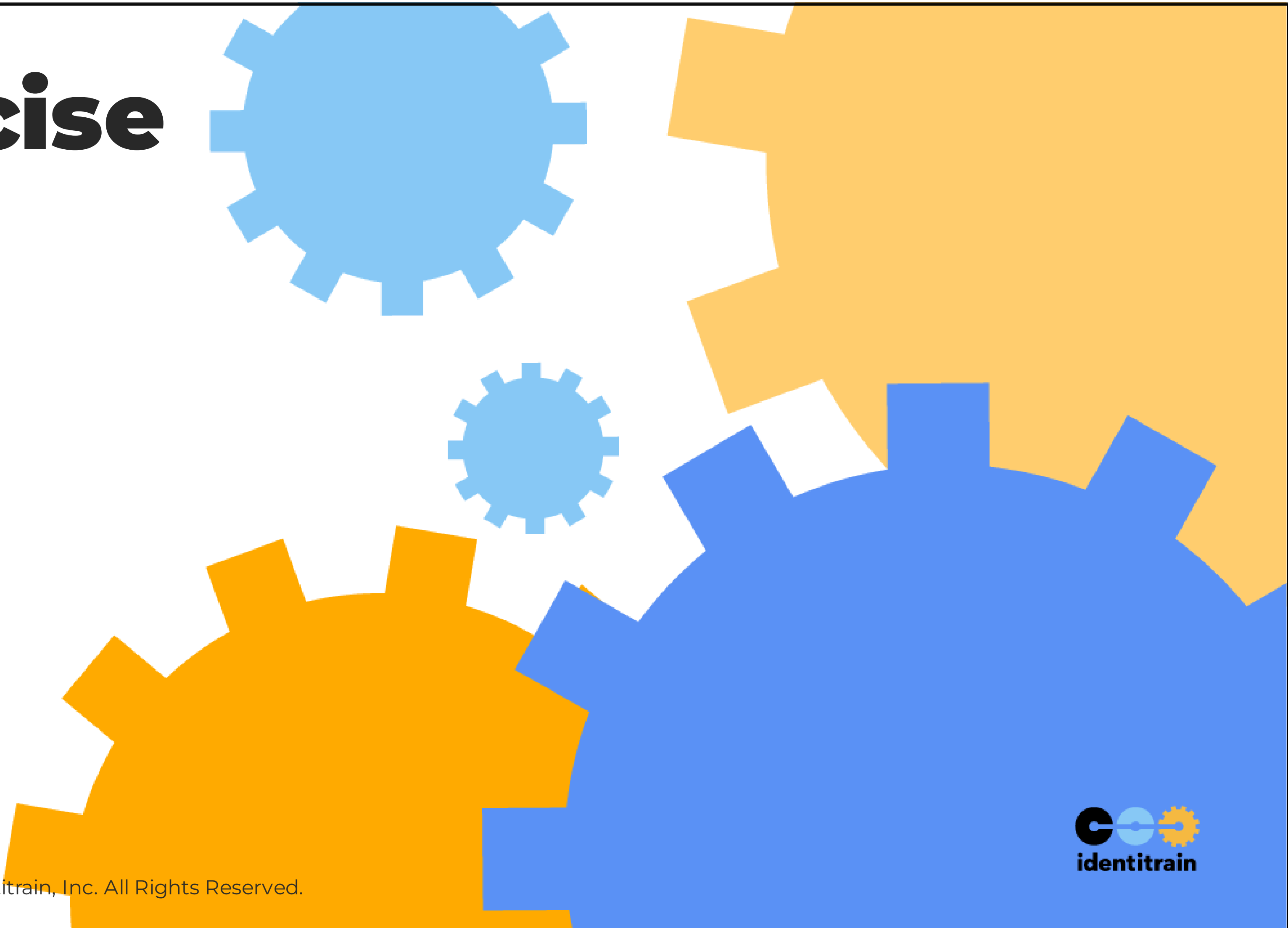
- Read Execute Print Loop

Exercise



identitrain

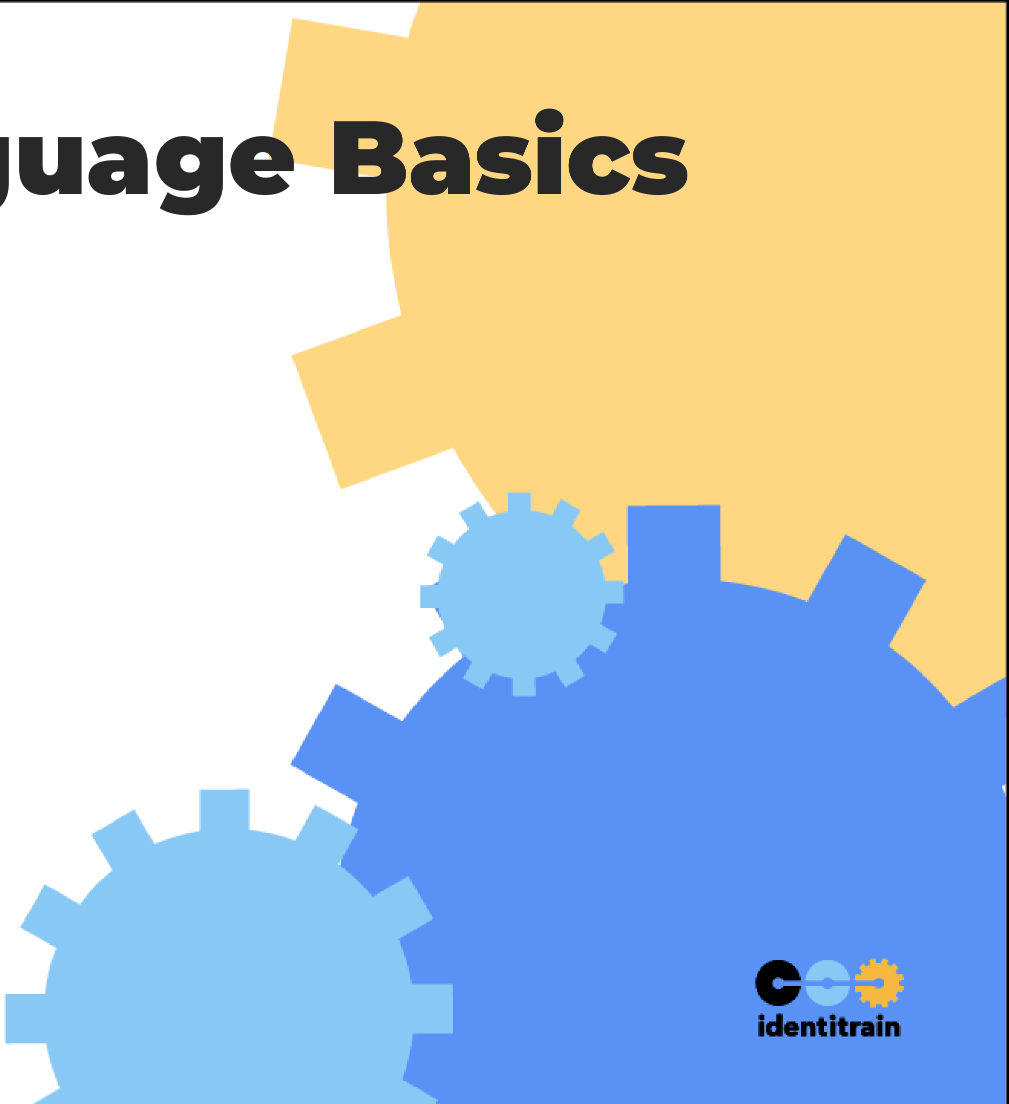
Exercise



identitrain

Questions?

Groovy Language Basics

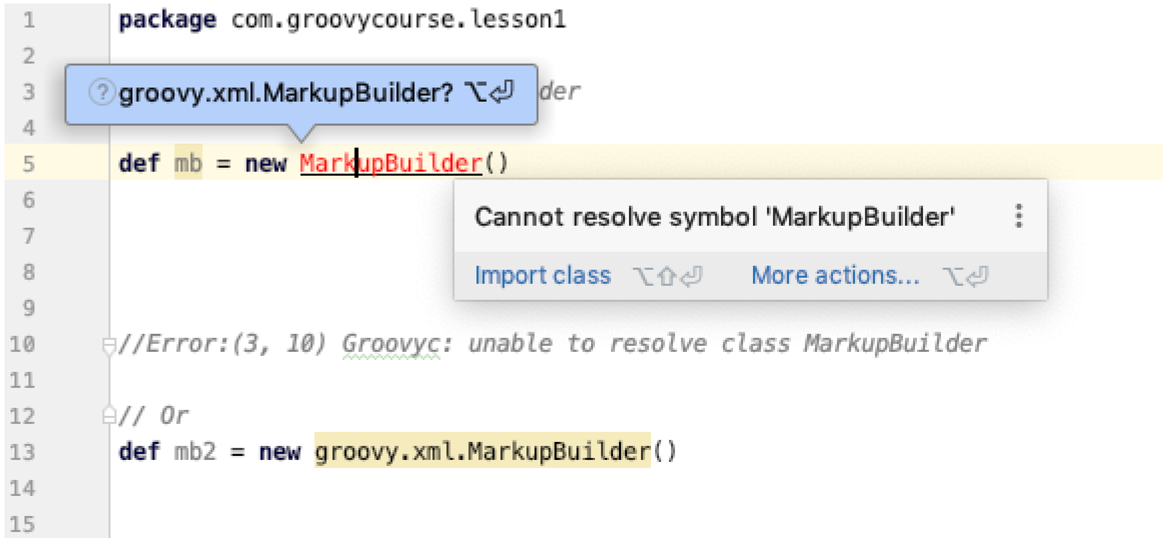


Groovy Basics – Imports

- Default Imports
- java.io.*
- java.lang.*
- java.math.BigDecimal
- java.math.BigInteger
- java.net.*
- java.util.*
- groovy.lang.*
- groovy.util.*
- Importing other classes
- import groovy.xml.MarkupBuilder
- import groovy.xml.*

Groovy Basics – Imports Continued

- IntelliJ Code Completion
- IntelliJ has built-in code completion support for groovy libraries.
- It will add the libraries to the import statements at the beginning of the groovy files. You can add other 3rd party libraries to the build path and it will use code complete with those also.



```
package com.groovycourse.lesson1

groovy.xml.MarkupBuilder? ⌥⏎           der

def mb = new MarkupBuilder()
                              Cannot resolve symbol 'MarkupBuilder'
                              Import class ⌥⇧⏎    More actions... ⌥⏎



//Error:(3, 10) Groovyc: unable to resolve class MarkupBuilder

// Or
def mb2 = new groovy.xml.MarkupBuilder()
```

Groovy Basics – Keywords

- Keywords
- Java Keywords are also reserved in Groovy: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html
- Groovy also introduces its own Keywords: http://groovy-lang.org/syntax.html#_keywords

Groovy Basics – Numbers

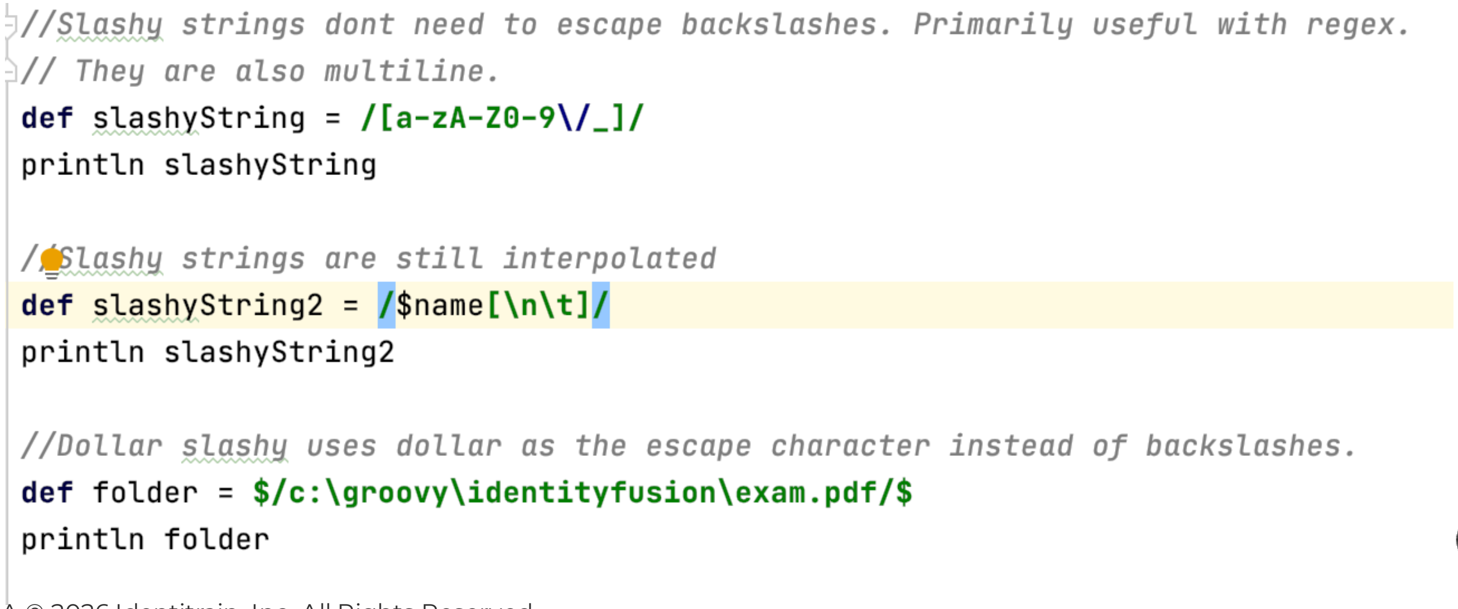
- Basic Types are Classes
- In Groovy, all of the native types are converted to their object types.
- If no type is given, then numbers become numeric types based on their size and precision.
- Whole numbers become Integer, Long, or BigInteger based on the size of the number.
- Floating-point numbers (those with a decimal) become BigDecimal by default https://groovy-lang.org/syntax.html#_number_type_suffixes

Groovy Basics - Strings

- Java Strings and Groovy Strings
- Strings are declared using single quotes
- All other Strings are GStrings and support interpolation.
- Interpolation allows expressions to be evaluated during the string use.
- Multiline strings begin with 3 consecutive ‘’’ or “”” and are terminated by the same sequence.

Groovy Basics – Strings Continued

- Slashy Strings and Dollar Slashy
- You can also declare a string with a / and end it with another /.
- It supports interpolation and does not require \ to be escaped. You can declare a string with $/andenditwith/$ to use a dollar as the escape character instead.
- (These are mostly to make life easier with RegEx - covered later)



```
//Slashy strings dont need to escape backslashes. Primarily useful with regex.
// They are also multiline.
def slashyString = /[a-zA-Z0-9\/_]/
println slashyString

//Slashy strings are still interpolated
def slashyString2 = /$name[\n\t]/
println slashyString2

//Dollar slashy uses dollar as the escape character instead of backslashes.
def folder = $/c:\groovy\identityfusion\exam.pdf/$
println folder
```

Groovy Basics – Simple Data Types

- Simple Data Types There are no native types in Groovy. Everything is an object. Even if you declare it as the native type, it will create the Object automatically.
- Integral Types:
- byte - Byte
- char - Char
- short - Short
- int - Integer
- long - Long
- Decimal Types:
- float - Float
- double - Double

Groovy Basics – Operators

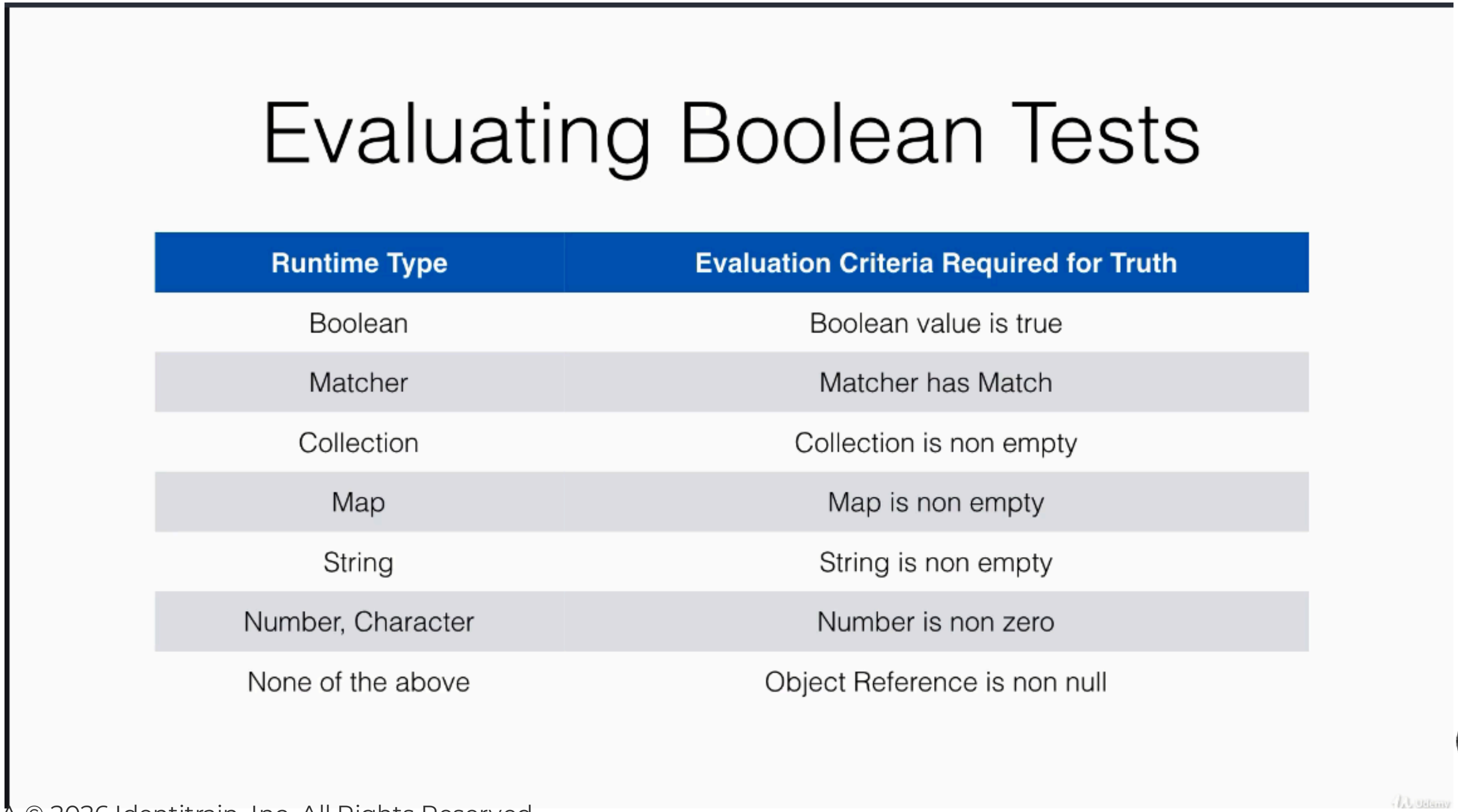
- Groovy Documentation https://www.groovy-lang.org/operators.html
- Groovy has basic arithmetic operators: +,-,*,/,%,**, ++, –
- Basic arithmetic operators can also be used as assignment by adding an = after the operator.
- Groovy has several relational operators: ==, !=, <, <=, >, >=, ===, !==, <=>
- Groovy has Logical operators: &&, ||, ! Bitwise operators: &, |, ^, ~, <<, >>, >>> *Note these should not be used for floating point numbers or string types.
- *Full table of precedence can be found here: http://groovy-lang.org/operators.html#_operator_precedence

Groovy Basics – More Operators

- Groovy Documentation
- Ternary & Elvis Operator - is covered in control structures
- Elvis Operator - is covered in control structures
- Object Operators - to be covered in Object Oriented section
- Regex Operators - to be covered with Regex section

Groovy Control Structures – Groovy Truth

- It’s Different From Java



Evaluating Boolean Tests

Runtime Type	Evaluation Criteria Required for Truth
Boolean	Boolean value is true
Matcher	Matcher has Match
Collection	Collection is non empty
Map	Map is non empty
String	String is non empty
Number, Character	Number is non zero
None of the above	Object Reference is non null

Checkpoint

What does the statement below print?

- def name = “Patrick”

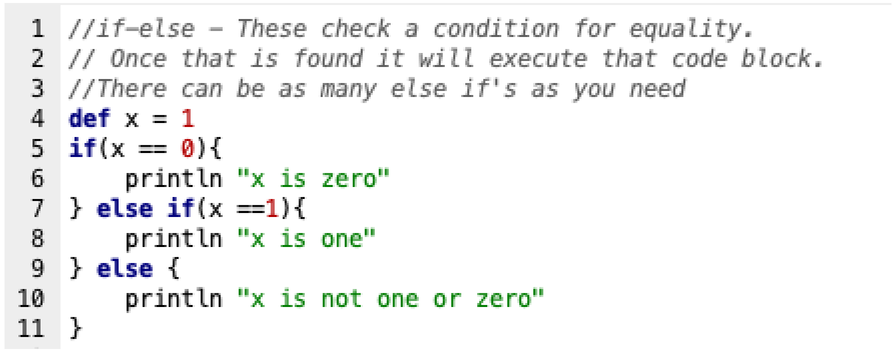
Checkpoint

What is the value of i after running?

- def i = 1

Groovy Basics – Control Structures

- If Statements
- If Statements check whether a value is true or not, then will execute code. Else if is another condition to check and is optional - you can have as many of these as you want in your if - else if - else statement.
- Else blocks are optional and will execute if all other if statements are not true
- Shortcuts
- Ternary Operator - condition ? false result: true result
- Elvis Operator - condition ?: true result
- Elvis Assignment - var ?= true result



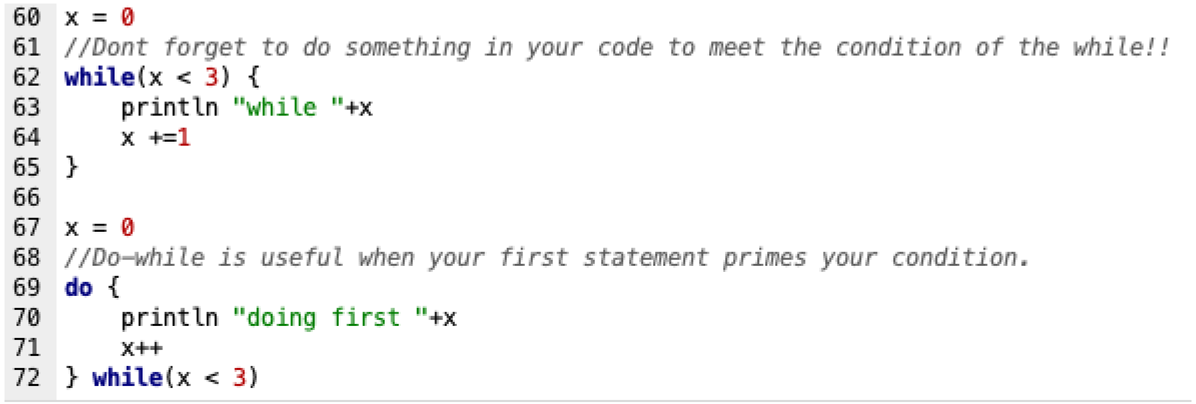
```
//if-else - These check a condition for equality.
// Once that is found it will execute that code block.
//There can be as many else if's as you need
def x = 1
if(x == 0){
    println "x is zero"
} else if(x ==1){
    println "x is one"
} else {
    println "x is not one or zero"
}
```

Groovy Basics – Control Structures

- Switch Statements
- Switch can also check for equality
- It can check if a value is in a list
- It can check if a value is a specific class type
- It can check if a value matches a regex
- It can check if a closure evaluates to true
- It has a default case if none of the others are matched Note the use of break - without it, the remaining statements would execute after the case is matched.

Groovy Basics – Control Structures

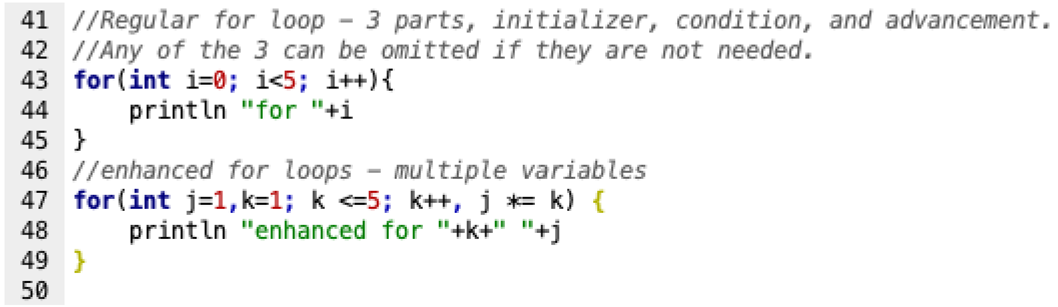
- While and Do-While Statements A while statement is a condition that is checked, and if the condition is met, the body of the code is executed. It will execute the body UNTIL the condition returns false. There is also a do - while loop which is structured a bit differently where you do something BEFORE you check the condition. But if the condition is still true it will execute the do block again until the condition is false.
- Don't forget to do something in your code blocks to advance whatever your condition is!



```
x = 0
//Dont forget to do something in your code to meet the condition of the while!!
while(x < 3) {
    println "while "+x
    x +=1
}

x = 0
//Do-while is useful when your first statement primes your condition.
do {
    println "doing first "+x
    x++
} while(x < 3)
```

Groovy Basics – Control Structures

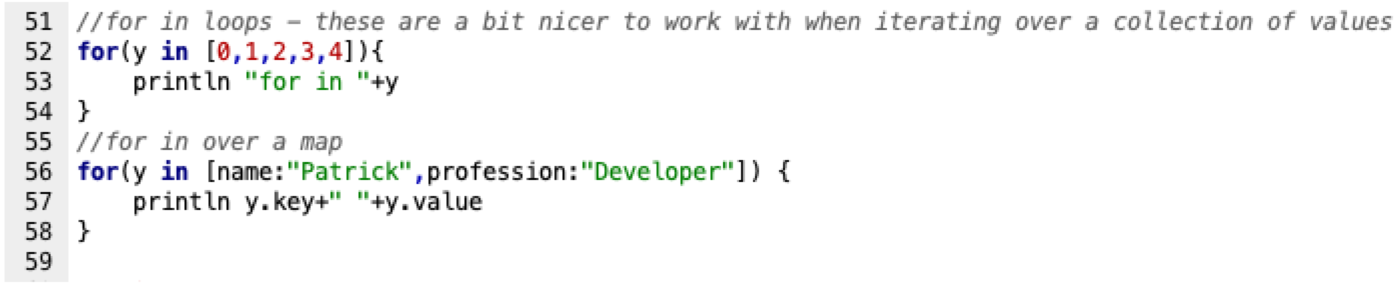
- For Statements
- Used to execute something a number of times
- Three parts to a for statement separated by ;
- Initializer - called once
- Condition - checked each iteration
- Advancement - executed on each iteration
- All 3 parts are optional Think of them as structured while loops. They make it clearer in many cases what the advancement of the loop is.



```groovy
//Regular for loop - 3 parts, initializer, condition, and advancement.
//Any of the 3 can be omitted if they are not needed.
for(int i=0; i<5; i++){
    println "for "+i
}
//enhanced for loops - multiple variables
for(int j=1,k=1; k <=5; k++, j *= k) {
    println "enhanced for "+k+" "+j
}

```

Groovy Basics – Control Structures

- For - In Statements
- A shorthand for loop that will execute for each element in a collection.
- It will always go through each of the elements in the collection (unless you use a break)



```groovy
//for in loops - these are a bit nicer to work with when iterating over a collection of values
for(y in [0,1,2,3,4]){
    println "for in "+y
}
//for in over a map
for(y in [name:"Patrick",profession:"Developer"]) {
    println y.key+" "+y.value
}

```

Control Structures – Exception Handling

- Try, Catch, Finally
- Introduction to Docker
- Try blocks allow you to take some action if something unexpected happens.
- Catch blocks let you take a specific action based on the type of exception.
- Finally blocks ‘always’ run regardless of whether an exception was encountered or not

Groovy Features – Comments

- Comments in Groovy
- #!/usr/bin/env groovy
- //Single line comment
- /*
- Block comment
- */
- /**
- - GroovyDoc comment - these are used in generating API docs
- */

Groovy Features – Assertions

- Demonstrated in Groovy Console
- assert true
- assert
- 1
- ==
- 1
- assert
- 1
- .is(
- 1
-)
- assert
- 1
- == ((
- 3
- ▪
- 10
-) *

Checkpoint

What does this output?

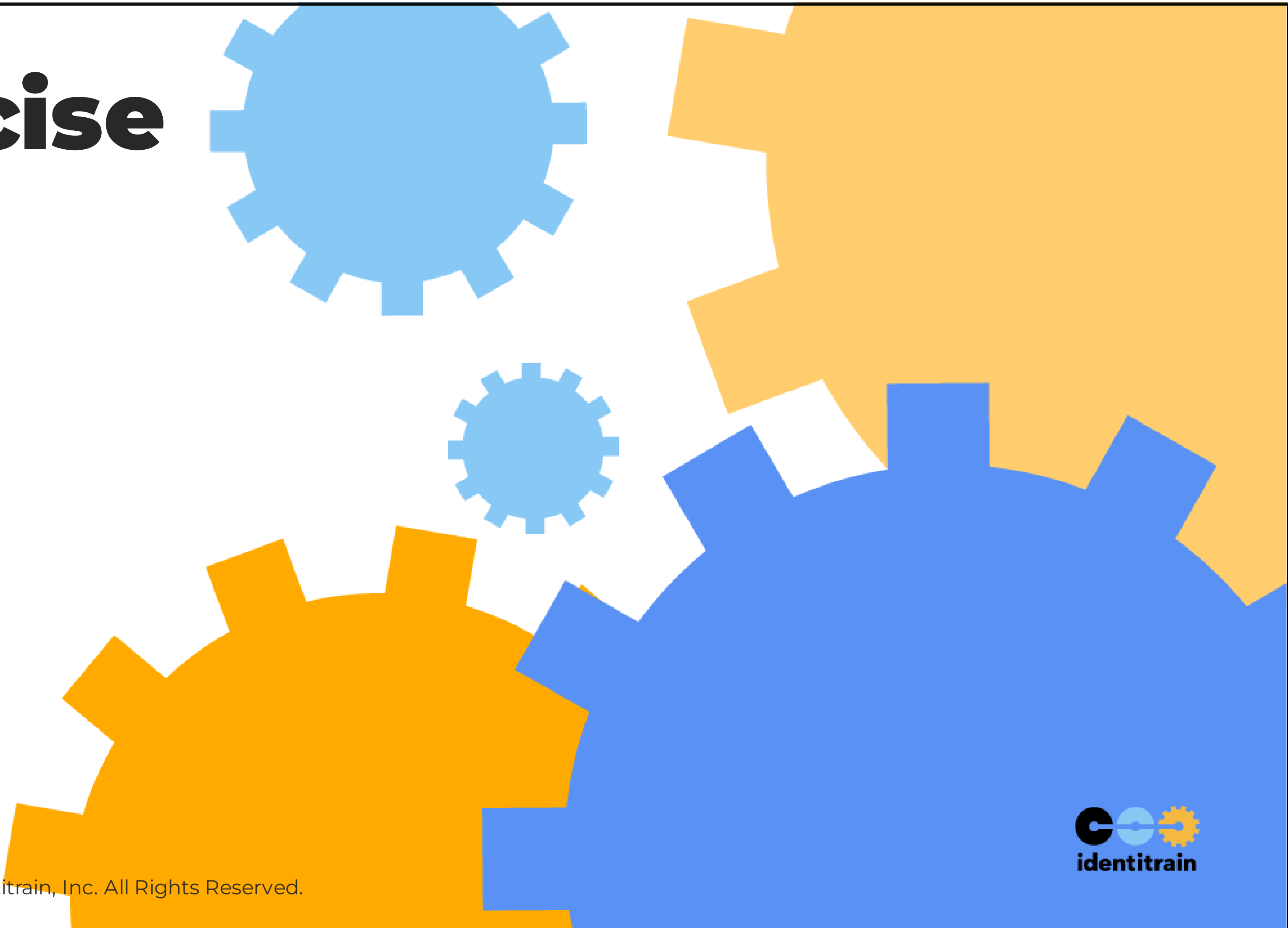
- List numbers = [1,2,3]

Checkpoint

How could you rewrite this for loop with a while statement?

- for(def i=0; i<10; i+=2) {

Exercise

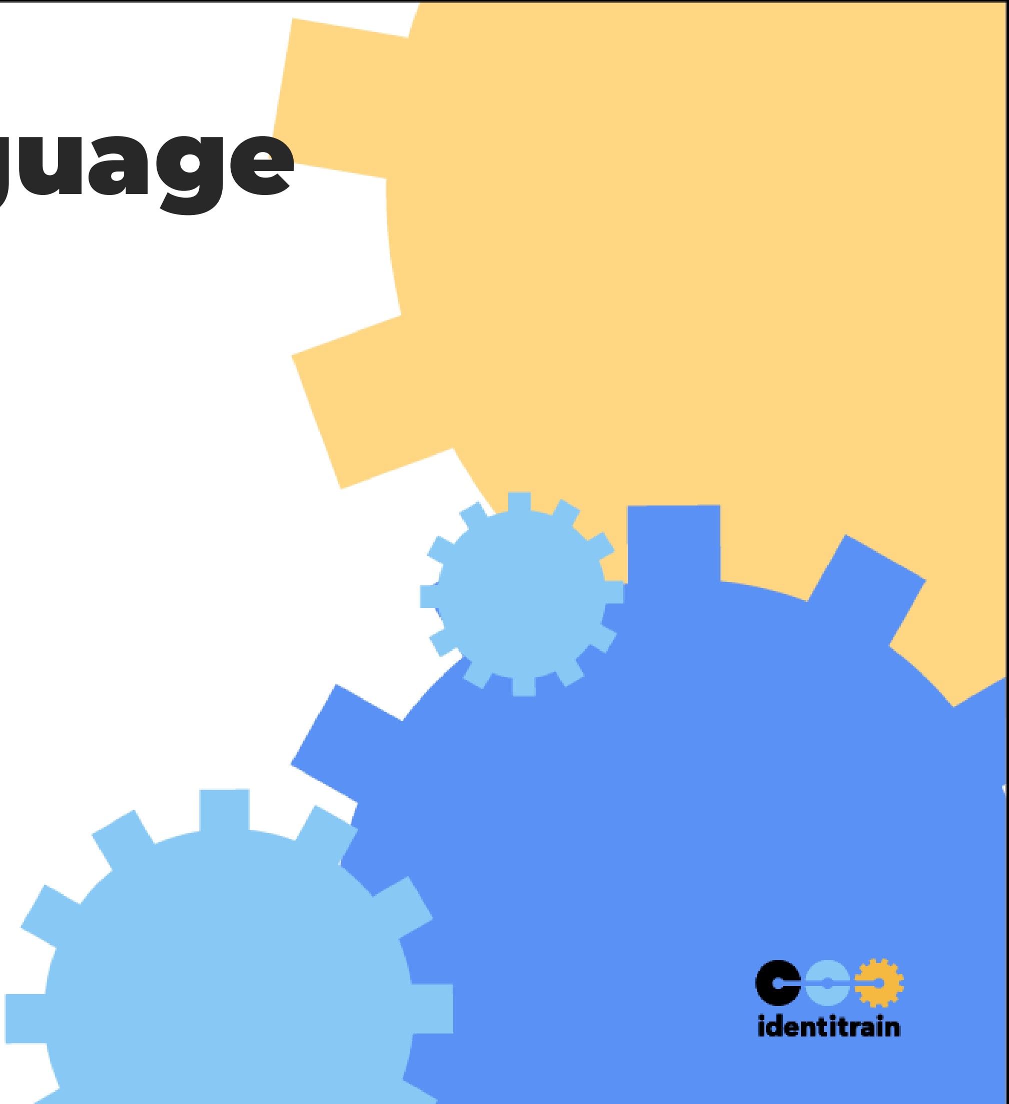


Exercise

identitrain

Questions?

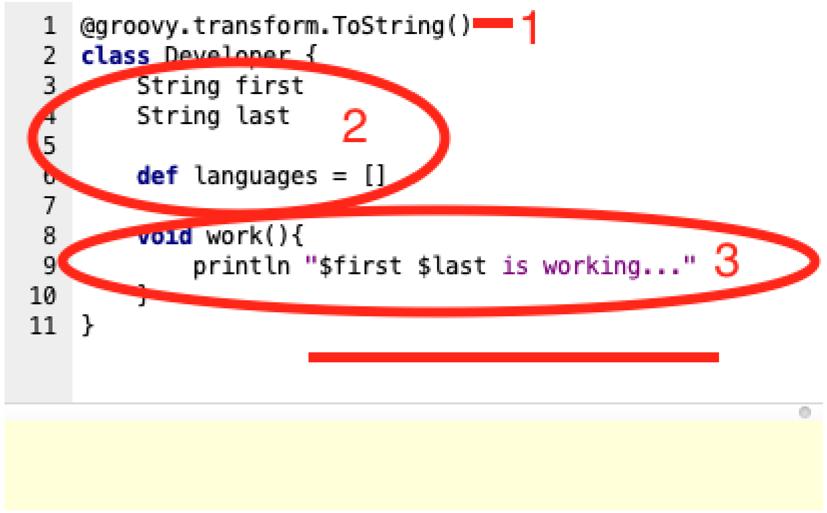
Groovy Language Features



identitrain

Groovy Features – Classes

- Define a Class Classes are like a blueprint for how we want our data stored and the things we can do with our data.
- This is an annotation, this one will transform the class to include a default toString method.
- These are the class properties. They can hold state for an object once it is created. This is a single method. Classes can have as many methods as needed. These usually perform some useful logic based on, or using the properties of the class.



```
@groovy.transform.ToString()
class Developer {
    String first
    String last

    def languages = []

    void work(){
        println "$first $last is working..."
    }
}
```

Groovy Features – Classes Continued

- Use a Class Instances are their own discrete collections of properties and methods. Setting a value in one instance does not impact other instances.
- This creates a new instance of our class.
- Setting properties on our new instance.
- Calling our methods for our Developer instance.

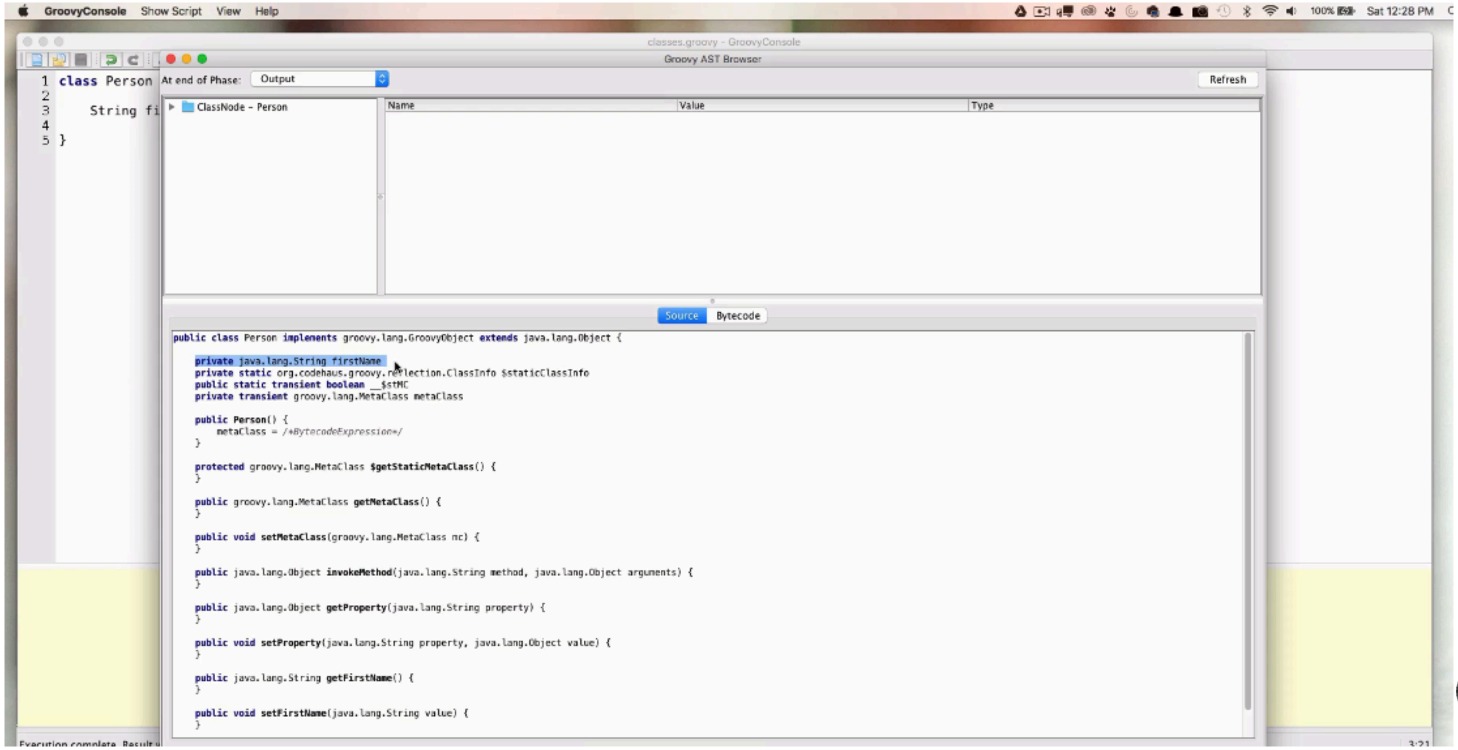
Checkpoint

What holds the state of an object?

- Properties

Groovy OOP – Access Modifiers

- Apache Groovy for Developers
- An Overview



```
class Person

    String fi
}
```

```
public class Person implements groovy.lang.GroovyObject extends java.lang.Object {

    private java.lang.String firstName
    private static org.codehaus.groovy.reflection.ClassInfo $staticClassInfo
    public static transient boolean __$stMC
    private transient groovy.lang.MetaClass metaClass

    public Person() {
        metaClass = /*BytecodeExpression*/
    }

    protected groovy.lang.MetaClass $getStaticMetaClass() {
    }

    public groovy.lang.MetaClass getMetaClass() {
    }

    public void setMetaClass(groovy.lang.MetaClass mc) {
    }

    public java.lang.Object invokeMethod(java.lang.String method, java.lang.Object arguments) {
    }

    public java.lang.Object getProperty(java.lang.String property) {
    }

    public void setProperty(java.lang.String property, java.lang.Object value) {
    }

    public java.lang.String getFirstName() {
    }

    public void setFirstName(java.lang.String value) {
    }
```

Groovy OOP – Access Modifiers

- Fields and variables

Groovy OOP – Constructors

Groovy OOP – Methods

- Methods in Groovy

Groovy OOP – Packages

- Organizing Classes Into Packages

Checkpoint

By default a Class will have an access modifier of ____?

- public

Groovy OOP – Inheritance

- Introduction
- Add a diagram of a class, interface, abstract class, and trait working together here.

Groovy OOP – Inheritance

- Introduction

Groovy OOP – Inheritance

- Overview

Groovy OOP - Inheritance

Interfaces define a contract that the user of the class can expect from the implementation of the class

- They have abstract method signatures
- Classes declare they implement them using the ‘implements’ keyword in the class
- Classes can implement multiple interfaces The class that implements the interface must have a method matching the interface signature with a body
- Interfaces

Groovy OOP - Inheritance

Abstract Classes are classes that are partially implemented. This means that they have some methods with bodies and other abstract method signatures.

- Classes extend abstract classes using the ‘extends’ keyword
- Classes can only extend one abstract class
- Abstract classes can extend other abstract classes
- Abstract Classes

Groovy OOP – Inheritance

- Traits
- Traits are interfaces that can hold an implementation of a method.
- They can declare abstract methods
- They can have method bodies
- They can declare properties that can be accessed by classes that inherit from it

Groovy OOP – Traits Continued

- Demonstration

Groovy OOP – Traits Continued

- Demonstration

Groovy OOP – Traits Continued

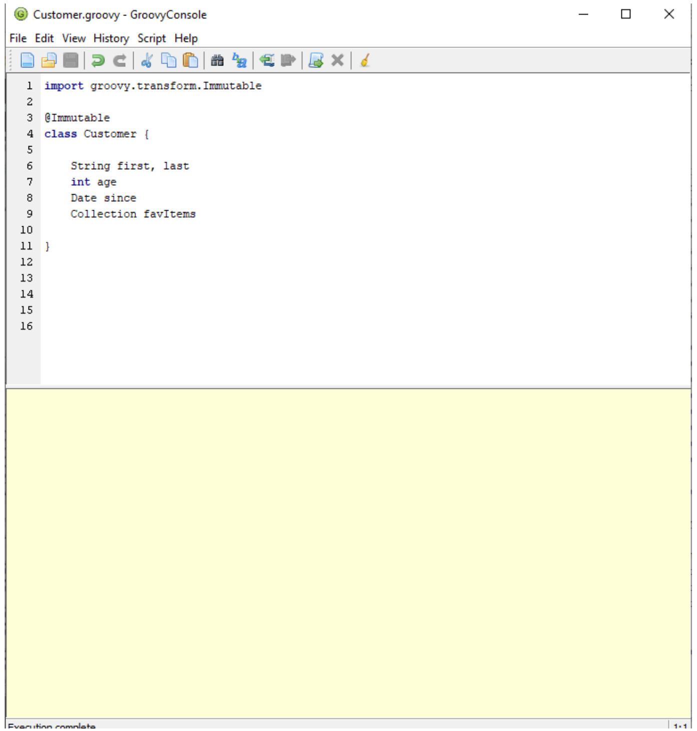
- Demonstration

Groovy Features – Scripts Implementation

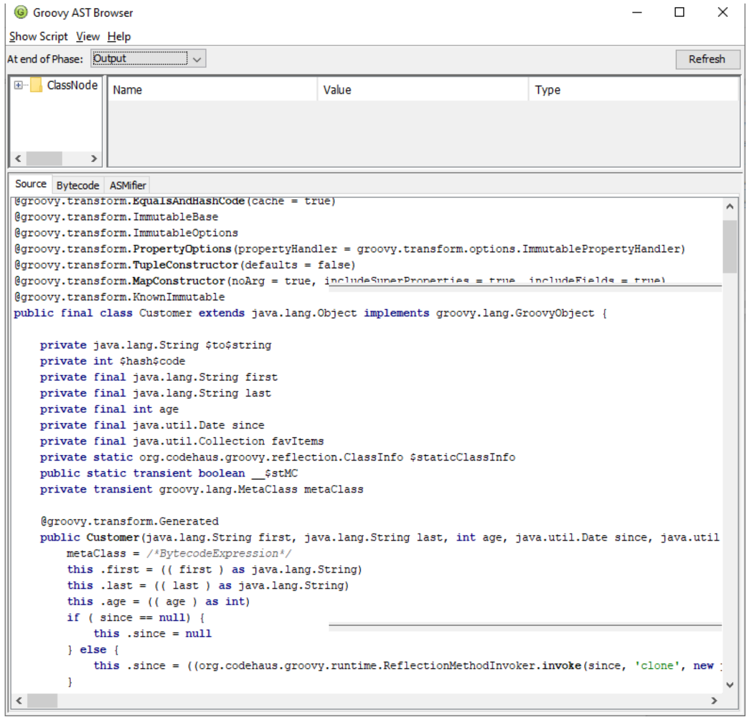
- Scripts and the AST Browser

Groovy Features – Annotations and AST

- An Immutable Class


Customer.groovy - GroovyConsole
File Edit View History Script Help
Execution complete
1:1

```
import groovy.transform.Immutable

@Immutable
class Customer {

    String first, last
    int age
    Date since
    Collection favItems

}
```



```
@groovy.transform.EqualsAndHashCode(cache = true)
@groovy.transform.ImmutableBase
@groovy.transform.ImmutableOptions
@groovy.transform.PropertyOptions(propertyHandler = groovy.transform.options.ImmutablePropertyHandler)
@groovy.transform.TupleConstructor(defaults = false)
@groovy.transform.MapConstructor(noArg = true, includeSuperProperties = true, includeFields = true)
@groovy.transform.KnownImmutable
public final class Customer extends java.lang.Object implements groovy.lang.GroovyObject {

    private java.lang.String $to$string
    private int $hash$code
    private final java.lang.String first
    private final java.lang.String last
    private final int age
    private final java.util.Date since
    private final java.util.Collection favItems
    private static org.codehaus.groovy.reflection.ClassInfo $staticClassInfo
    public static transient boolean __$stMC
    private transient groovy.lang.MetaClass metaClass

    @groovy.transform.Generated
    public Customer(java.lang.String first, java.lang.String last, int age, java.util.Date since, java.util
        metaClass = /*BytecodeExpression*/
        this .first = (( first ) as java.lang.String)
        this .last = (( last ) as java.lang.String)
        this .age = (( age ) as int)
        if ( since == null) {
            this .since = null
        } else {
            this .since = ((org.codehaus.groovy.runtime.ReflectionMethodInvoker.invoke(since, 'clone', new
        }
```

Groovy Features – Annotations and AST Continued

- Attempt to Change an Immutable Property

Exercise

identitrain

Exercise

identitrain

Questions?

Groovy Features

With an understanding of how OOP works in Groovy, we can begin to look at some more advanced features of Groovy that rely on the underpinnings of Object Oriented Programming.

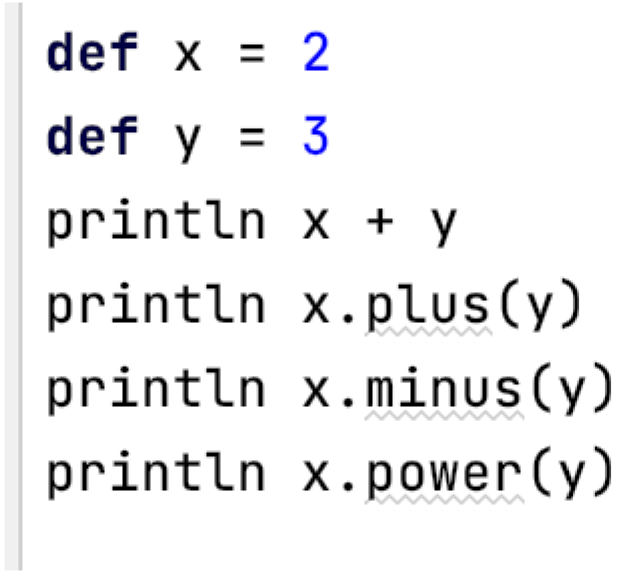
Groovy Features – Operator Overloading

Since everything in Groovy is an object, the operators themselves resolve to function calls on the objects.

- This means that you can implement your own operations for the operators that we’ve covered!
- This is helpful when you have two objects that seem like adding them would make sense.
- Example: totalBalance = CheckingAccount + SavingsAccount

Groovy Features – Operator Overloading Continued

- Operations on Built-in Groovy Classes
- You can use the operator or the method to perform operations with the built-in types in Groovy.



```
def x = 2
def y = 3
println x + y
println x.plus(y)
println x.minus(y)
println x.power(y)
```

Groovy Features – Operator Overloading Continued

- Adding Operator Overloads to Our Own Classes Operator Overloading can help us make logical choices for how object should be treated with operators.

Groovy Features – Recap and Casting

- Default types for Numbers, Explicit Conversion
- Table of resulting data type for math operations: http://groovy-lang.org/syntax.html#_math_operations

Groovy Features - Coercion

- Implicit Conversion

Groovy Features – Big Decimal

- Coercion rules 2

Groovy Features – More Cases

- Double Arithmetic and Integer Division

Groovy Features – GDK Methods

- String and Number Conversion

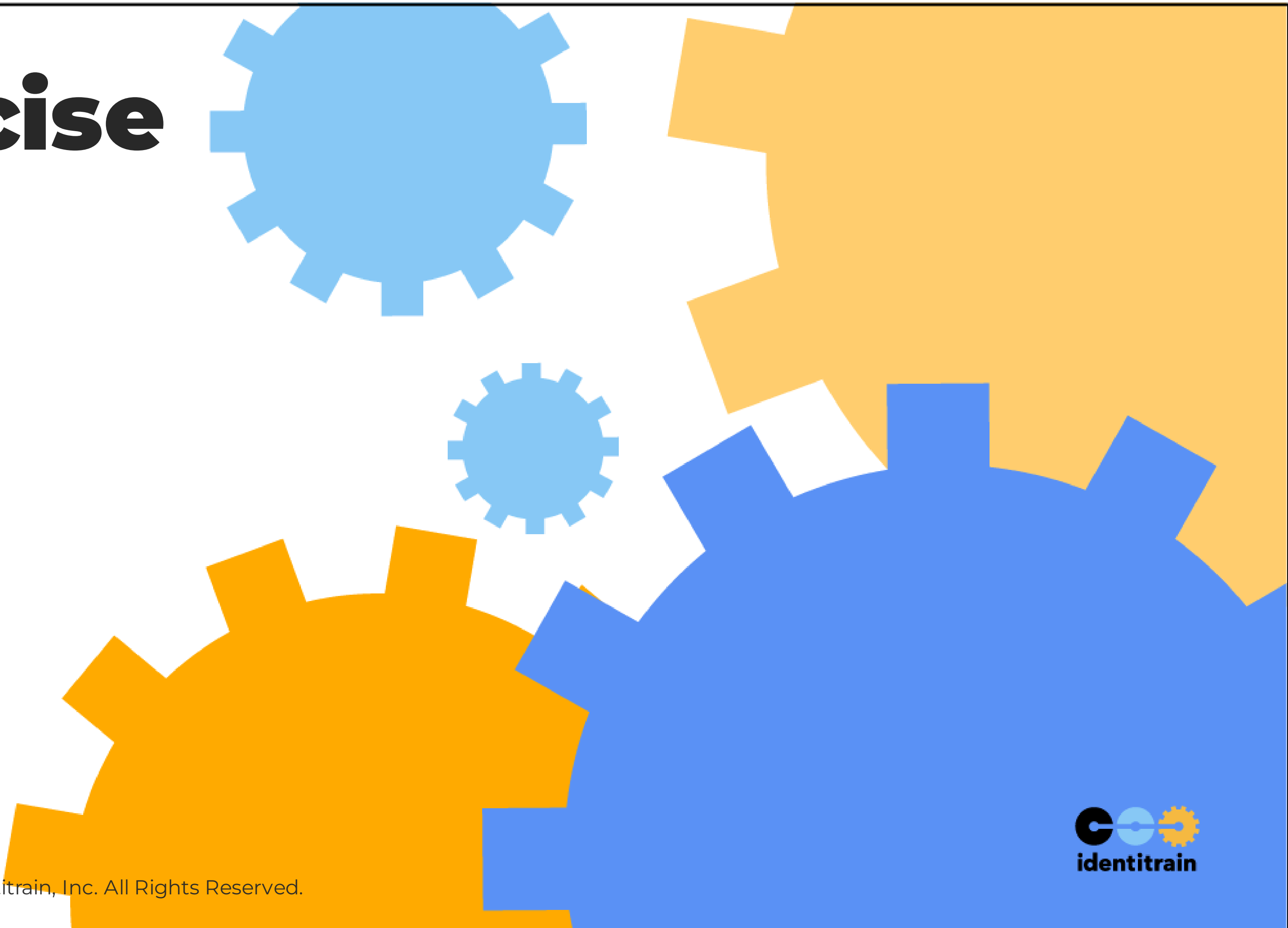
Groovy Features – GDK Methods Continued

- Iteration Methods

Exercise

identitrain

Exercise



identitrain

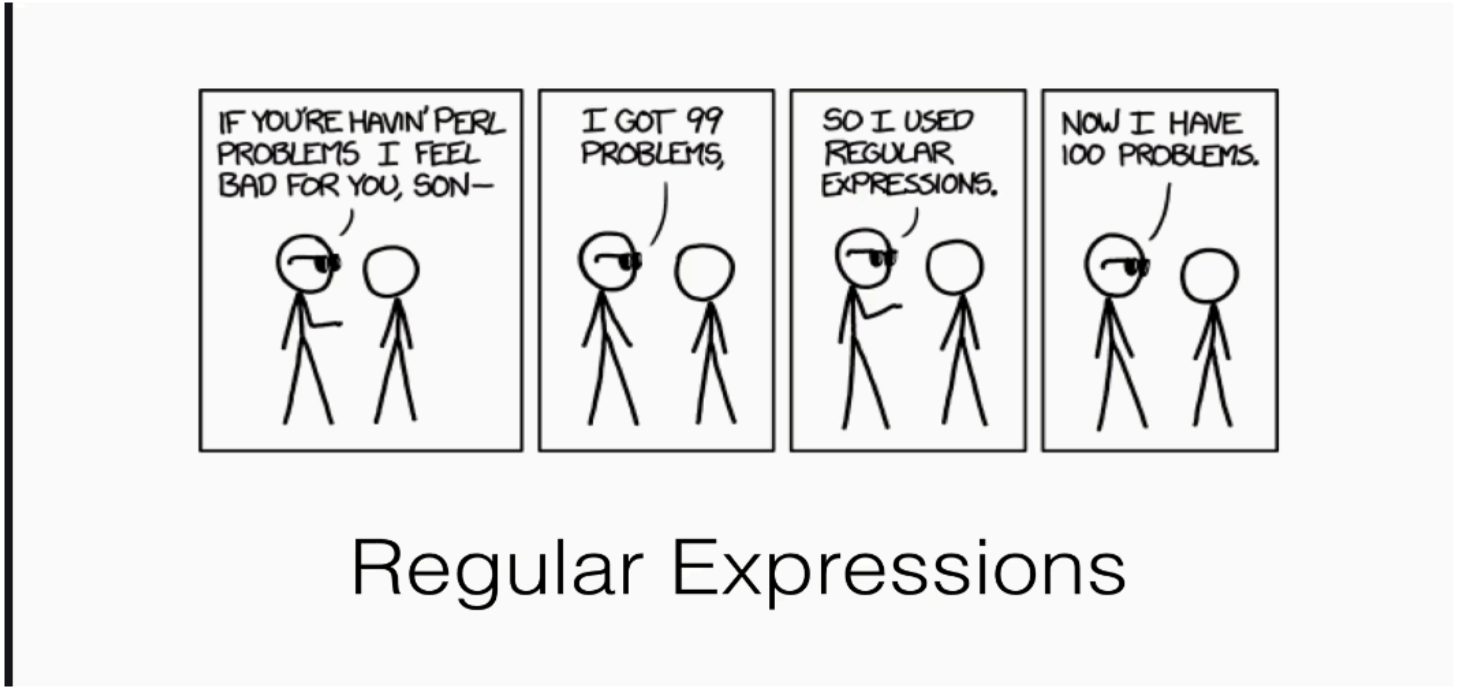
Questions?

identitrain

Regular Expressions & Collections

Groovy Regex – Regular Expressions

- Powerful But Not Simple


IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—
I GOT 99 PROBLEMS,
SO I USED REGULAR EXPRESSIONS.
NOW I HAVE 100 PROBLEMS.
Regular Expressions

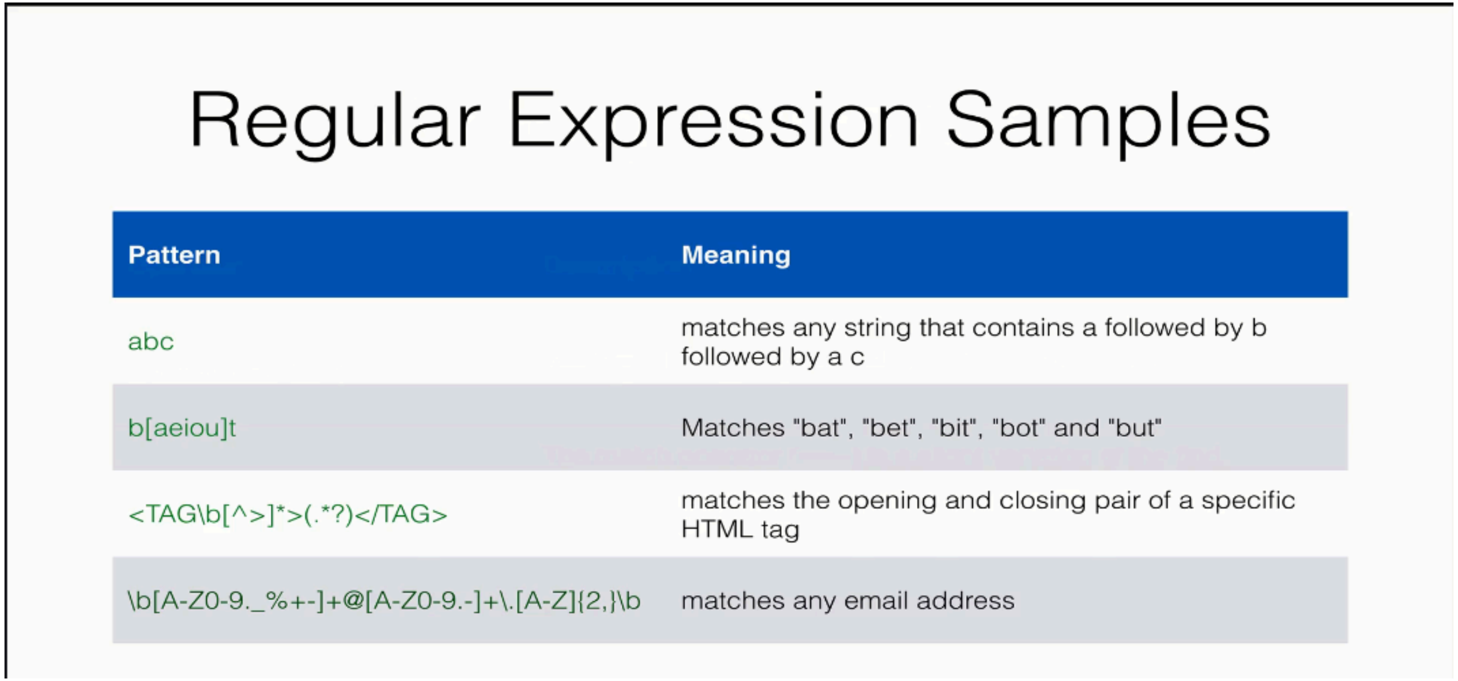
Groovy Regex – Regular Expressions

- Online Regex Resources https://www.regexone.com
- Interactive tutorial thats walks through the basics. https://regex101.com
-
- Interactive regex tester that also explains how its matching the groups etc.

identitrain

Groovy Regex – Regular Expressions Continued

- Search Patterns



Regular Expression Samples

Pattern	Meaning
abc	matches any string that contains a followed by b followed by a c
b[aeiou]t	Matches "bat", "bet", "bit", "bot" and "but"
<TAG\b[^>]*>(.*?)</TAG>	matches the opening and closing pair of a specific HTML tag
\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b	matches any email address

Groovy Regex – Regular Expressions Continued

- Groovy Regular Expression Operators
- Pattern Operator: ~ Returns an instance of a Pattern.
- Find Operator: =~ Returns an instance of a Matcher.
- Match Operator: ==~ Returns a boolean if the string matches.
- Pattern Documentation: https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html
- Matcher Documentation: https://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html

Groovy Regex – Regular Expressions Continued

- Demonstration
- Getting a pattern is good if you'll need to check the same pattern for multiple matchers. Using the matcher operator is a shortcut to print each of the matches to a regex (something like parsing)
- Using the matches operator is good to just verify data (such as input)

Groovy Collections – Lists

- Working with Lists
- There are several operators and methods that can be used to navigate lists.
- To initialize a list you can use a range operator … as in 1…4 - to represent [1,2,3,4]
- Subscript operator [1] - refers to the element at index 1 of the list.
- Safe index ?[2] - null safe element of an array
- Spread * - equivalent to each() and supports *. to reference a property of an object
- Membership Operator in - 1 in [0,1,2] - returns boolean if the element is in the list.
- Denormalizing lists can be done with flatten()

Groovy Collections – Ranges

- The New Way
- Range Operators work with anything that is Comparable and has a next() and previous() method.

Groovy Collections – Ranges Continued

- Ranges of Other Things

Groovy Collections – Lists

- Introduction

Groovy Collections – Lists Continued

- Manipulating List Contents

Groovy Collections – Lists Continued

- More on Manipulating List Contents

Groovy Collections – Lists Continued

- Other Methods

Groovy Collections – Maps

- A List of Name Value Pairs

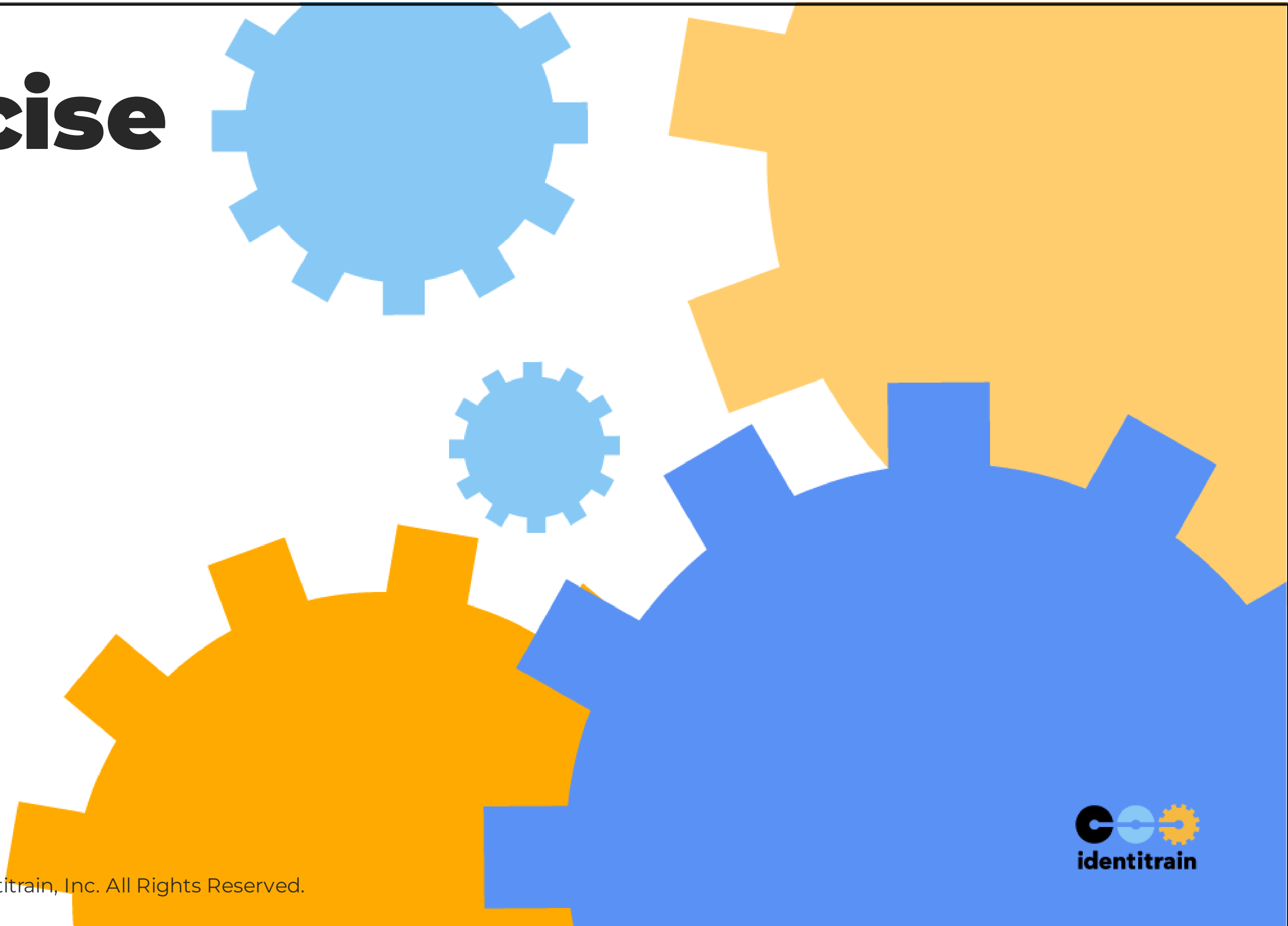
Groovy Collections – Maps Continued

- More About Maps

Exercise

identitrain

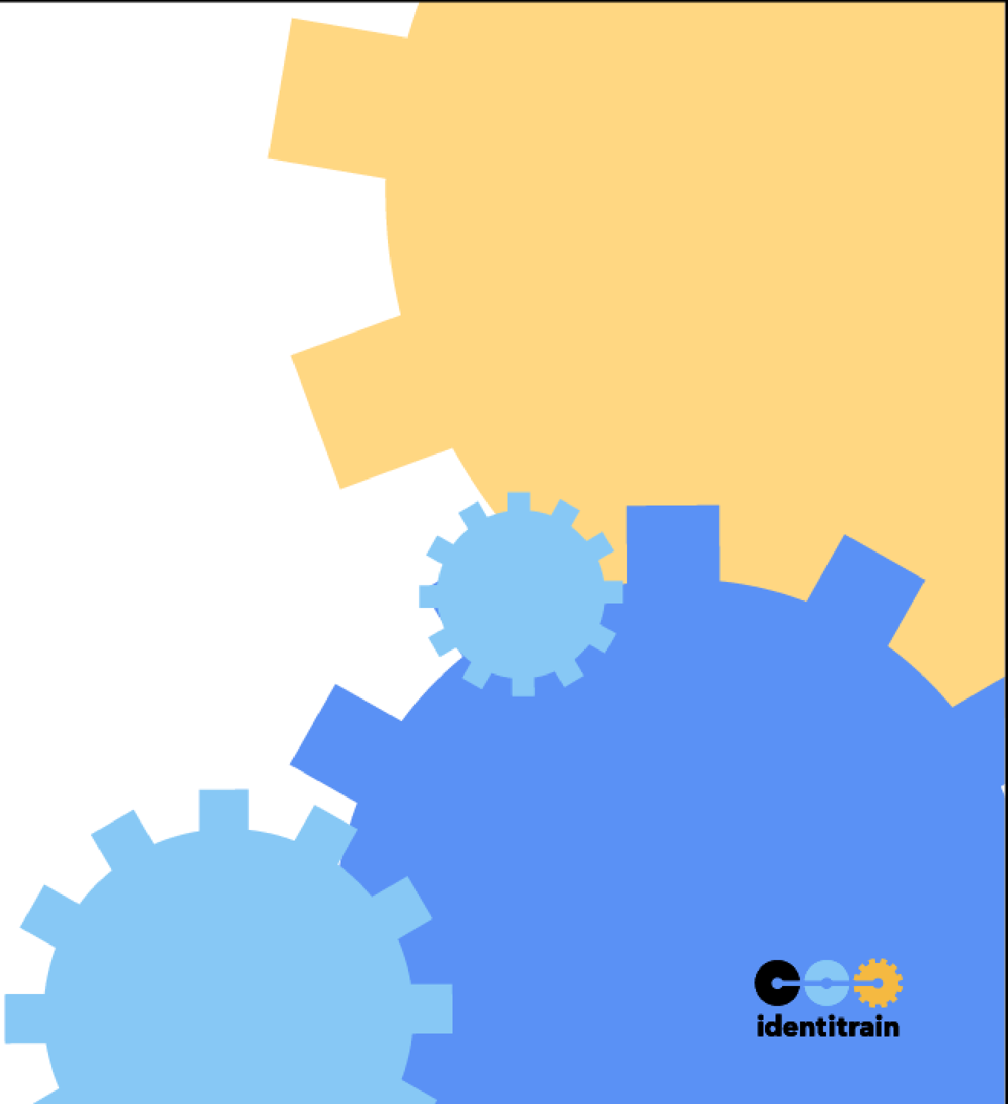
Exercise



identitrain

Questions?

Closures



Groovy Closures – What are Closures?

- What are Closures used for?
- Iterators
- Callbacks
- Higher-Order functions
- Specialized Control Structure
- Builders
- Resource Allocation
- Threads
- DSLs
- Fluent Interfaces
- Documentation Link: http://groovy-lang.org/closures.html

identitrain

Groovy Closures – Syntax

- Syntax
- Groovy Closures have an implicit parameter of ‘it’ if one is not defined.
- If you define a parameter, the ‘it’ parameter is no longer referenceable.
- 3 other important variables are available to all Closures
- this
- owner
- delegate
- Closure syntax
- { parameters -> code block }
- Example:
- def setValues = {
- (name, value) -> {
- delegate.name = name
- delegate.value = value
- }
- }
- setValues(‘State’, ‘Florida’)

Groovy Closures – Variables

- Resolution Strategies
- Closures can reference variables from this, owner, and delegate
- Groovy allows developers to specify the order that they are evaluated
- Groovy allows developers to restrict to a single entity to resolve from
- See :
- Closures.OWNER_FIRST
- Closures.DELEGATE_FIRST
- Closures.ONLY_OWNER
- Closures.DELEGATE_ONLY

Groovy Closures – Useful Methods

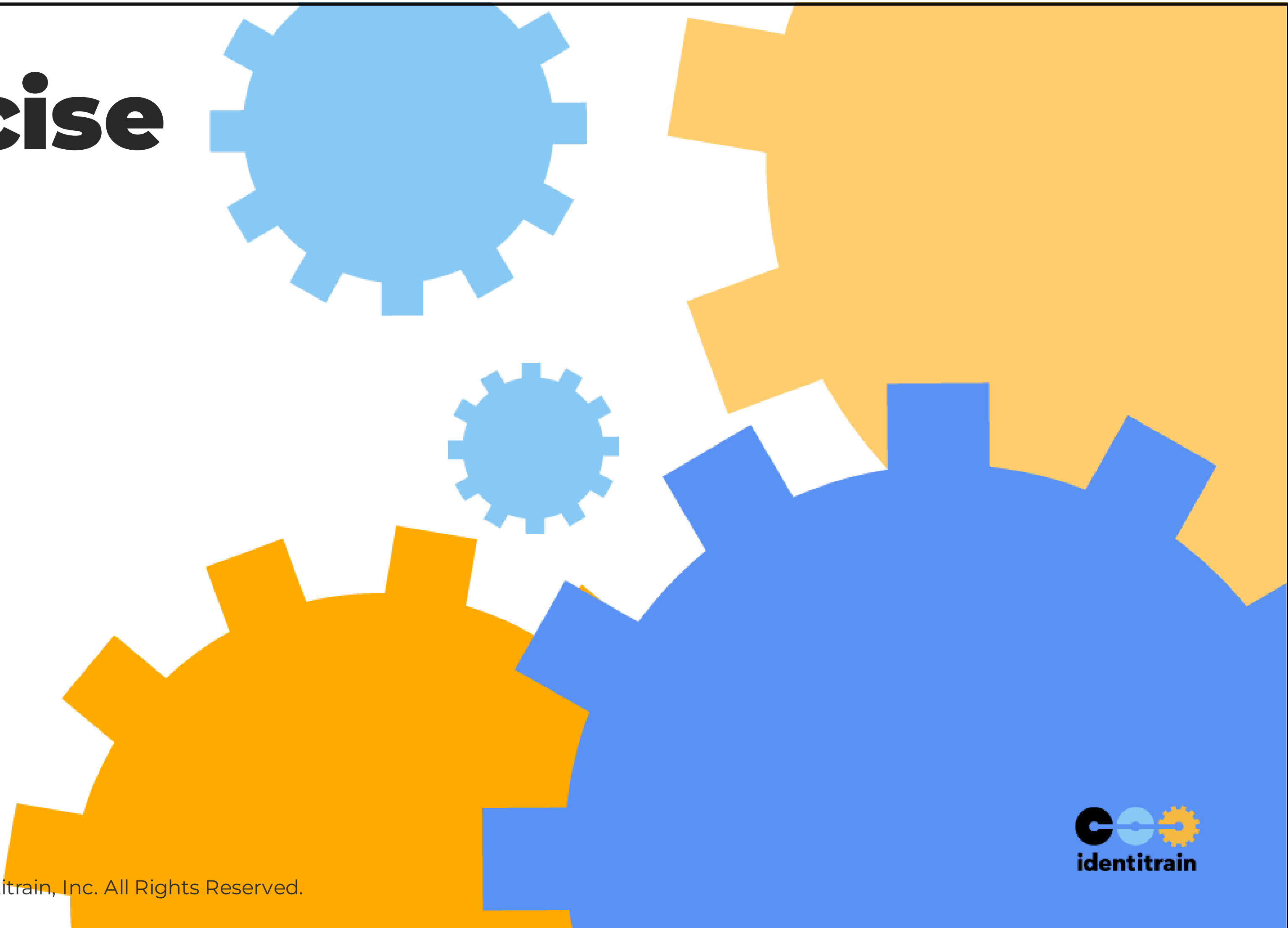
- Closure API
- Groovy Closures support currying
- curry
- ncurry
- rcurry
- Groovy Closures can use trampoline() for recursion Groovy Closures have a special compose operator to define a new closure as a composition of 2 or more other closures
- greetPatrick = greet.curry("patrick")
- greetOnSaturday = greet.rcurry("saturday")
- greetOnSaturday = greet.ncurry(1, "saturday")
- greet.trampoline()
- greetPatrickOnSaturday = greetOnSaturday() << greetPatrick()

Groovy Closures – GStrings

- GStrings are Special
- You can use Groovy closures in GStrings (we’ve seen these) Strings are special types and the value is always static, when you assign a string anew value it actually points to a new location.
- When you assign the string with the closure, the closure is evaluated at that time. Avoiding it means to declare a closure to return a newly declared string - thus forcing a new read of the value.

Exercise

Exercise



identitrain

Questions?

Builders and Slurpers in Groovy

Builders

- Familiar Syntax Groovy uses a builder syntax to help simplify setting up some of the complex data structures that developers encounter.
- It ‘flows’ the same for different types of structures Removes a lot of the boilerplate that would be required to create them using just the classes themselves
- They are implemented using some of the core concepts we covered in closures
- Documentation for Builders: https://docs.groovy-lang.org/latest/html/documentation/core-domain-specific-languages.html#_builders

Builders

- ObjectGraphBuilder
- ObjectGraphBuilders help with objects that are composed of many different properties

Builders

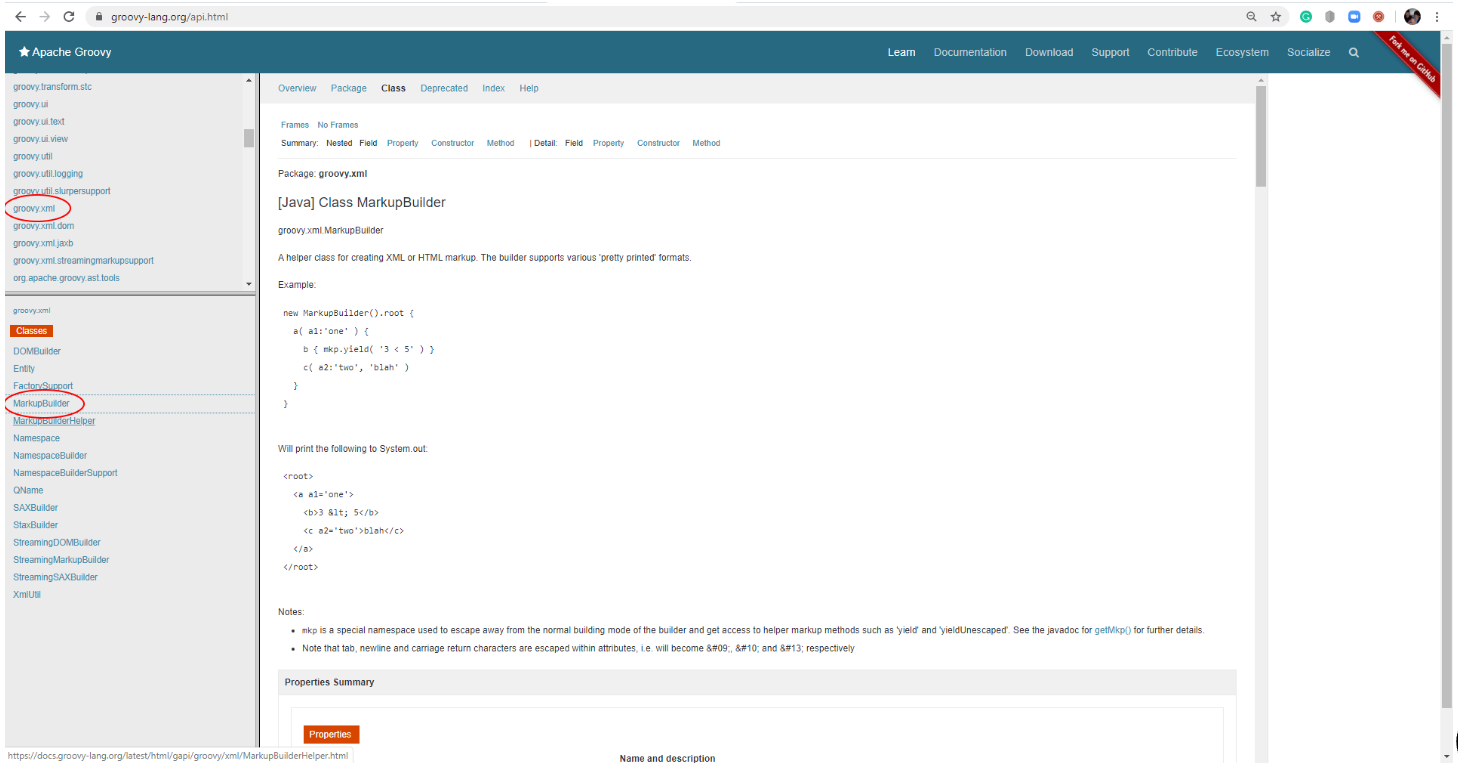
- HTML Builder
- Markup Builders help by removing the boilerplate around markup languages.
- Reduces the lines of code needed
- Clearer code that will help with readability and maintainability

Slurpers

- JSON and XML Slurpers
- Flexible parsing to meet memory/cpu needs
- Implementation is backed by reliable Java class libraries

Groovy Builders – MarkupBuilder XML

- Documentation



Package: **groovy.xml**

[Java] Class MarkupBuilder

groovy.xml.MarkupBuilder

A helper class for creating XML or HTML markup. The builder supports various 'pretty printed' formats.

Example:

```
new MarkupBuilder().root {
  a( a1:'one' ) {
    b { mkp.yield( '3 < 5' ) }
    c( a2:'two', 'blah' )
  }
}
```

Will print the following to System.out:

```
<root>
  <a a1='one'>
    <b>3 &lt; 5</b>
    <c a2='two'>blah</c>
  </a>
</root>
```

Notes:

- `mkp` is a special namespace used to escape away from the normal building mode of the builder and get access to helper markup methods such as 'yield' and 'yieldUnescaped'. See the javadoc for getMkp() for further details.
- Note that tab, newline and carriage return characters are escaped within attributes, i.e. will become &#09;, &#10; and &#13; respectively

Properties Summary

Properties

Name and description

Groovy Builders – MarkupBuilder XML Continued

- Demonstration

Groovy Builders – Builder Documentation

- Github Tests

Groovy Builders – MarkupBuilder HTML

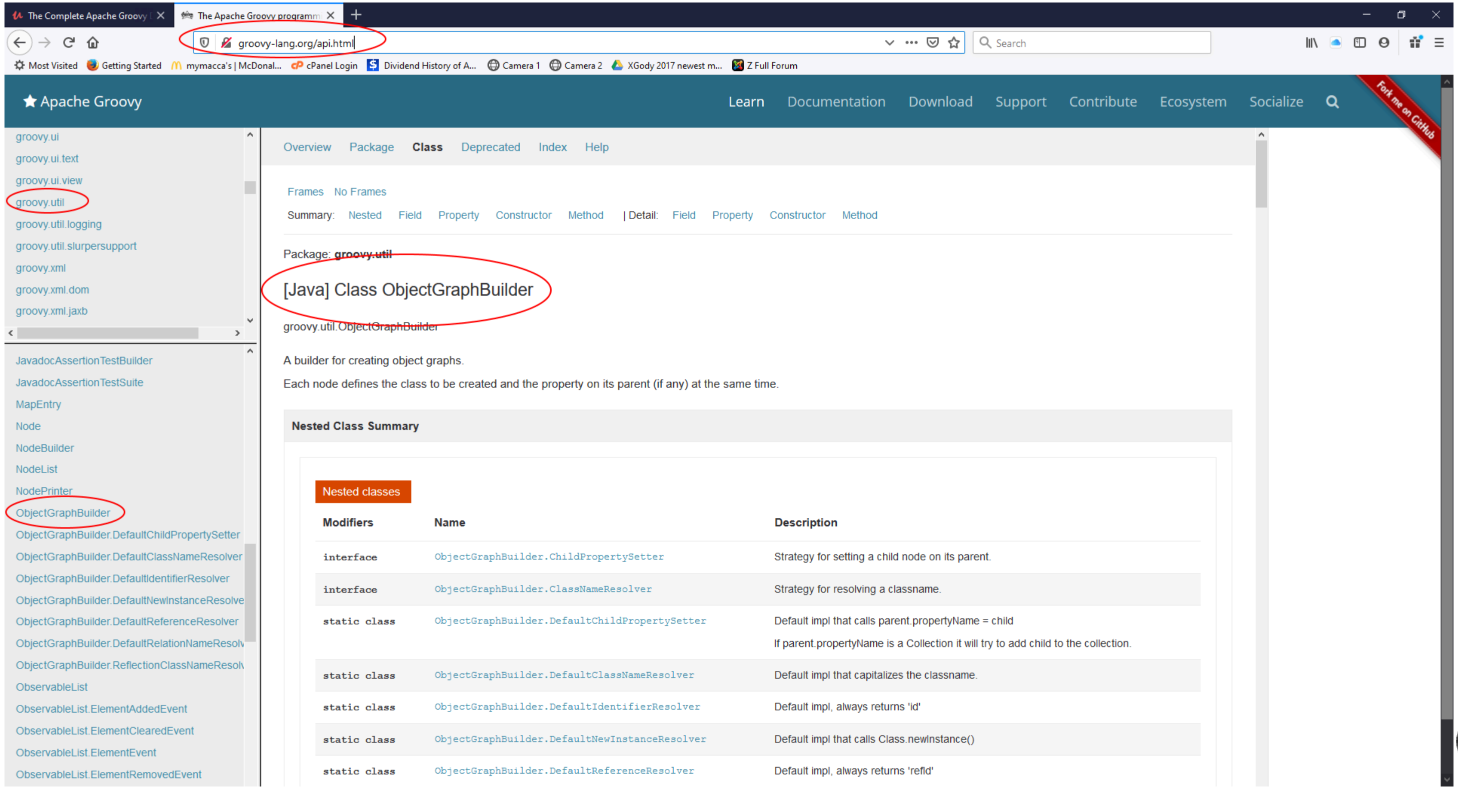
- Generating HTML

Groovy Builders – The JsonBuilder

- Introduction

Groovy Builders – Object Graph Builder

- Documentation



The Complete Apache Groovy
The Apache Groovy programm
groovy-lang.org/api.html
Search
Most Visited
Getting Started
mymacca's | McDonal...
cPanel Login
Dividend History of A...
Camera 1
Camera 2
XGody 2017 newest m...
Z Full Forum
Apache Groovy
Learn
Documentation
Download
Support
Contribute
Ecosystem
Socialize
Fork me on GitHub
groovy.ui
groovy.ui.text
groovy.ui.view
groovy.util
groovy.util.logging
groovy.util.slurpersupport
groovy.xml
groovy.xml.dom
groovy.xml.jaxb
JavadocAssertionTestBuilder
JavadocAssertionTestSuite
MapEntry
Node
NodeBuilder
NodeList
NodePrinter
ObjectGraphBuilder
ObjectGraphBuilder.DefaultChildPropertySetter
ObjectGraphBuilder.DefaultClassNameResolver
ObjectGraphBuilder.DefaultIdentifierResolver
ObjectGraphBuilder.DefaultNewInstanceResolve
ObjectGraphBuilder.DefaultReferenceResolver
ObjectGraphBuilder.DefaultRelationNameResolv
ObjectGraphBuilder.ReflectionClassNameResolv
ObservableList
ObservableList.ElementAddedEvent
ObservableList.ElementClearedEvent
ObservableList.ElementEvent
ObservableList.ElementRemovedEvent
Overview
Package
Class
Deprecated
Index
Help
Frames
No Frames
Summary: Nested Field Property Constructor Method | Detail: Field Property Constructor Method

Package: **groovy.util**

[Java] Class ObjectGraphBuilder

groovy.util.ObjectGraphBuilder

A builder for creating object graphs.

Each node defines the class to be created and the property on its parent (if any) at the same time.

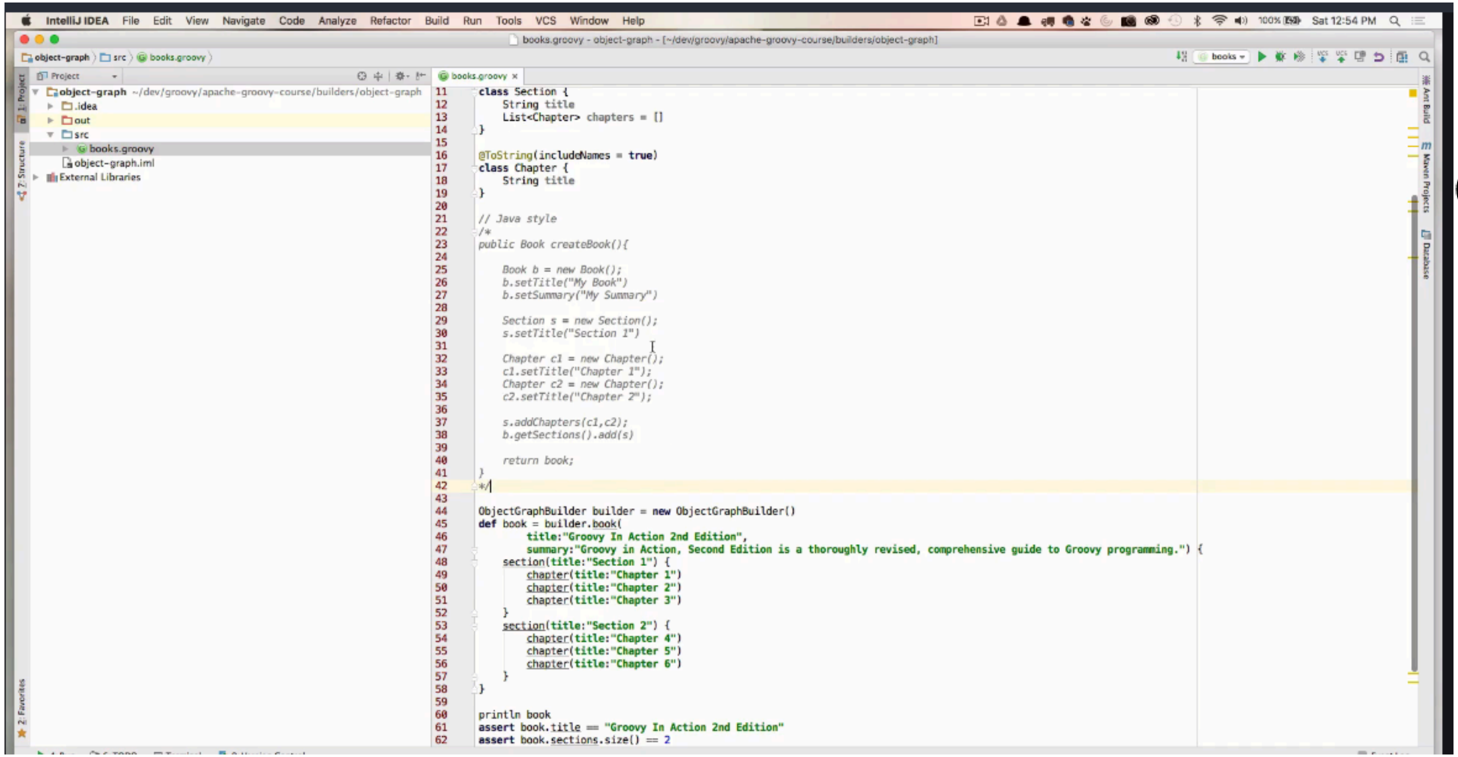
Nested Class Summary

Nested classes

Modifiers	Name	Description
`interface`	`ObjectGraphBuilder.ChildPropertySetter`	Strategy for setting a child node on its parent.
`interface`	`ObjectGraphBuilder.ClassNameResolver`	Strategy for resolving a classname.
`static class`	`ObjectGraphBuilder.DefaultChildPropertySetter`	Default impl that calls parent.propertyName = child If parent.propertyName is a Collection it will try to add child to the collection.
`static class`	`ObjectGraphBuilder.DefaultClassNameResolver`	Default impl that capitalizes the classname.
`static class`	`ObjectGraphBuilder.DefaultIdentifierResolver`	Default impl, always returns 'id'
`static class`	`ObjectGraphBuilder.DefaultNewInstanceResolver`	Default impl that calls Class.newInstance()
`static class`	`ObjectGraphBuilder.DefaultReferenceResolver`	Default impl, always returns 'refId'

Groovy Builders – Object Graph Builder Continued

- Demonstration

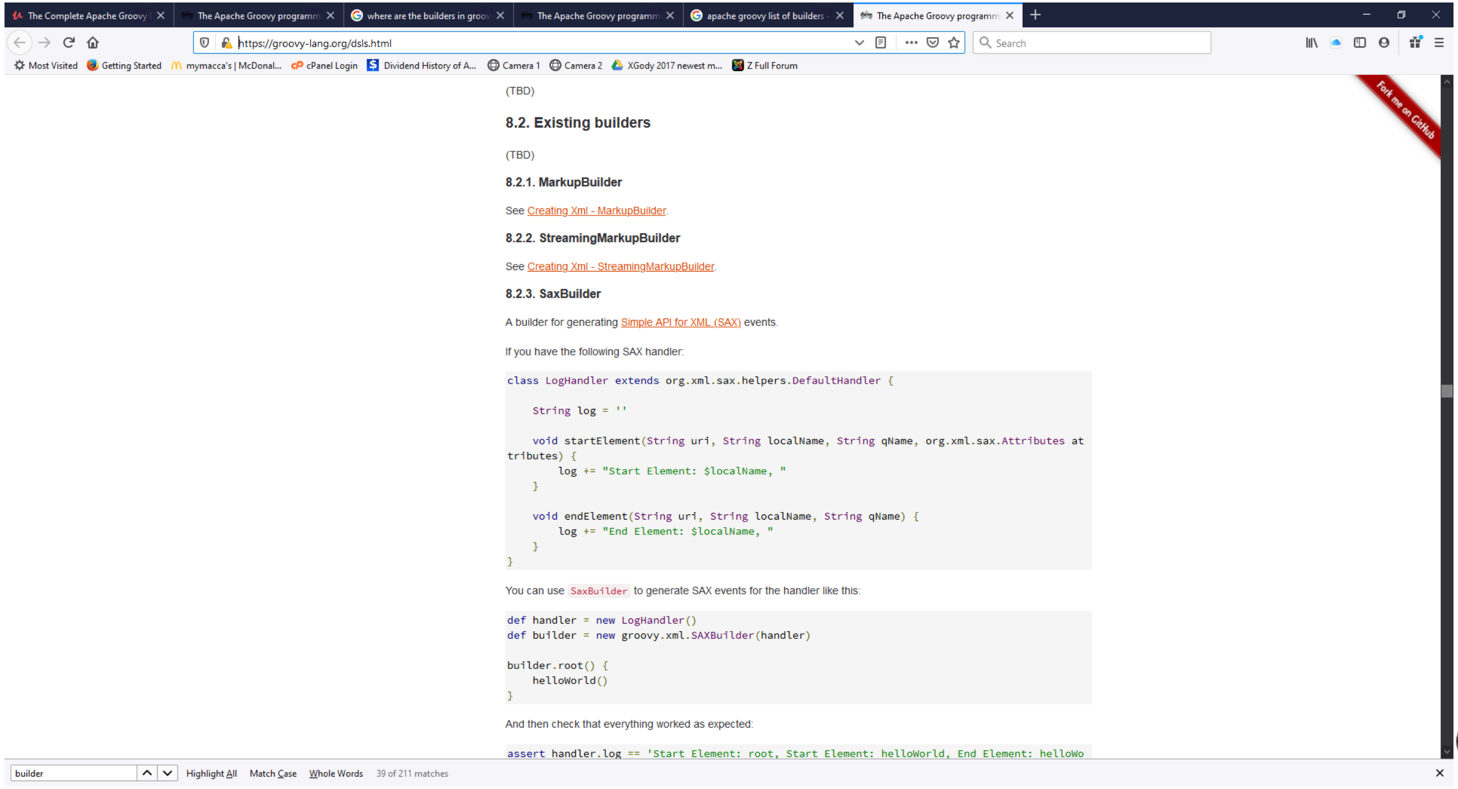


books.groovy - object-graph - [~/dev/groovy/apache-groovy-course/builders/object-graph]

```groovy
class Section {
    String title
    List<Chapter> chapters = []
}

@ToString(includeNames = true)
class Chapter {
    String title
}

// Java style
/*
public Book createBook(){

    Book b = new Book();
    b.setTitle("My Book")
    b.setSummary("My Summary")

    Section s = new Section();
    s.setTitle("Section 1")

    Chapter c1 = new Chapter();
    c1.setTitle("Chapter 1");
    Chapter c2 = new Chapter();
    c2.setTitle("Chapter 2");

    s.addChapters(c1,c2);
    b.getSections().add(s)

    return book;
}
*/

ObjectGraphBuilder builder = new ObjectGraphBuilder()
def book = builder.book(
        title:"Groovy In Action 2nd Edition",
        summary:"Groovy in Action, Second Edition is a thoroughly revised, comprehensive guide to Groovy programming.") {
    section(title:"Section 1") {
        chapter(title:"Chapter 1")
        chapter(title:"Chapter 2")
        chapter(title:"Chapter 3")
    }
    section(title:"Section 2") {
        chapter(title:"Chapter 4")
        chapter(title:"Chapter 5")
        chapter(title:"Chapter 6")
    }
}

println book
assert book.title == "Groovy In Action 2nd Edition"
assert book.sections.size() == 2
```

Groovy Builders – List of Builders

- Documentation



(TBD)

8.2. Existing builders

(TBD)

8.2.1. MarkupBuilder

See Creating Xml - MarkupBuilder.

8.2.2. StreamingMarkupBuilder

See Creating Xml - StreamingMarkupBuilder.

8.2.3. SaxBuilder

A builder for generating Simple API for XML (SAX) events.

If you have the following SAX handler:

```
class LogHandler extends org.xml.sax.helpers.DefaultHandler {

    String log = ''

    void startElement(String uri, String localName, String qName, org.xml.sax.Attributes at
tributes) {
        log += "Start Element: $localName, "
    }

    void endElement(String uri, String localName, String qName) {
        log += "End Element: $localName, "
    }
}
```

You can use `SaxBuilder` to generate SAX events for the handler like this:

```
def handler = new LogHandler()
def builder = new groovy.xml.SAXBuilder(handler)

builder.root() {
    helloWorld()
}
```

And then check that everything worked as expected:

```
assert handler.log == 'Start Element: root, Start Element: helloWorld, End Element: helloWo
```

Checkpoint

The MarkupBuilder is located in what package?

- Groovy.xml

Exercise

identitrain

Exercise

identitrain

Questions?

REST Services

Groovy Feature – Grapes

- Managing Dependencies in Groovy
- Groovy has a built-in dependency management system
- Artifact inclusion similar to Ivy, Maven
- Customizable artifact resolver configuration (if necessary)
- Implemented with Annotations
- Isolates the code that needs the dependency
- Allows scripts to be extremely portable
- Resolves dependencies on the machine running code
- Documentation: https://www.groovy-lang.org/grape.html
- @Grapes
- ([
-
- @Grab
- (group=
- 'org.codehaus.groovy.modules.http-builder'
- , module=
- 'http-builder'
- , version=

IntelliJ Configuration – Grapes

- Hints don't work with Grapes
- Configure your project to support Ivy Dependencies
- Create an ivy.xml file at the base of your project
- Resolve ivy dependencies Without doing this, the hints and autocomplete won't know what is available from the libraries you've included with Grapes.
- Note that this is not necessary if you are not using Grapes!
- Mapping a @Grab to Ivy
- grab:group -> ivy:org
- grab:module -> ivy:name
- grab:version -> ivy:rev
- Ivy File format:
- <?
- xml version
- ="1.0"
- encoding
- ="UTF-8"
- ?>
- <

Working with REST – Working with XML

- MarkupBuilder Revisited

Working with REST – Working with JSON

- Documentation

Working with REST – JSON Continued

- Demonstration

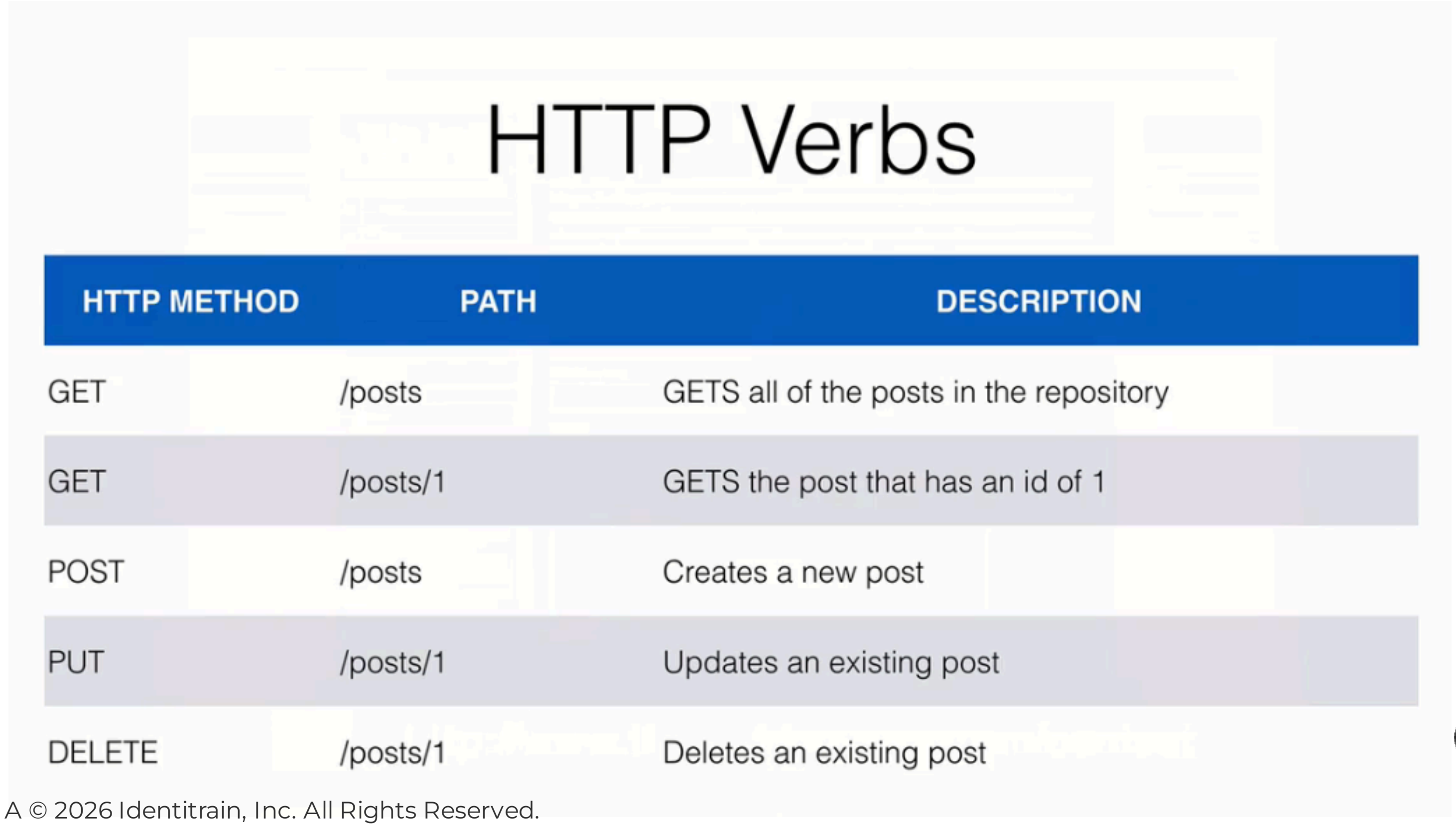
Checkpoint

What class in the Groovy API is used to parse xml into a document structure?

- XmlSlurper

Working with REST – HTTP Verbs

- Get and Post



HTTP Verbs

HTTP METHOD	PATH	DESCRIPTION
GET	/posts	GETS all of the posts in the repository
GET	/posts/1	GETS the post that has an id of 1
POST	/posts	Creates a new post
PUT	/posts/1	Updates an existing post
DELETE	/posts/1	Deletes an existing post

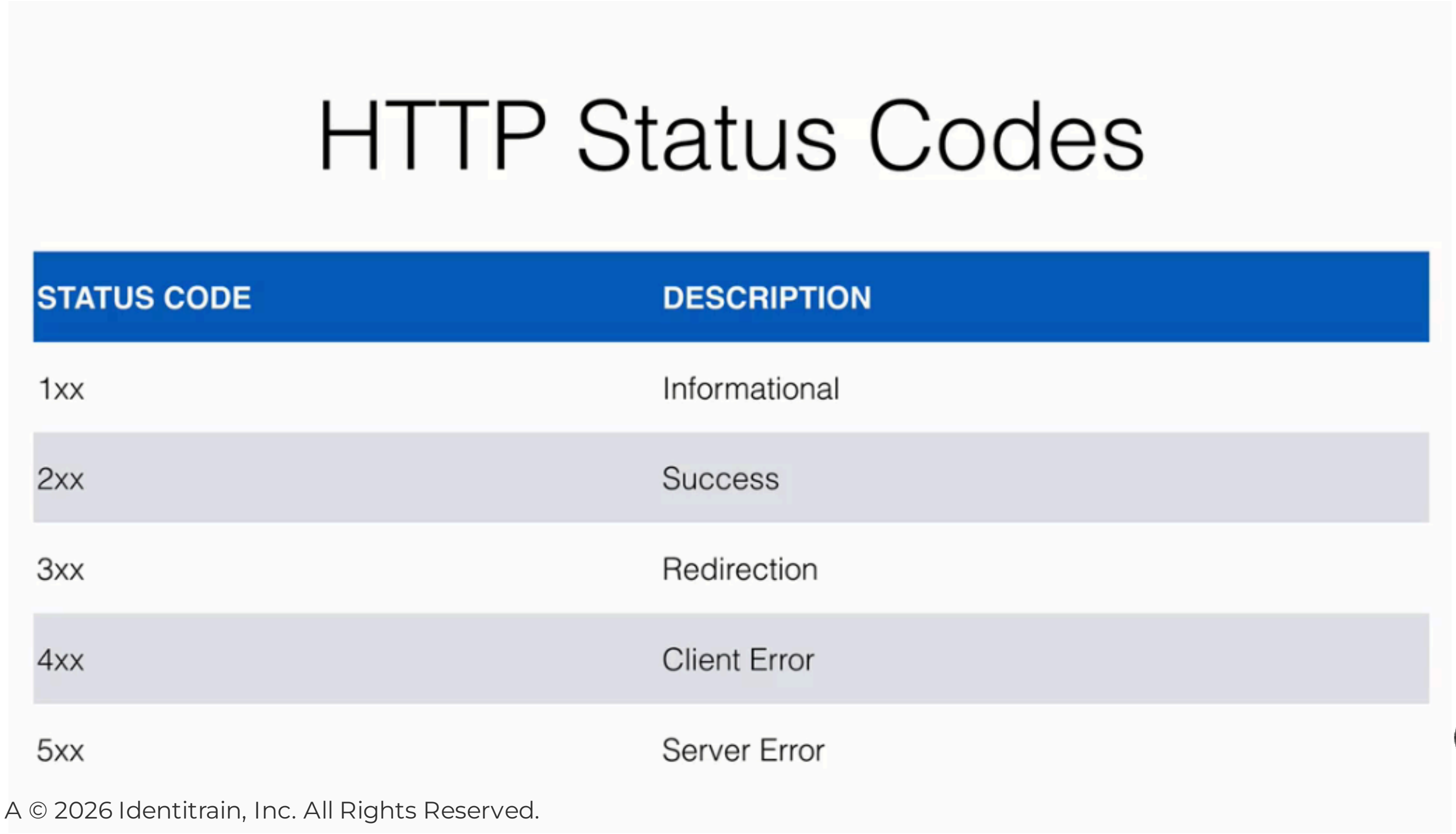
identitrain

Working with REST – HTTP Verb Recap

- Web Site or Server to Server

Working with REST – HTTP Status Codes

- Summary

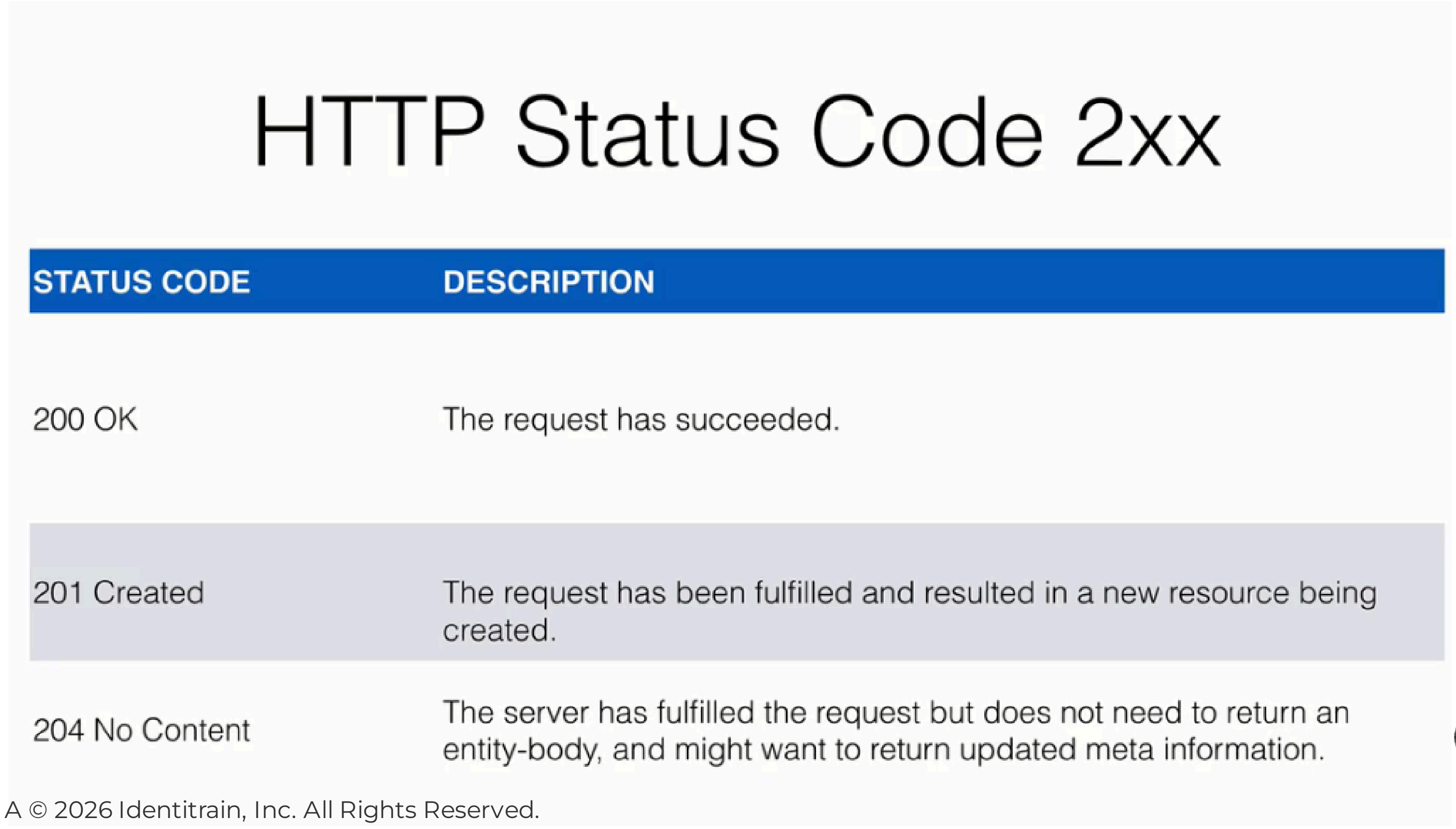


HTTP Status Codes

STATUS CODE	DESCRIPTION
1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error

Working with REST – HTTP Status Codes Continued

- 2xx Codes

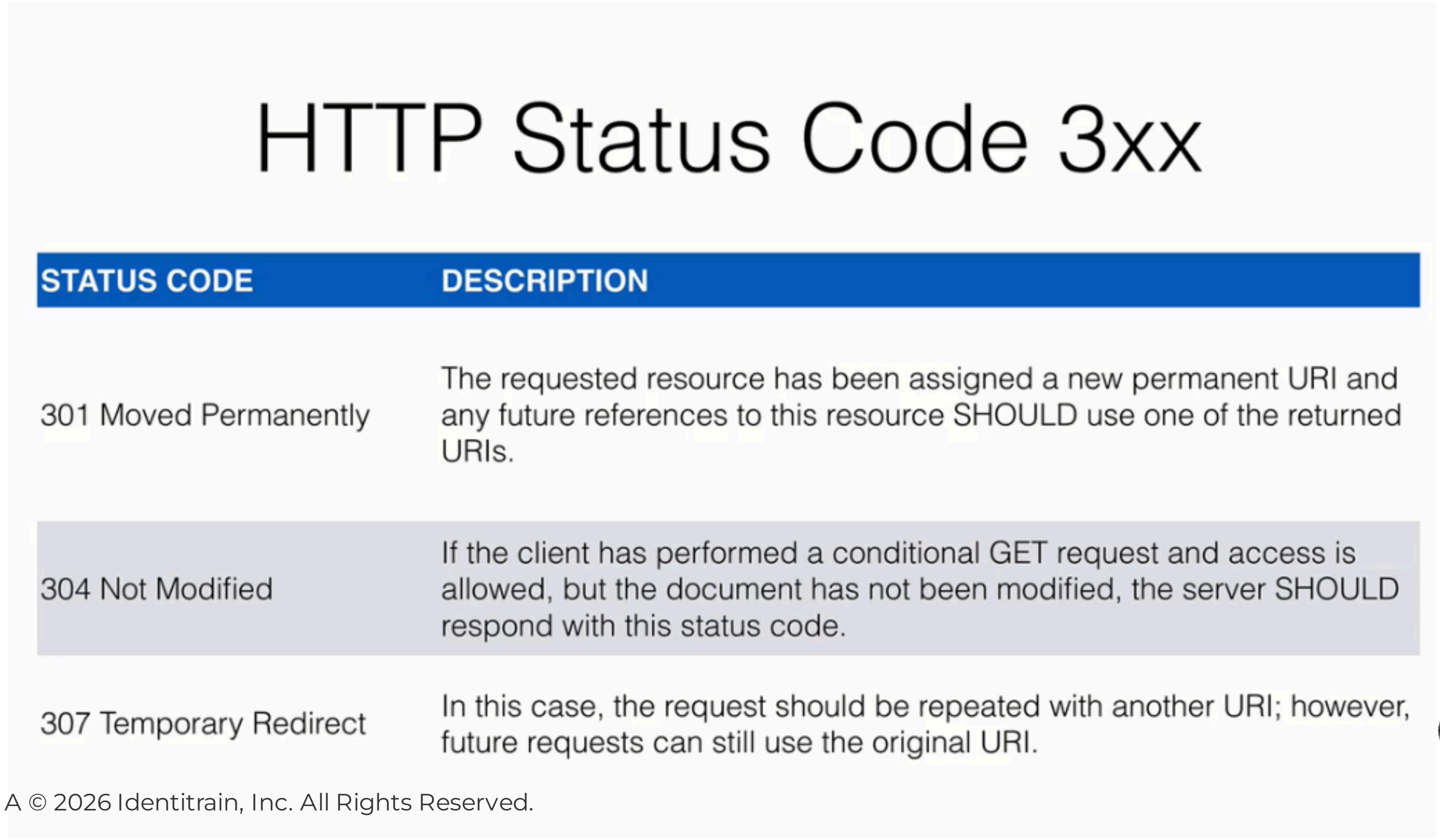


HTTP Status Code 2xx

STATUS CODE	DESCRIPTION
200 OK	The request has succeeded.
201 Created	The request has been fulfilled and resulted in a new resource being created.
204 No Content	The server has fulfilled the request but does not need to return an entity-body, and might want to return updated meta information.

Working with REST – HTTP Status Codes Continued

- 3xx Codes

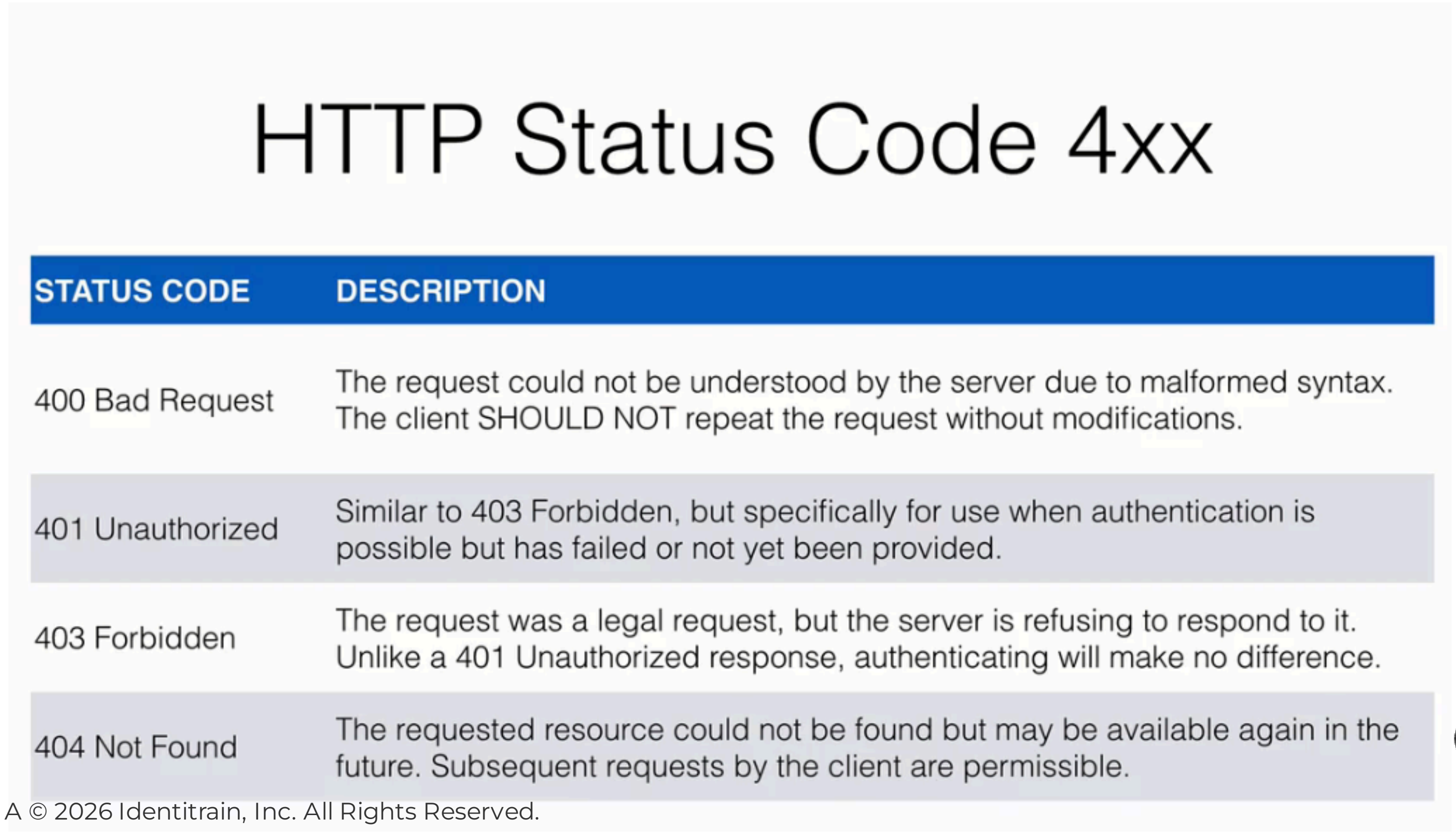


HTTP Status Code 3xx

STATUS CODE	DESCRIPTION
301 Moved Permanently	The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs.
304 Not Modified	If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code.
307 Temporary Redirect	In this case, the request should be repeated with another URI; however, future requests can still use the original URI.

Working with REST – HTTP Status Codes Continued

- 4xx Codes

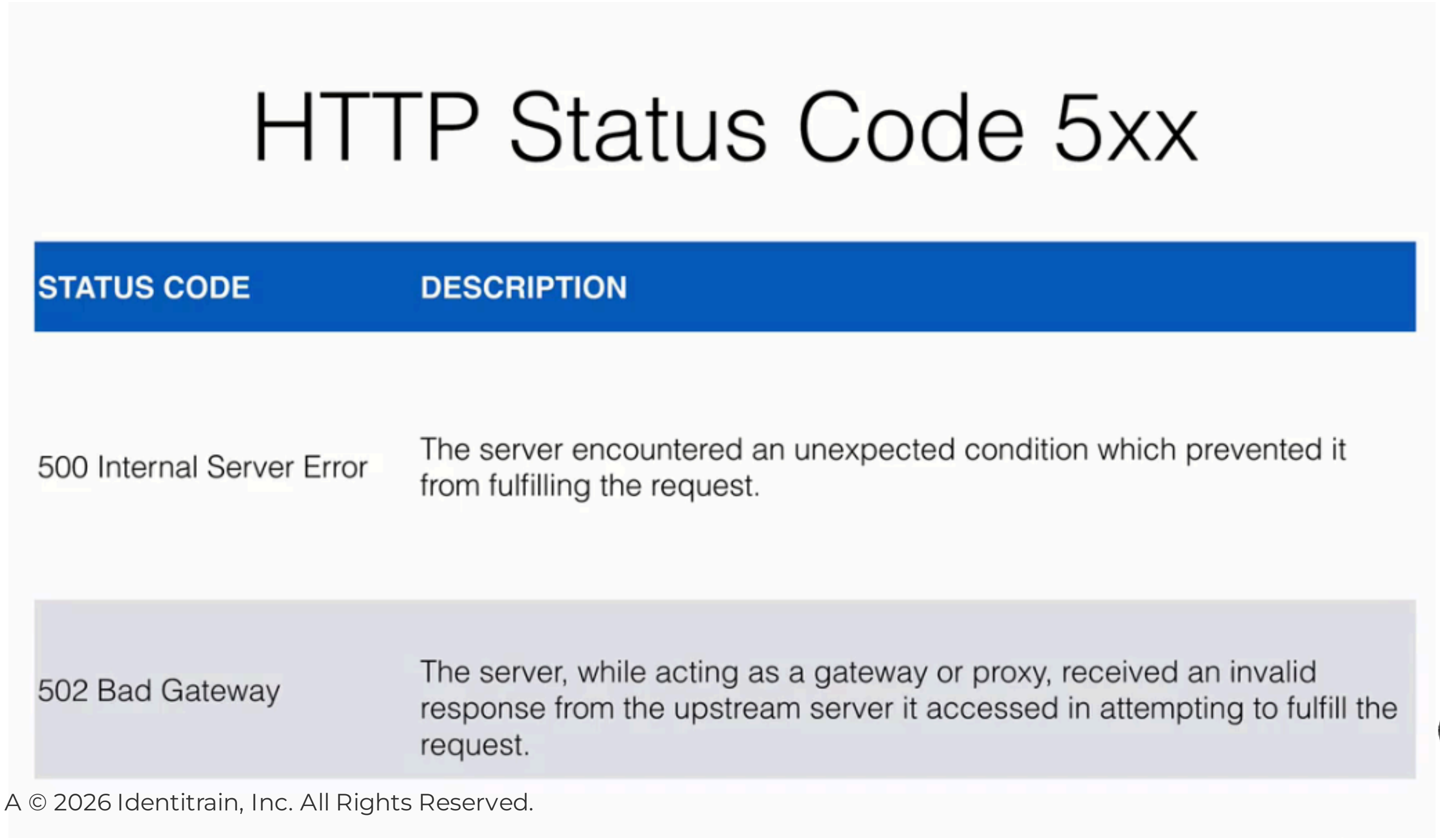


HTTP Status Code 4xx

STATUS CODE	DESCRIPTION
400 Bad Request	The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.
401 Unauthorized	Similar to 403 Forbidden, but specifically for use when authentication is possible but has failed or not yet been provided.
403 Forbidden	The request was a legal request, but the server is refusing to respond to it. Unlike a 401 Unauthorized response, authenticating will make no difference.
404 Not Found	The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.

Working with REST – HTTP Status Codes Continued

- 5xx Codes

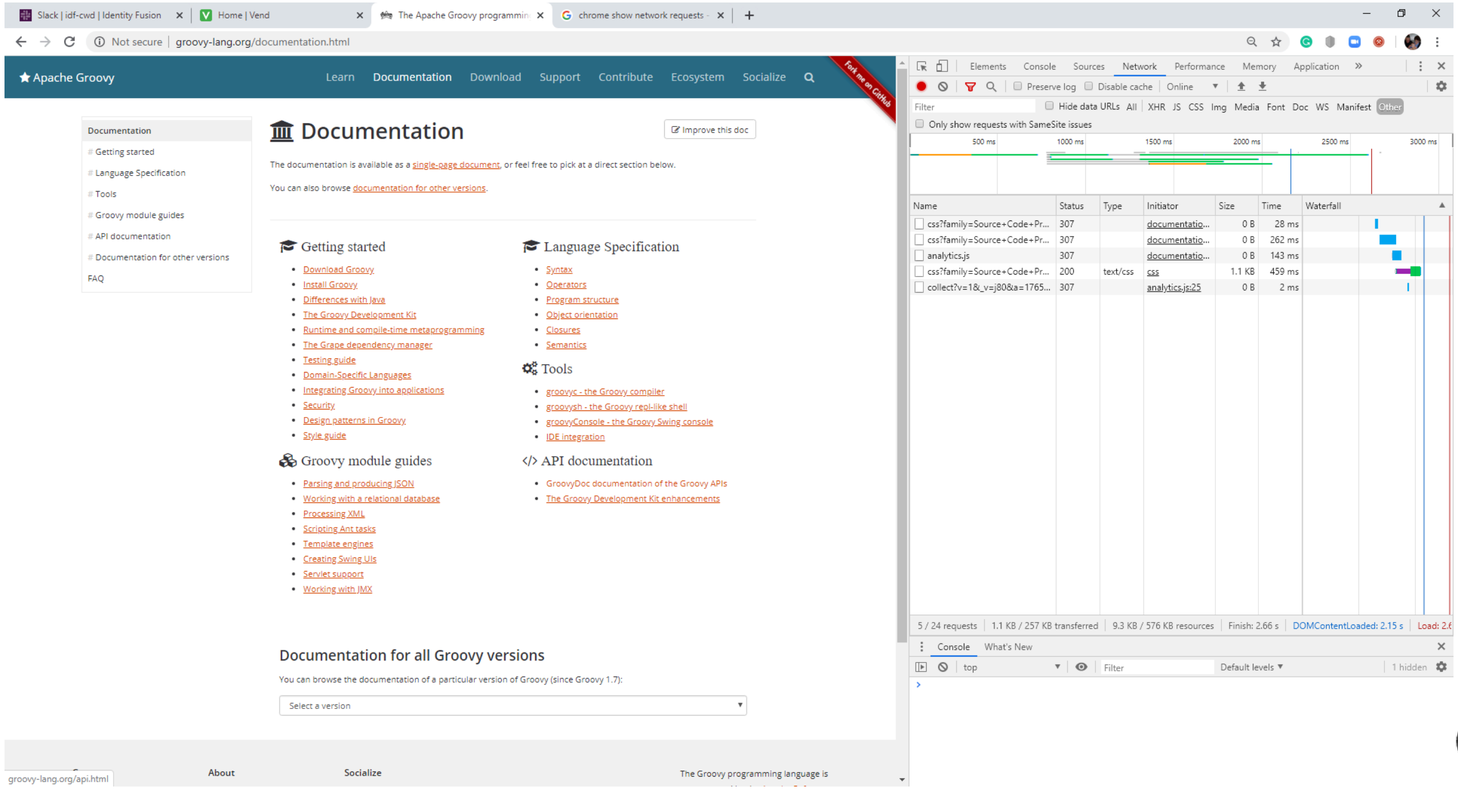


HTTP Status Code 5xx

STATUS CODE	DESCRIPTION
500 Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
502 Bad Gateway	The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

Working with REST – HTTP Status Codes Continued

- Chrome Developer Tools



groovy-lang.org/documentation.html

The documentation is available as a single-page document, or feel free to pick at a direct section below.

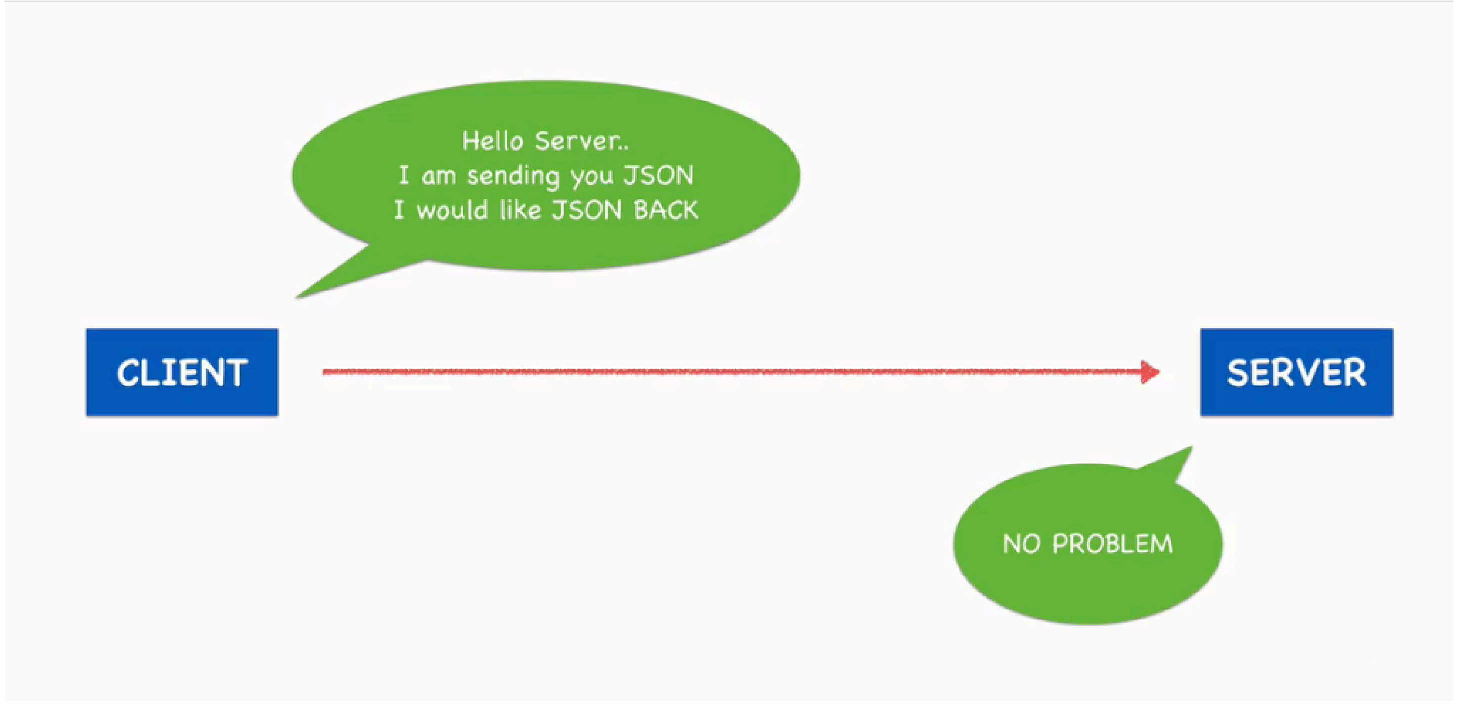
You can also browse documentation for other versions.

Name	Status	Type	Initiator	Size	Time
css?family=Source+Code+Pr...	307		documentatio...	0 B	28 ms
css?family=Source+Code+Pr...	307		documentatio...	0 B	262 ms
analytics.js	307		documentatio...	0 B	143 ms
css?family=Source+Code+Pr...	200	text/css	css	1.1 KB	459 ms
collect?v=1&_v=j80&a=1765...	307		analytics.js:25	0 B	2 ms

5 / 24 requests | 1.1 KB / 257 KB transferred | 9.3 KB / 576 KB resources | Finish: 2.66 s | DOMContentLoaded: 2.15 s | Load: 2.6

Working with REST – Content Negotiation

- Knowing What to Send



Checkpoint

When you open a web browser and visit a URL what is the Http Request Method (verb) that is used by default?

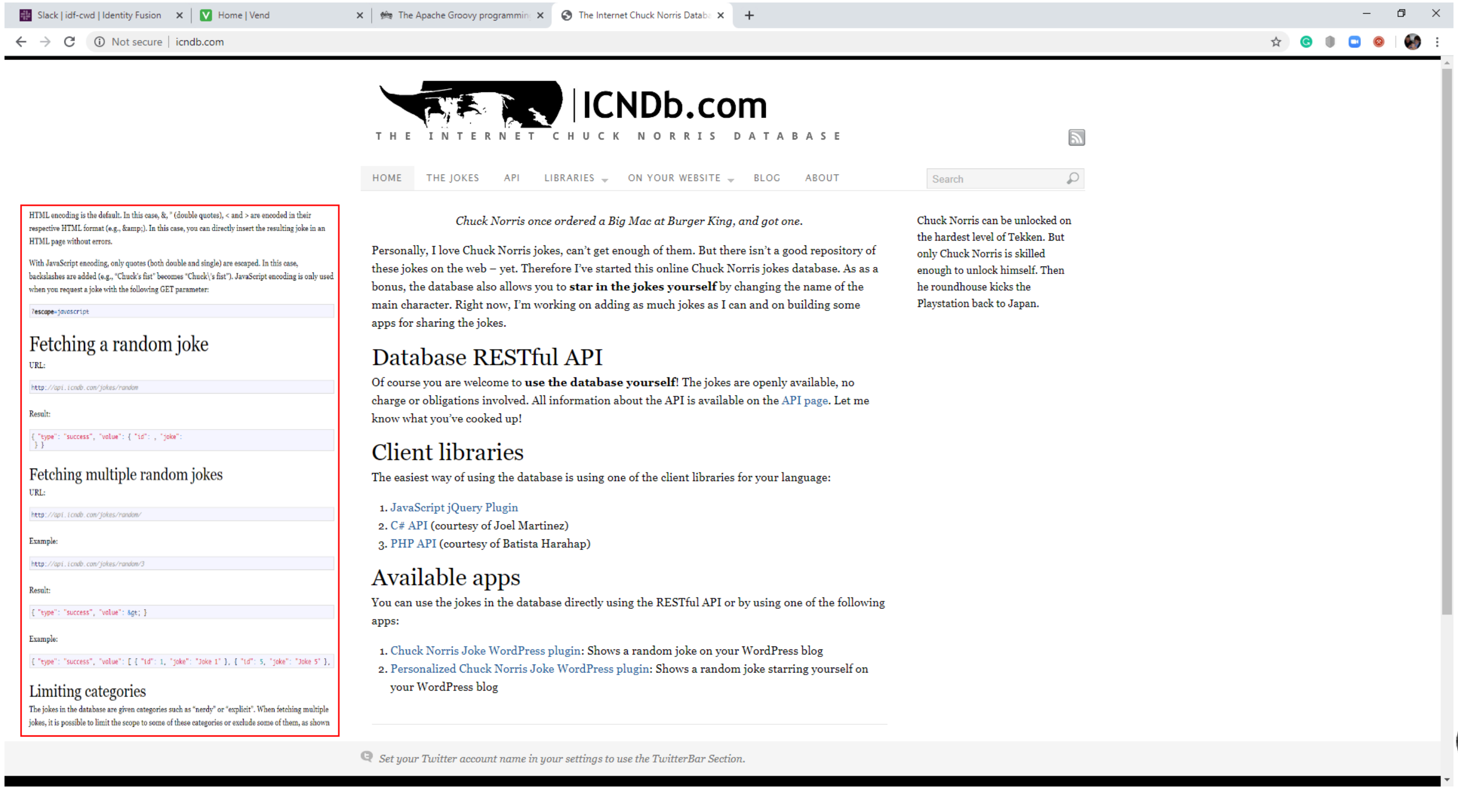
- GET

Working with REST – Using REST based API

- Get Text

Working with REST – REST based API Continued

- Chuck Norris



Slack | idf-cwd | Identity Fusion
Home | Vend
The Apache Groovy programmin
The Internet Chuck Norris Datab
Not secure | icndb.com
ICNDb.com
THE INTERNET CHUCK NORRIS DATABASE
HOME
THE JOKES
API
LIBRARIES
ON YOUR WEBSITE
BLOG
ABOUT
Search

HTML encoding is the default. In this case, &, " (double quotes), < and > are encoded in their respective HTML format (e.g., &amp;). In this case, you can directly insert the resulting joke in an HTML page without errors.

With JavaScript encoding, only quotes (both double and single) are escaped. In this case, backslashes are added (e.g., "Chuck's fist" becomes "Chuck\'s fist"). JavaScript encoding is only used when you request a joke with the following GET parameter:

```
?escape=javascript
```

Fetching a random joke

URL:

```
http://api.icndb.com/jokes/random
```

Result:

```
{ "type": "success", "value": { "id": , "joke":
 } }
```

Fetching multiple random jokes

URL:

```
http://api.icndb.com/jokes/random/
```

Example:

```
http://api.icndb.com/jokes/random/3
```

Result:

```
{ "type": "success", "value": &gt; }
```

Example:

```
{ "type": "success", "value": [ { "id": 1, "joke": "Joke 1" }, { "id": 5, "joke": "Joke 5" },
```

Limiting categories

The jokes in the database are given categories such as "nerdy" or "explicit". When fetching multiple jokes, it is possible to limit the scope to some of these categories or exclude some of them, as shown

Chuck Norris once ordered a Big Mac at Burger King, and got one.

Personally, I love Chuck Norris jokes, can't get enough of them. But there isn't a good repository of these jokes on the web – yet. Therefore I've started this online Chuck Norris jokes database. As as a bonus, the database also allows you to **star in the jokes yourself** by changing the name of the main character. Right now, I'm working on adding as much jokes as I can and on building some apps for sharing the jokes.

Database RESTful API

Of course you are welcome to **use the database yourself**! The jokes are openly available, no charge or obligations involved. All information about the API is available on the API page. Let me know what you've cooked up!

Client libraries

The easiest way of using the database is using one of the client libraries for your language:

1. JavaScript jQuery Plugin
2. C# API (courtesy of Joel Martinez)
3. PHP API (courtesy of Batista Harahap)

Available apps

You can use the jokes in the database directly using the RESTful API or by using one of the following apps:

1. Chuck Norris Joke WordPress plugin: Shows a random joke on your WordPress blog
2. Personalized Chuck Norris Joke WordPress plugin: Shows a random joke starring yourself on your WordPress blog

Chuck Norris can be unlocked on the hardest level of Tekken. But only Chuck Norris is skilled enough to unlock himself. Then he roundhouse kicks the Playstation back to Japan.

Set your Twitter account name in your settings to use the TwitterBar Section.

Working with REST – REST based API Continued

- Demonstration

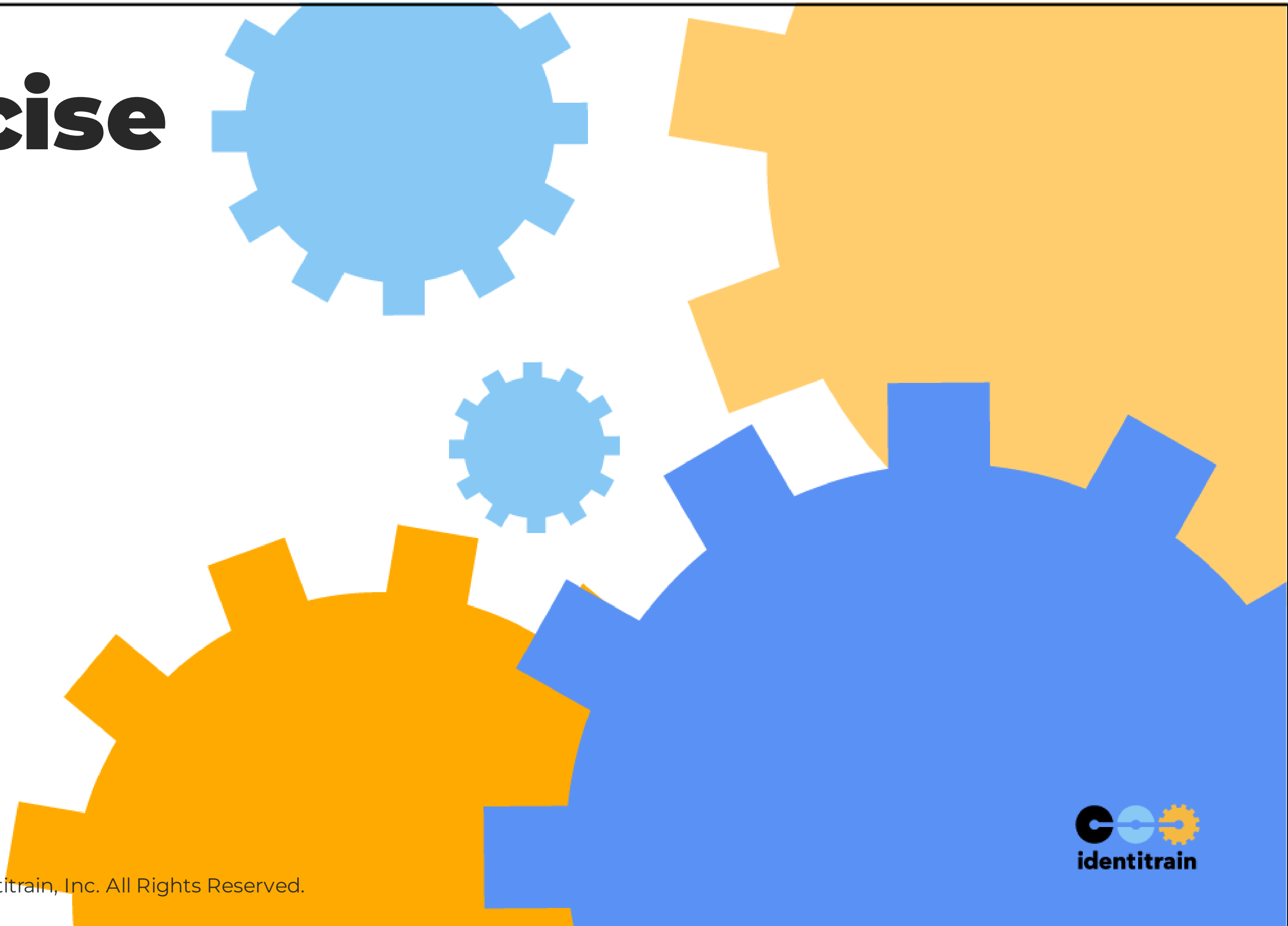
Working with REST – REST based API Continued

- REST with Parameters

Exercise

identitrain

Exercise

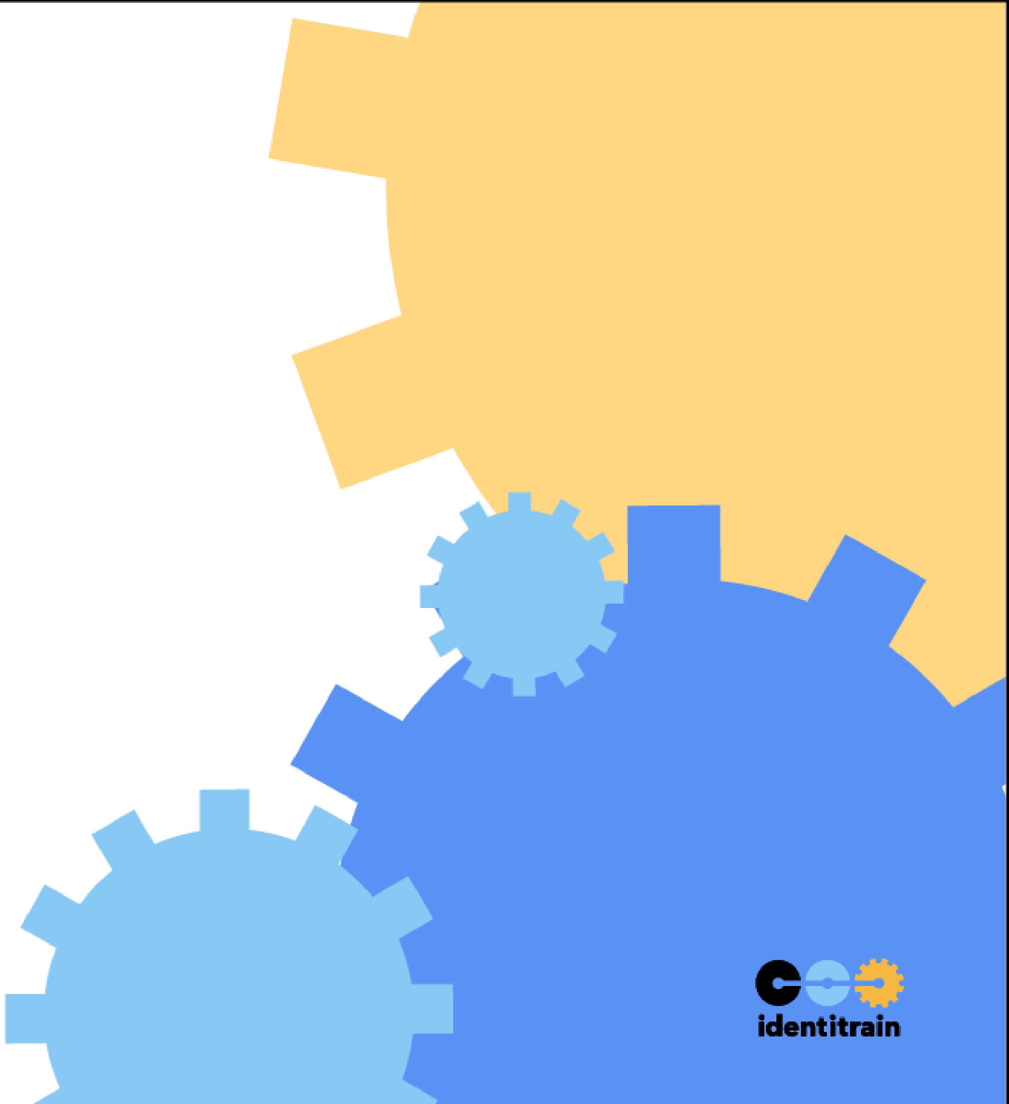


identitrain

Questions?

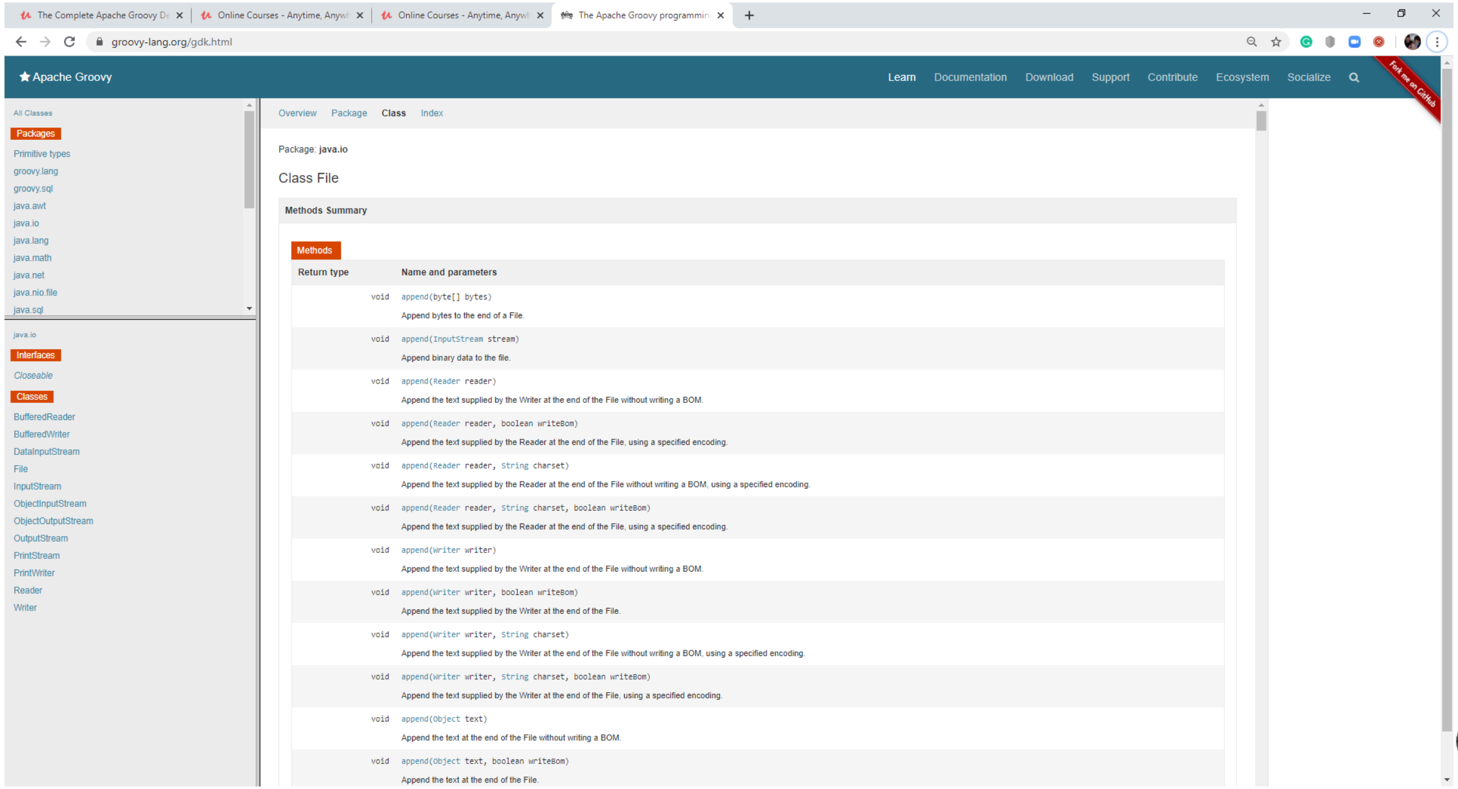
identitrain

GDK



Groovy GDK – Files and I/O

- Documentation



The Complete Apache Groovy De | Online Courses - Anytime, Anyw | Online Courses - Anytime, Anyw | The Apache Groovy programmin

groovy-lang.org/gdk.html

Apache Groovy — Learn · Documentation · Download · Support · Contribute · Ecosystem · Socialize

All Classes

Packages: Primitive types, groovy.lang, groovy.sql, java.awt, java.io, java.lang, java.math, java.net, java.nio.file, java.sql

java.io

Interfaces: *Closeable*

Classes: BufferedReader, BufferedWriter, DataInputStream, File, InputStream, ObjectInputStream, ObjectOutputStream, OutputStream, PrintStream, PrintWriter, Reader, Writer

Overview · Package · Class · Index

Package: **java.io**

Class File

Methods Summary

Methods

Return type	Name and parameters
void	`append(byte[] bytes)` Append bytes to the end of a File.
void	`append(InputStream stream)` Append binary data to the file.
void	`append(Reader reader)` Append the text supplied by the Writer at the end of the File without writing a BOM.
void	`append(Reader reader, boolean writeBom)` Append the text supplied by the Reader at the end of the File, using a specified encoding.
void	`append(Reader reader, String charset)` Append the text supplied by the Reader at the end of the File without writing a BOM, using a specified encoding.
void	`append(Reader reader, String charset, boolean writeBom)` Append the text supplied by the Reader at the end of the File, using a specified encoding.
void	`append(Writer writer)` Append the text supplied by the Writer at the end of the File without writing a BOM.
void	`append(Writer writer, boolean writeBom)` Append the text supplied by the Writer at the end of the File.
void	`append(Writer writer, String charset)` Append the text supplied by the Writer at the end of the File without writing a BOM, using a specified encoding.
void	`append(Writer writer, String charset, boolean writeBom)` Append the text supplied by the Writer at the end of the File, using a specified encoding.
void	`append(Object text)` Append the text at the end of the File without writing a BOM.
void	`append(Object text, boolean writeBom)` Append the text at the end of the File.

Groovy GDK – Files and I/O Continued

- Demonstration

Groovy GDK – Files and I/O Continued

- File Methods Continued

Groovy GDK – Files and I/O Continued

- Directories

Groovy GDK – Files and I/O Continued

- Console IO

Checkpoint

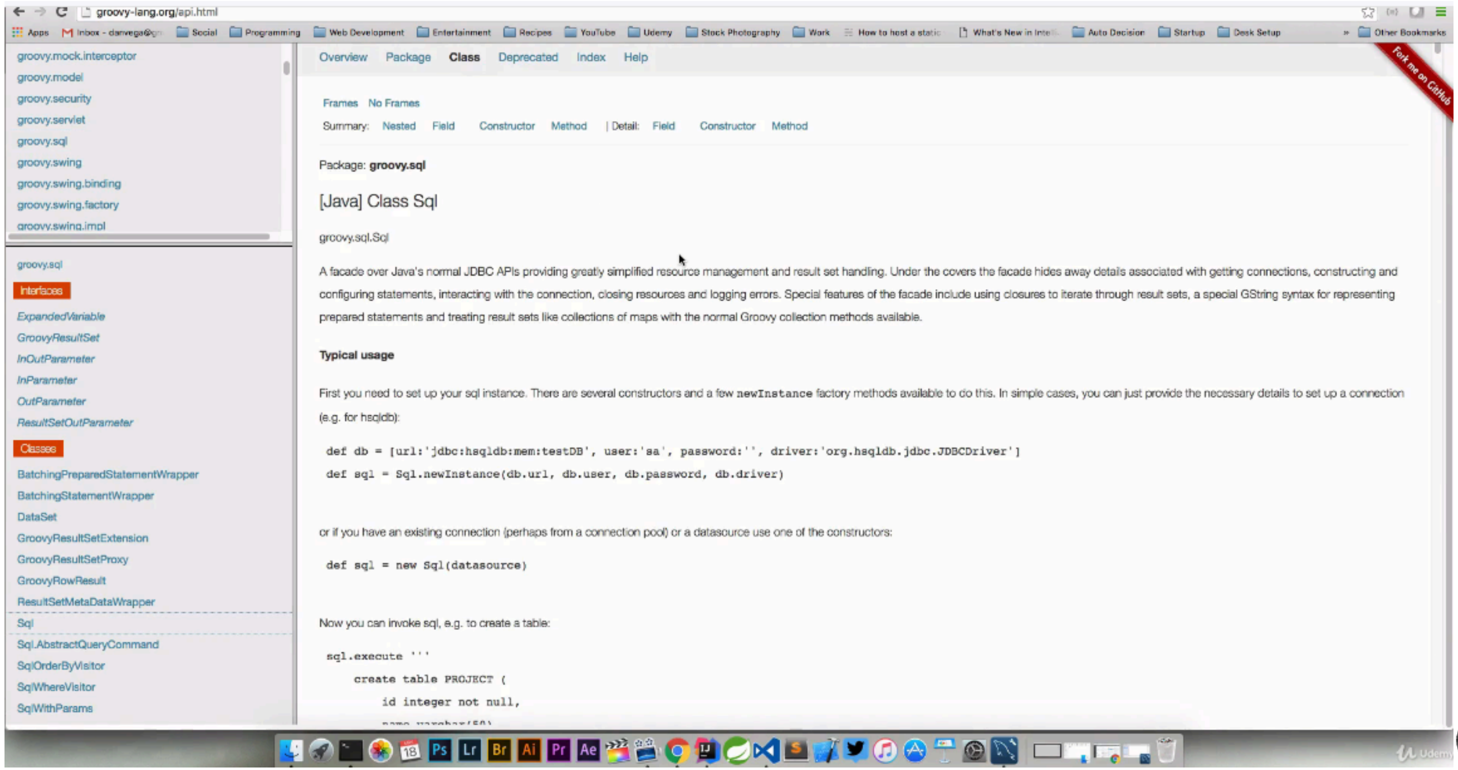
How would you create a new file instance that references the current directory?

- New File(.)

identitrain

Groovy GDK – Database Programming

- Documentation



groovy-lang.org/api.html

Overview Package Class Deprecated Index Help

Frames No Frames

Summary: Nested Field Constructor Method | Detail: Field Constructor Method

Package: **groovy.sql**

[Java] Class Sql

groovy.sql.Sql

A facade over Java's normal JDBC APIs providing greatly simplified resource management and result set handling. Under the covers the facade hides away details associated with getting connections, constructing and configuring statements, interacting with the connection, closing resources and logging errors. Special features of the facade include using closures to iterate through result sets, a special GString syntax for representing prepared statements and treating result sets like collections of maps with the normal Groovy collection methods available.

Typical usage

First you need to set up your sql instance. There are several constructors and a few `newInstance` factory methods available to do this. In simple cases, you can just provide the necessary details to set up a connection (e.g. for hsqldb):

```
def db = [url:'jdbc:hsqldb:mem:testDB', user:'sa', password:'', driver:'org.hsqldb.jdbc.JDBCDriver']
def sql = Sql.newInstance(db.url, db.user, db.password, db.driver)
```

or if you have an existing connection (perhaps from a connection pool) or a datasource use one of the constructors:

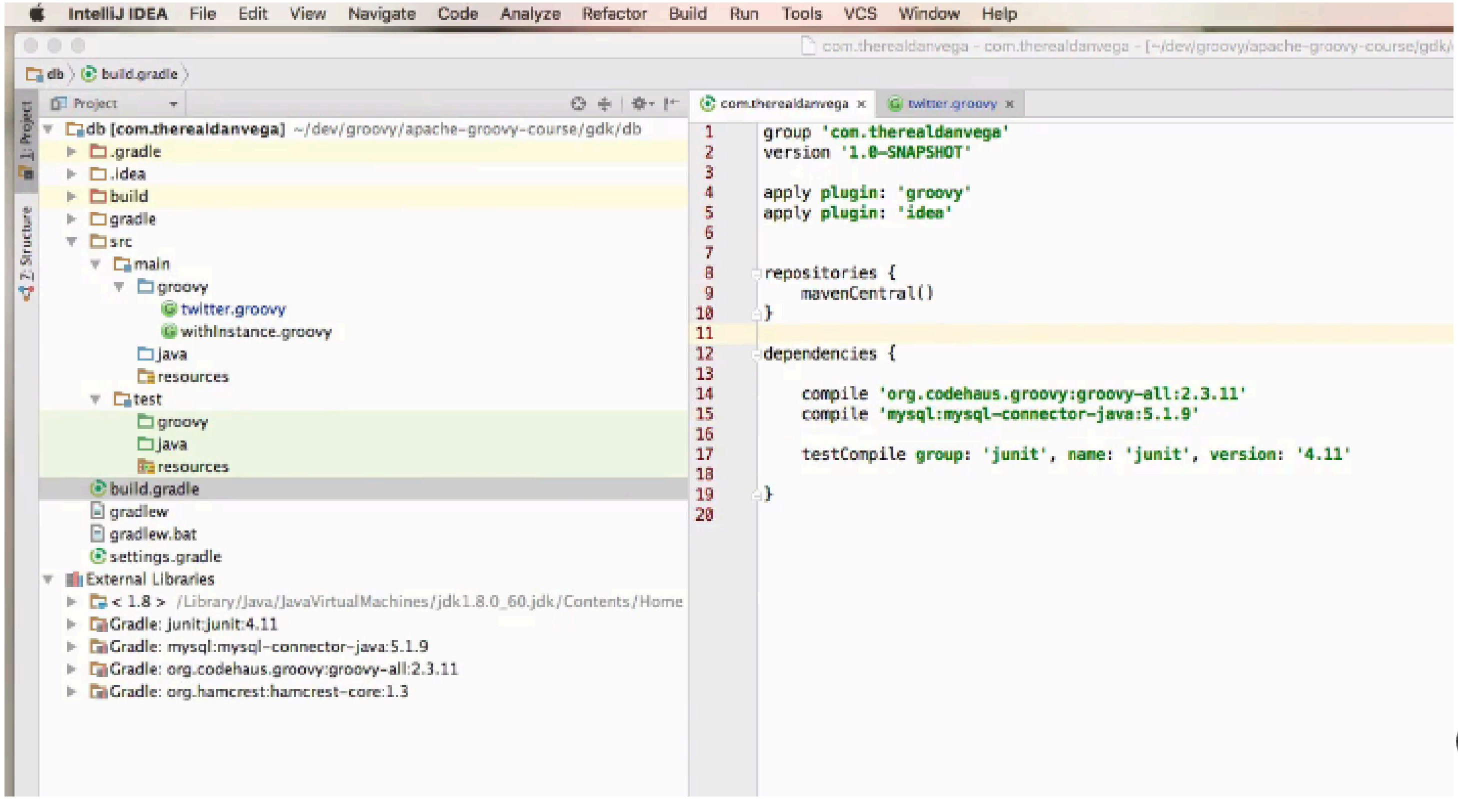
```
def sql = new Sql(datasource)
```

Now you can invoke sql, e.g. to create a table:

```
sql.execute '''
    create table PROJECT (
        id integer not null,
```

Groovy GDK – Database Programming Continued

- Demonstration

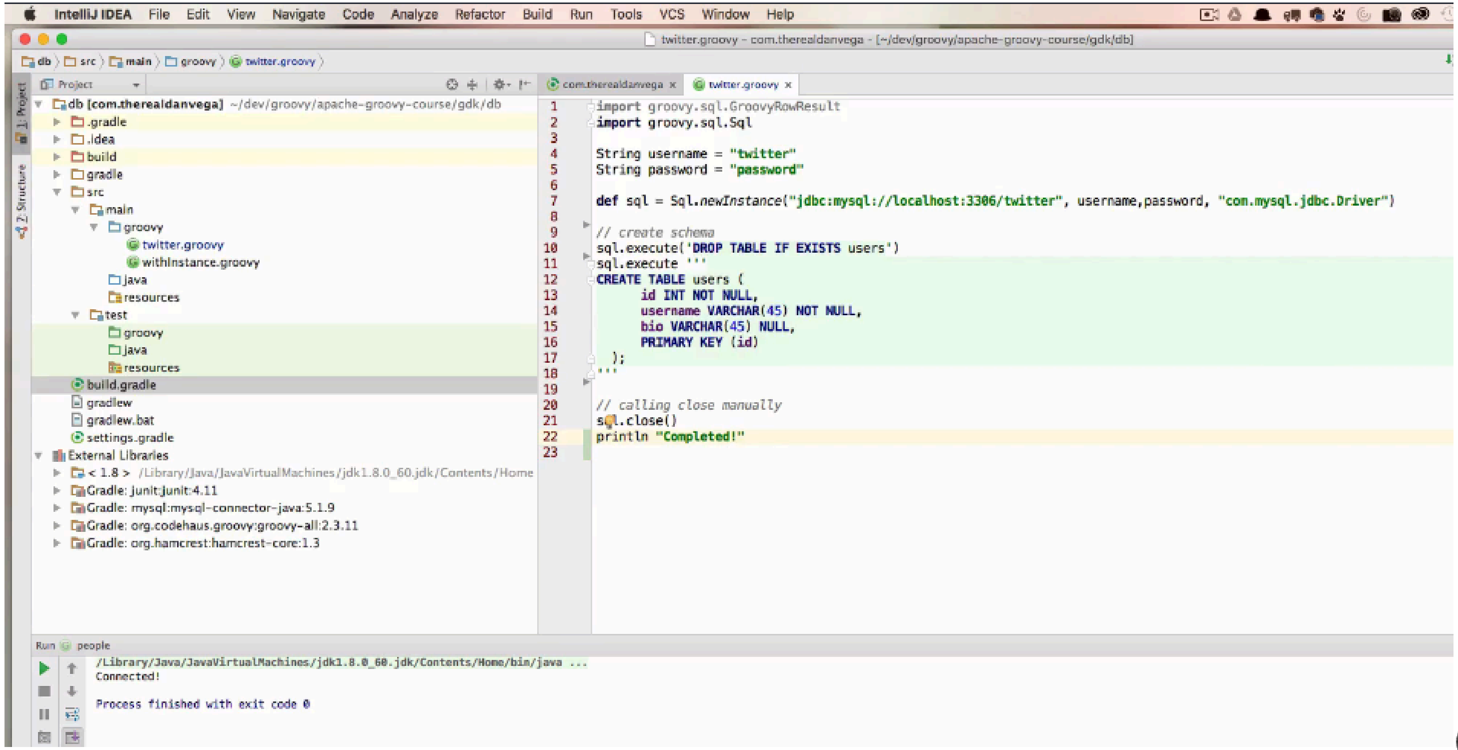

IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
com.therealdanvega - com.therealdanvega - [~/dev/groovy/apache-groovy-course/gdk/
db build.gradle
Project
com.therealdanvega
twitter.groovy

```
db [com.therealdanvega] ~/dev/groovy/apache-groovy-course/gdk/db
  .gradle
  .idea
  build
  gradle
  src
    main
      groovy
        twitter.groovy
        withInstance.groovy
      java
      resources
    test
      groovy
      java
      resources
  build.gradle
  gradlew
  gradlew.bat
  settings.gradle
External Libraries
  < 1.8 > /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home
  Gradle: junit:junit:4.11
  Gradle: mysql:mysql-connector-java:5.1.9
  Gradle: org.codehaus.groovy:groovy-all:2.3.11
  Gradle: org.hamcrest:hamcrest-core:1.3
```

```
group 'com.therealdanvega'
version '1.0-SNAPSHOT'

apply plugin: 'groovy'
apply plugin: 'idea'


repositories {
    mavenCentral()
}

dependencies {

    compile 'org.codehaus.groovy:groovy-all:2.3.11'
    compile 'mysql:mysql-connector-java:5.1.9'

    testCompile group: 'junit', name: 'junit', version: '4.11'

}
```

Groovy GDK – Database Programming Continued

- Demonstration Continued



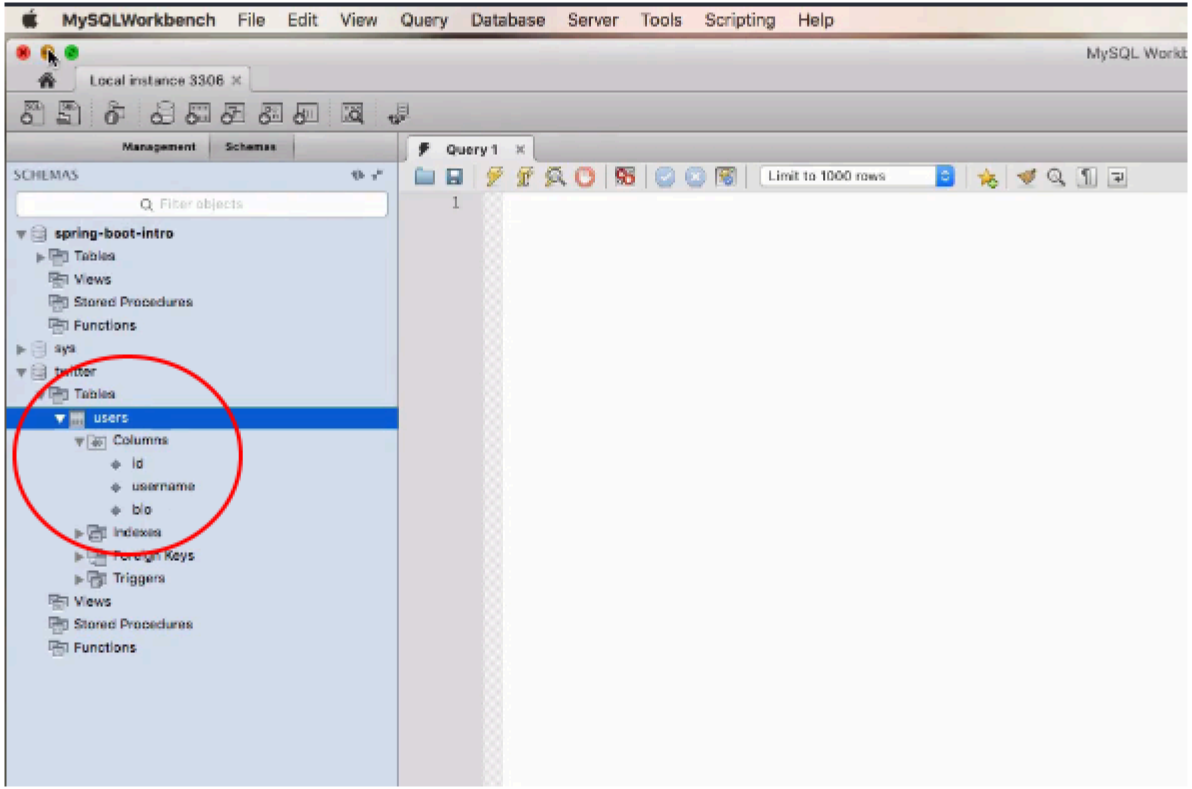
IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
twitter.groovy - com.therealdanvega - [~/dev/groovy/apache-groovy-course/gdk/db]
db src main groovy twitter.groovy
Project
db [com.therealdanvega] ~/dev/groovy/apache-groovy-course/gdk/db
.gradle
.idea
build
gradle
src
main
groovy
twitter.groovy
withInstance.groovy
java
resources
test
groovy
java
resources
build.gradle
gradlew
gradlew.bat
settings.gradle
External Libraries
< 1.8 > /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home
Gradle: junit:junit:4.11
Gradle: mysql:mysql-connector-java:5.1.9
Gradle: org.codehaus.groovy:groovy-all:2.3.11
Gradle: org.hamcrest:hamcrest-core:1.3
com.therealdanvega
twitter.groovy
Run people

```groovy
import groovy.sql.GroovyRowResult
import groovy.sql.Sql

String username = "twitter"
String password = "password"

def sql = Sql.newInstance("jdbc:mysql://localhost:3306/twitter", username,password, "com.mysql.jdbc.Driver")

// create schema
sql.execute('DROP TABLE IF EXISTS users')
sql.execute '''
CREATE TABLE users (
      id INT NOT NULL,
      username VARCHAR(45) NOT NULL,
      bio VARCHAR(45) NULL,
      PRIMARY KEY (id)
  );
'''

// calling close manually
sql.close()
println "Completed!"
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
Connected!

Process finished with exit code 0
```

Groovy GDK – Database Programming Continued

- MySQL Workbench



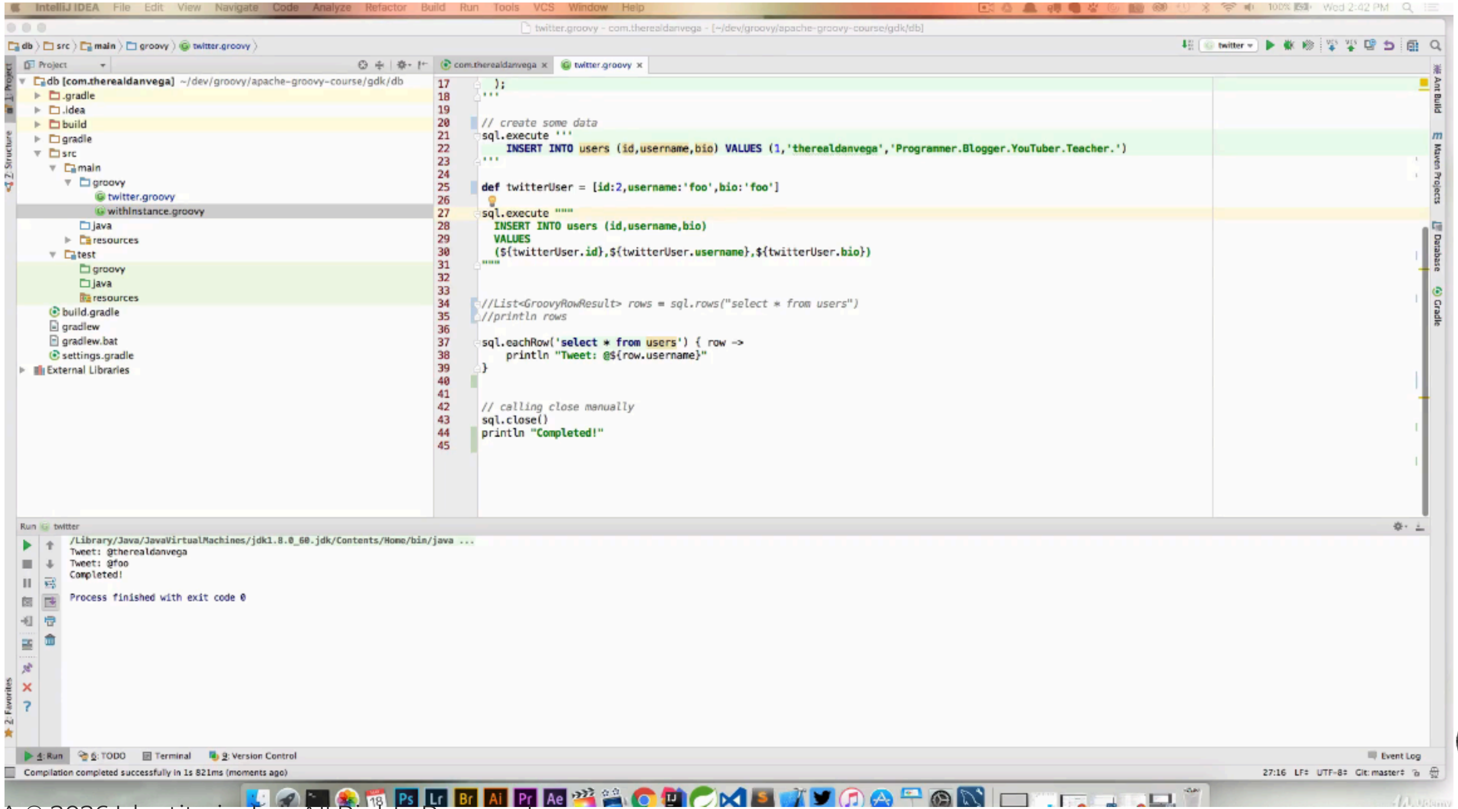
spring-boot-intro
- Tables
- Views
- Stored Procedures
- Functions

sys

twitter
- Tables
  - users
    - Columns
      - id
      - username
      - bio
    - Indexes
    - Foreign Keys
    - Triggers
- Views
- Stored Procedures
- Functions

Groovy GDK – Database Programming Continued

- Demonstration Continued



```groovy
  );
'''

// create some data
sql.execute '''
    INSERT INTO users (id,username,bio) VALUES (1,'therealdanvega','Programmer.Blogger.YouTuber.Teacher.')
'''

def twitterUser = [id:2,username:'foo',bio:'foo']

sql.execute """
  INSERT INTO users (id,username,bio)
  VALUES
  (${twitterUser.id},${twitterUser.username},${twitterUser.bio})
"""


//List<GroovyRowResult> rows = sql.rows("select * from users")
//println rows

sql.eachRow('select * from users') { row ->
    println "Tweet: @${row.username}"
}


// calling close manually
sql.close()
println "Completed!"
```

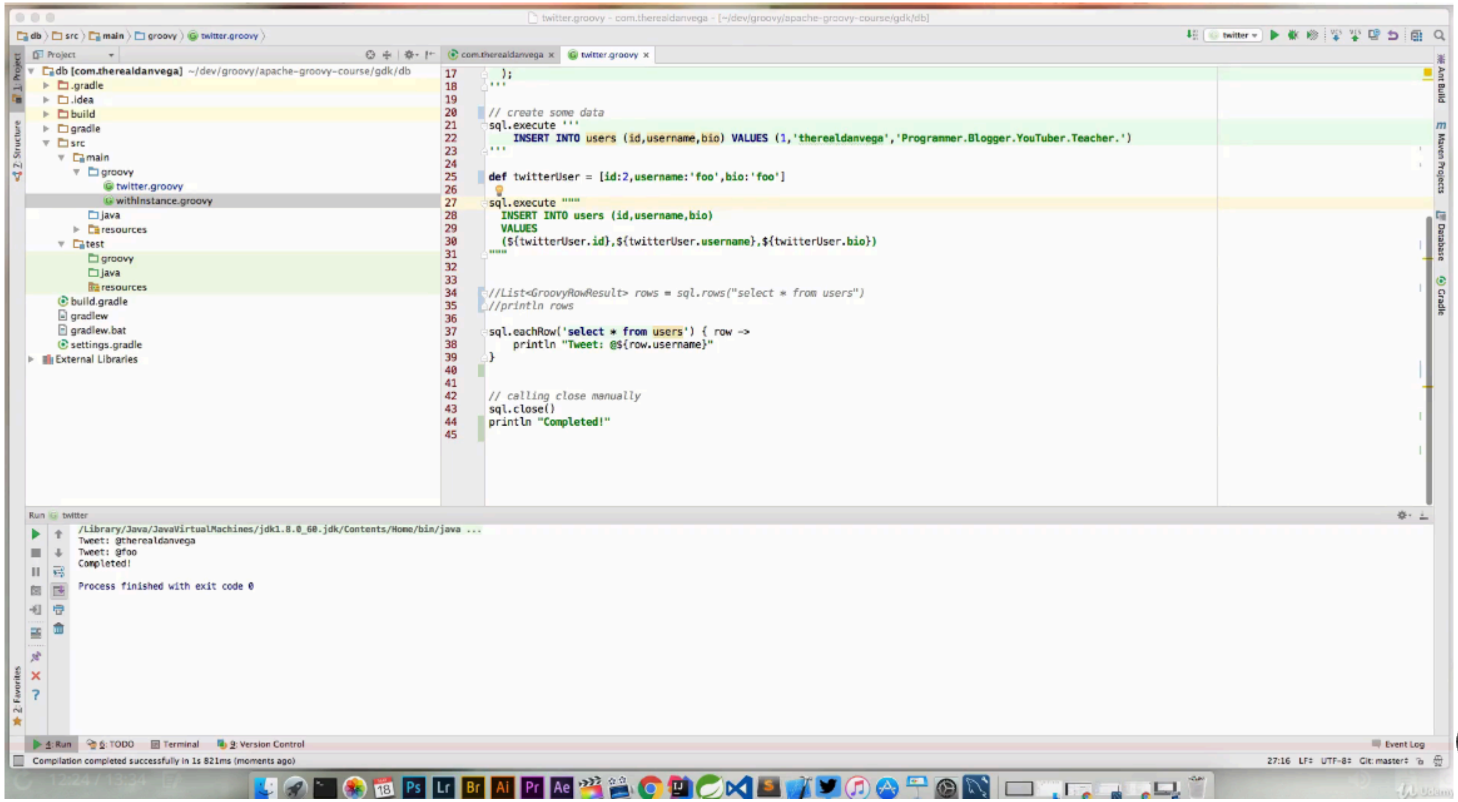
```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
Tweet: @therealdanvega
Tweet: @foo
Completed!

Process finished with exit code 0
```

Groovy GDK – Database Programming Continued

- Documentation – rows Methods

Groovy GDK – Database Programming Continued

- Demonstration Continued



```groovy
   );
'''

// create some data
sql.execute '''
    INSERT INTO users (id,username,bio) VALUES (1,'therealdanvega','Programmer.Blogger.YouTuber.Teacher.')
'''

def twitterUser = [id:2,username:'foo',bio:'foo']

sql.execute """
  INSERT INTO users (id,username,bio)
  VALUES
  (${twitterUser.id},${twitterUser.username},${twitterUser.bio})
"""


//List<GroovyRowResult> rows = sql.rows("select * from users")
//println rows

sql.eachRow('select * from users') { row ->
    println "Tweet: @${row.username}"
}


// calling close manually
sql.close()
println "Completed!"
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
Tweet: @therealdanvega
Tweet: @foo
Completed!

Process finished with exit code 0
```

Groovy GDK – Dates

- Demonstration 1

Groovy GDK – Dates Continued

- Demonstration 2

Groovy GDK – Dates Continued

- Demonstration 3

Exercise

identitrain

Questions?

License

- Apache Groovy for Developers

Runtime Metaprogramming

Groovy Runtime Metaprogramming – The MOP

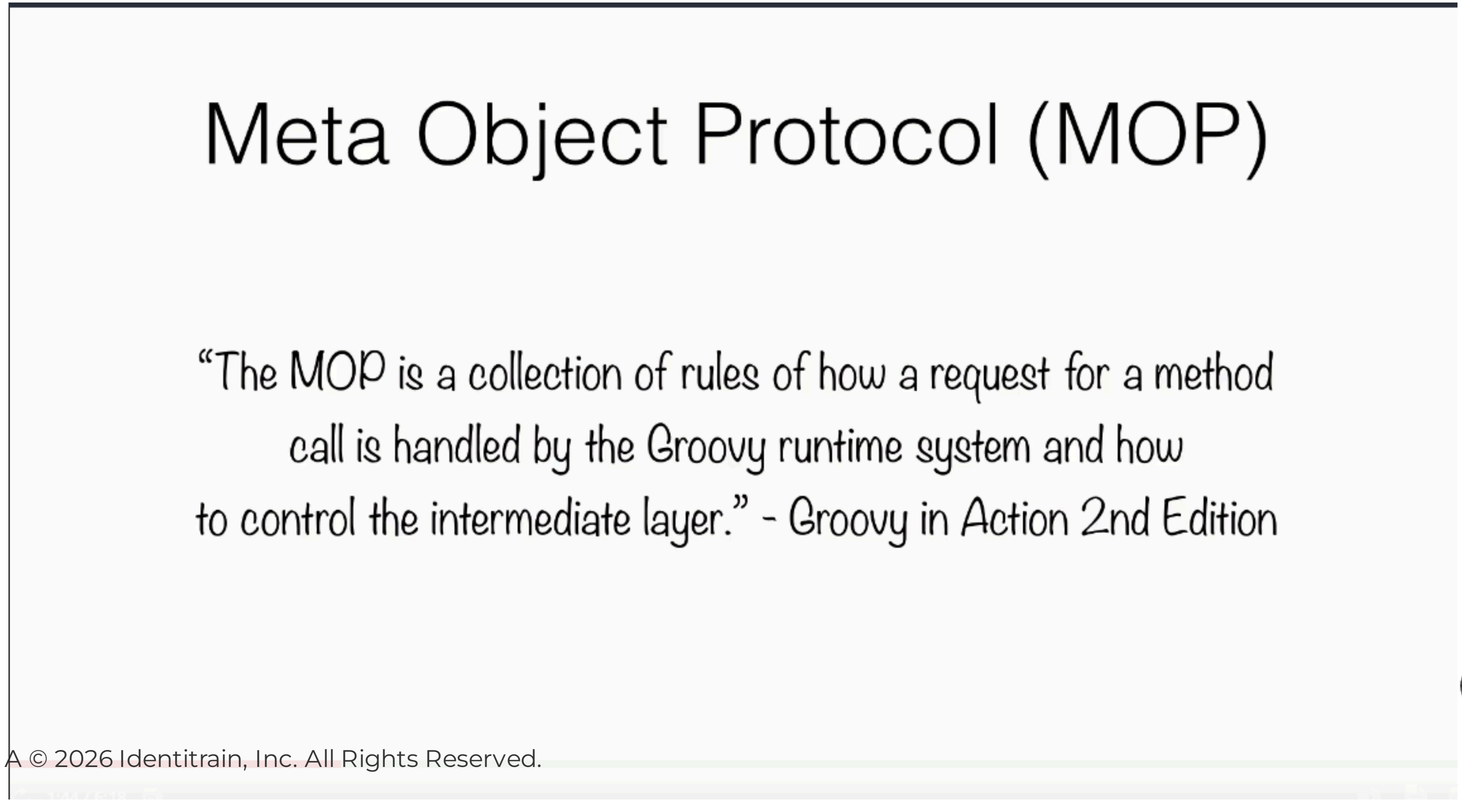
▪

Runtime MetaProgramming

MetaProgramming is the writing of computer programs that write or manipulate other programs (or themselves).

Groovy Runtime Metaprogramming – MOP Continued

- The MOP

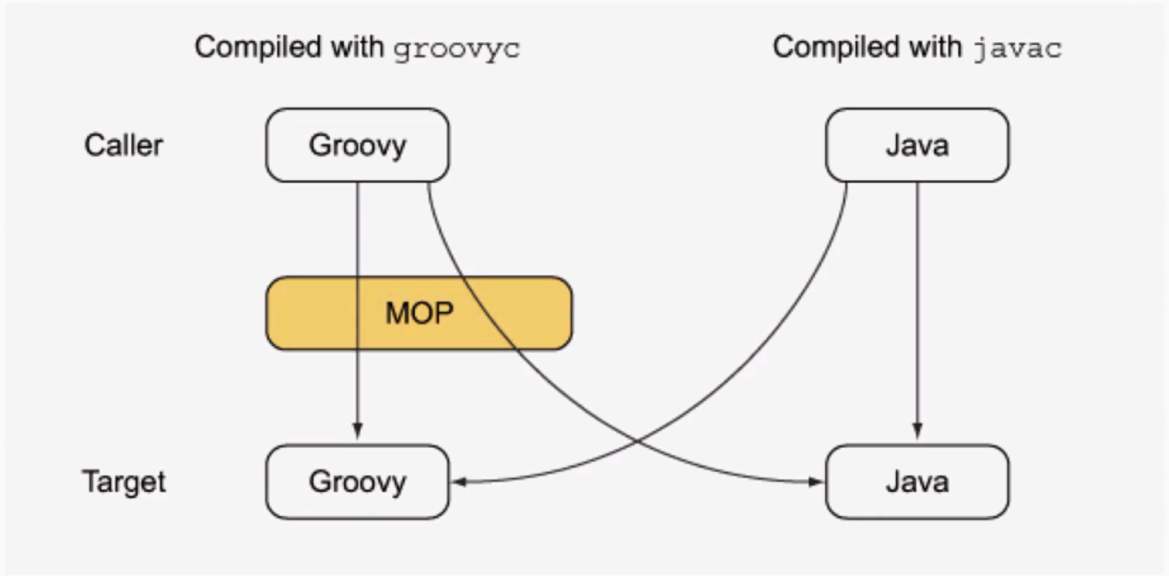


Meta Object Protocol (MOP)

“The MOP is a collection of rules of how a request for a method call is handled by the Groovy runtime system and how to control the intermediate layer.” - Groovy in Action 2nd Edition

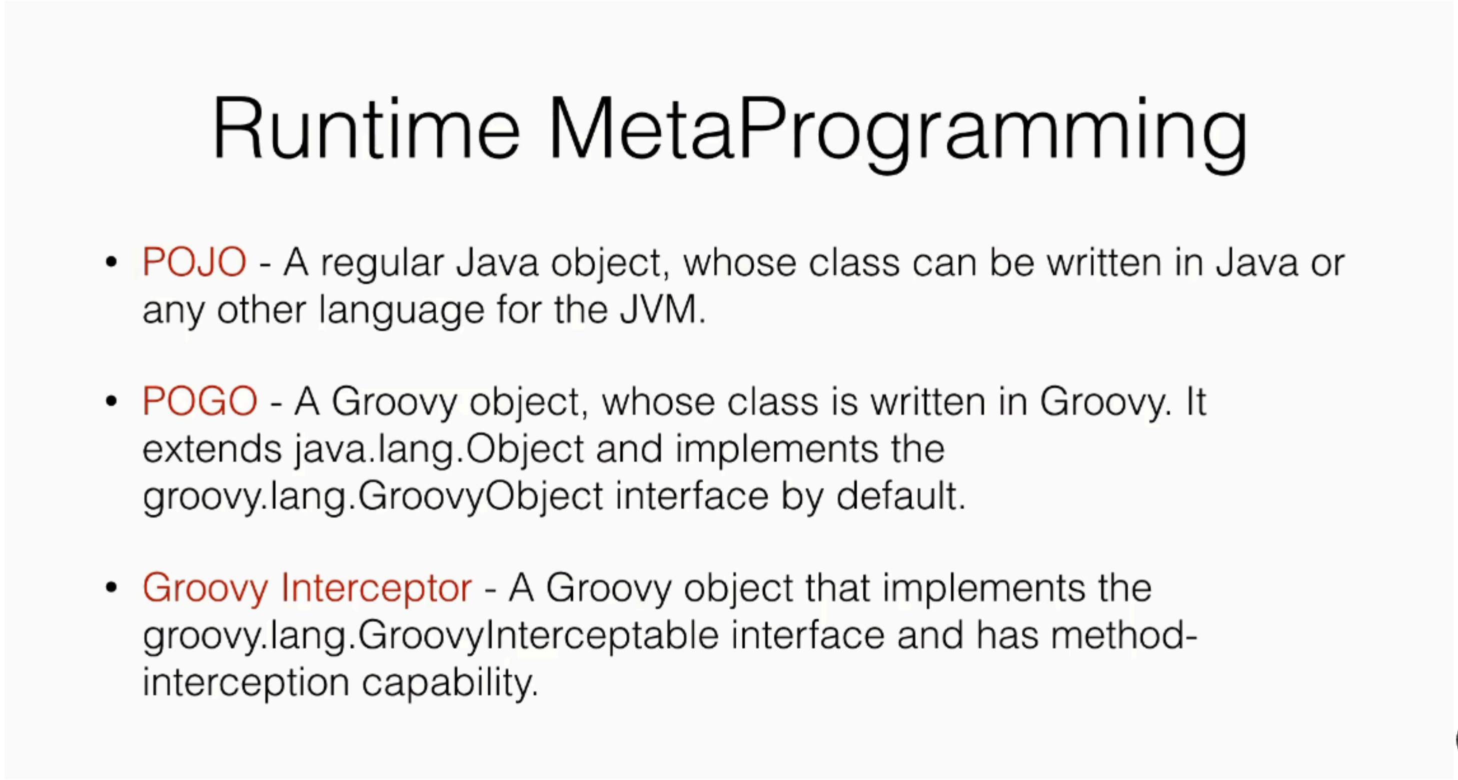
Groovy Runtime Metaprogramming – MOP Continued

- Control Flow



Groovy Metaprogramming – MOP Continued

- Glossary



Runtime MetaProgramming

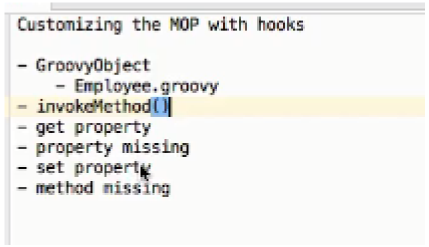
- POJO - A regular Java object, whose class can be written in Java or any other language for the JVM.
- POGO - A Groovy object, whose class is written in Groovy. It extends java.lang.Object and implements the groovy.lang.GroovyObject interface by default.
- Groovy Interceptor - A Groovy object that implements the groovy.lang.GroovyInterceptable interface and has method-interception capability.

Groovy Metaprogramming – MOP Continued

- Decision Tree

Groovy Metaprogramming – Customizing the MOP

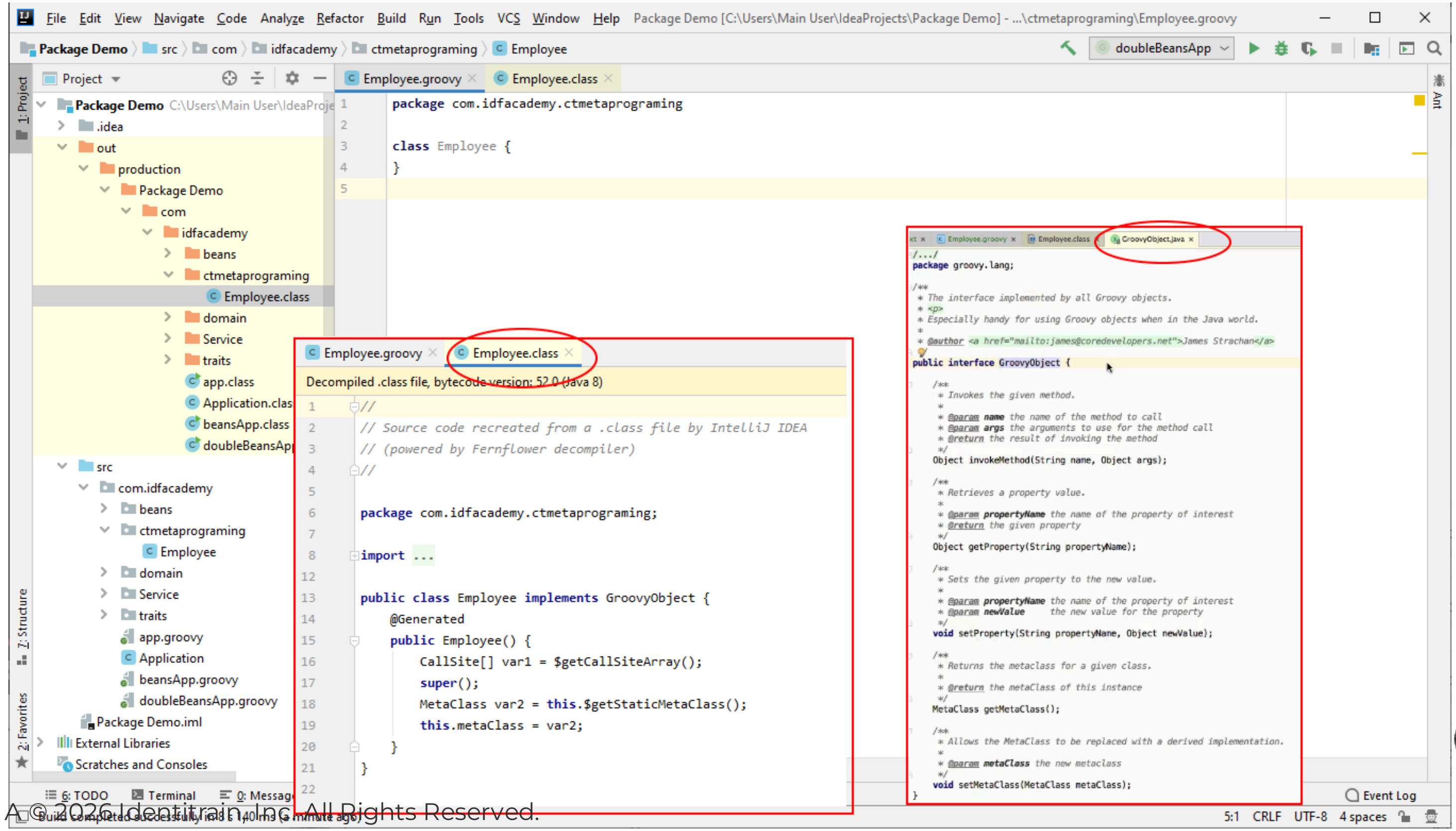
- Overview



```
Customizing the MOP with hooks

- GroovyObject
    - Employee.groovy
- invokeMethod()
- get property
- property missing
- set property
- method missing
```

Groovy Metaprogramming – Customizing the MOP

- A Normal Groovy Class



Employee.groovy

```
package com.idfacademy.ctmetaprograming

class Employee {
}
```

Employee.class

Decompiled .class file, bytecode version: 52.0 (Java 8)

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//

package com.idfacademy.ctmetaprograming;

import ...

public class Employee implements GroovyObject {
    @Generated
    public Employee() {
        CallSite[] var1 = $getCallSiteArray();
        super();
        MetaClass var2 = this.$getStaticMetaClass();
        this.metaClass = var2;
    }
}
```

GroovyObject.java

```
/.../
package groovy.lang;

/**
 * The interface implemented by all Groovy objects.
 * <p>
 * Especially handy for using Groovy objects when in the Java world.
 *
 * @author <a href="mailto:james@coredevelopers.net">James Strachan</a>
 */
public interface GroovyObject {

    /**
     * Invokes the given method.
     *
     * @param name the name of the method to call
     * @param args the arguments to use for the method call
     * @return the result of invoking the method
     */
    Object invokeMethod(String name, Object args);

    /**
     * Retrieves a property value.
     *
     * @param propertyName the name of the property of interest
     * @return the given property
     */
    Object getProperty(String propertyName);

    /**
     * Sets the given property to the new value.
     *
     * @param propertyName the name of the property of interest
     * @param newValue     the new value for the property
     */
    void setProperty(String propertyName, Object newValue);

    /**
     * Returns the metaclass for a given class.
     *
     * @return the metaClass of this instance
     */
    MetaClass getMetaClass();

    /**
     * Allows the MetaClass to be replaced with a derived implementation.
     *
     * @param metaClass the new metaclass
     */
    void setMetaClass(MetaClass metaClass);
}
```

Groovy Metaprogramming – Customizing the MOP

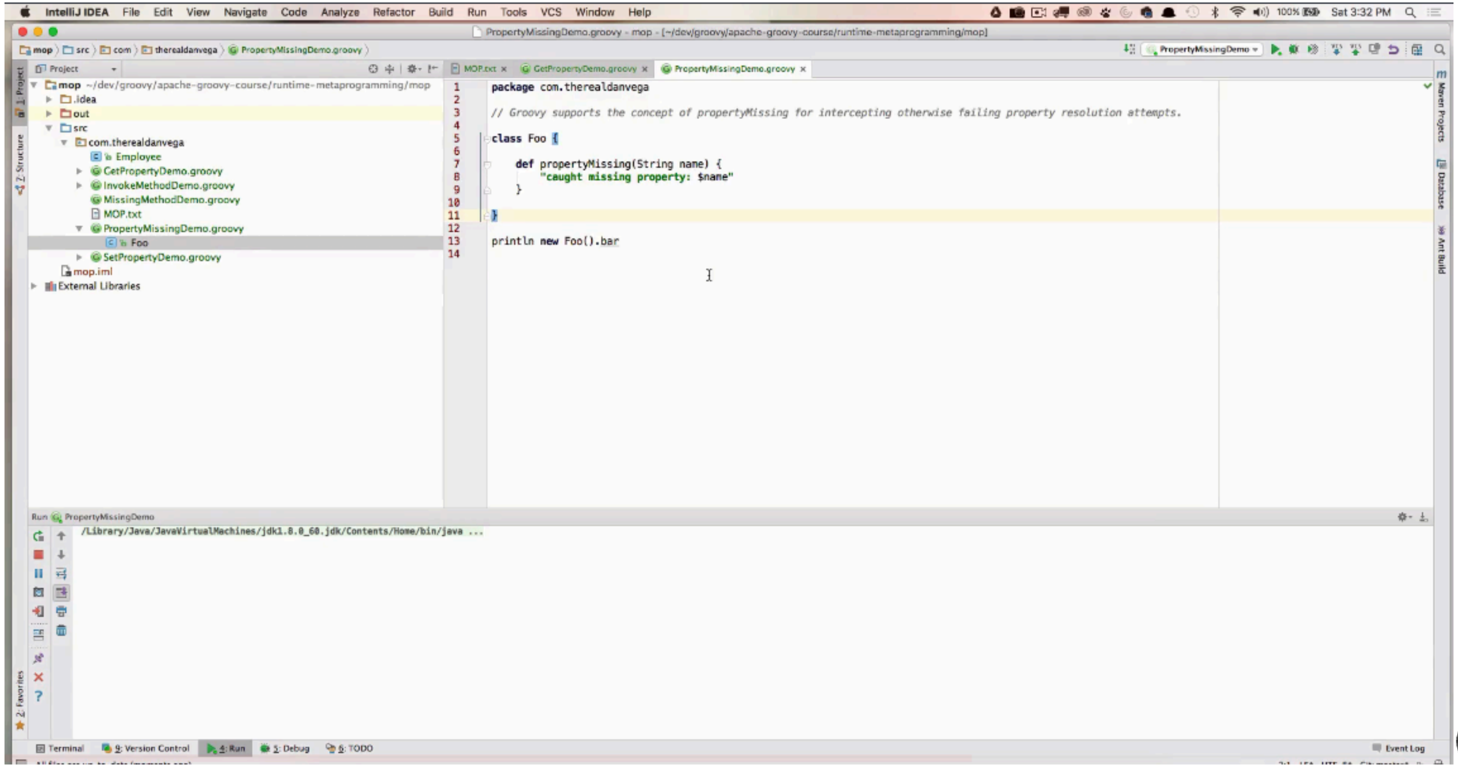
- Our Own InvokeMethod

Groovy Metaprogramming – Customizing the MOP

- getProperty

Groovy Metaprogramming – Customizing the MOP

- Property Missing



```
package com.therealdanvega

// Groovy supports the concept of propertyMissing for intercepting otherwise failing property resolution attempts.

class Foo {

    def propertyMissing(String name) {
        "caught missing property: $name"
    }

}

println new Foo().bar
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
```

Groovy Metaprogramming – Customizing the MOP

- Demonstration

Groovy Metaprogramming – Customizing the MOP

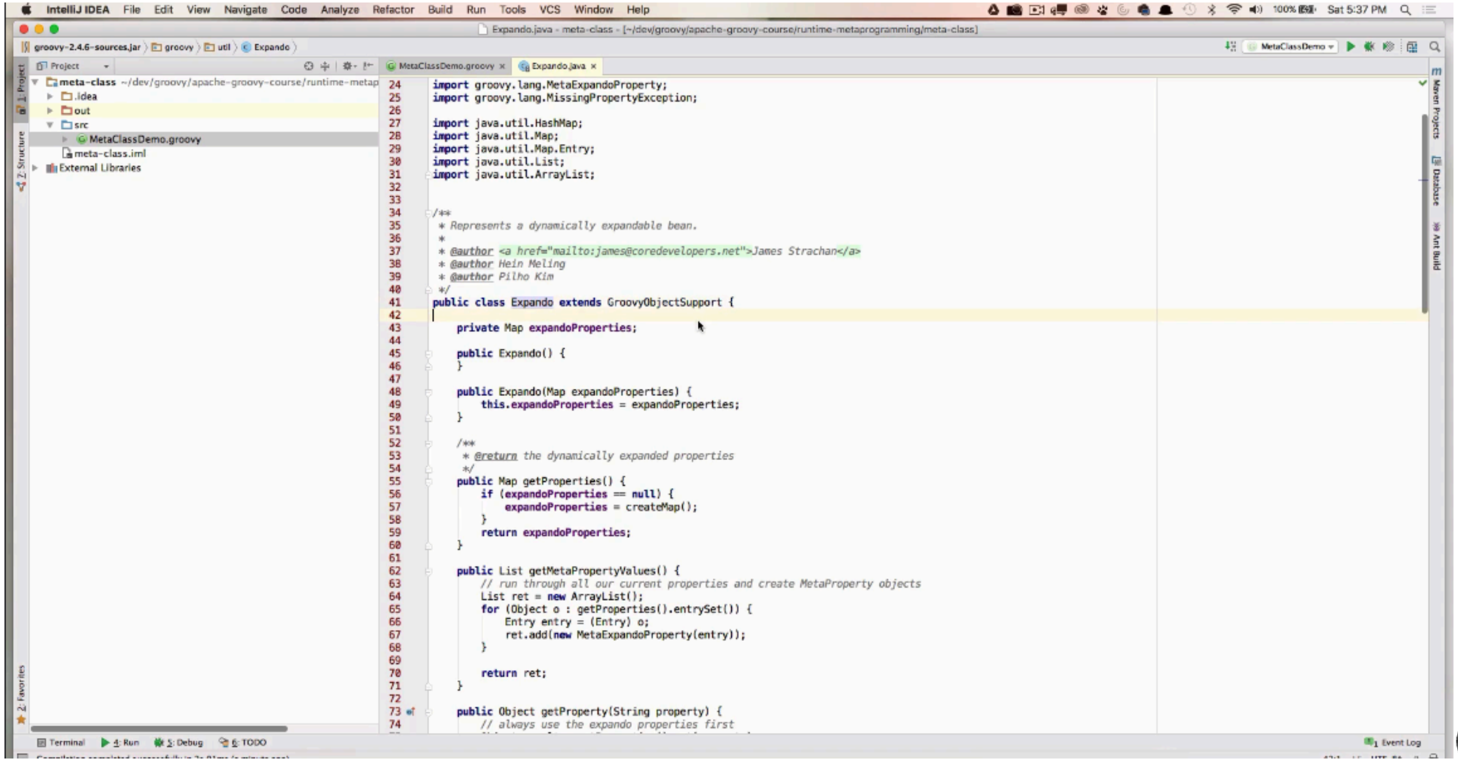
- Missing Method

Groovy Metaprogramming – MetaClass

- Introduction

Groovy Metaprogramming – MetaClass Continued

- Expando



```java
import groovy.lang.MetaExpandoProperty;
import groovy.lang.MissingPropertyException;

import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.List;
import java.util.ArrayList;


/**
 * Represents a dynamically expandable bean.
 *
 * @author <a href="mailto:james@coredevelopers.net">James Strachan</a>
 * @author Hein Meling
 * @author Pilho Kim
 */
public class Expando extends GroovyObjectSupport {

    private Map expandoProperties;

    public Expando() {
    }

    public Expando(Map expandoProperties) {
        this.expandoProperties = expandoProperties;
    }

    /**
     * @return the dynamically expanded properties
     */
    public Map getProperties() {
        if (expandoProperties == null) {
            expandoProperties = createMap();
        }
        return expandoProperties;
    }

    public List getMetaPropertyValues() {
        // run through all our current properties and create MetaProperty objects
        List ret = new ArrayList();
        for (Object o : getProperties().entrySet()) {
            Entry entry = (Entry) o;
            ret.add(new MetaExpandoProperty(entry));
        }

        return ret;
    }

    public Object getProperty(String property) {
        // always use the expando properties first
```

Groovy Metaprogramming – MetaClass Continued

Groovy Metaprogramming – MetaClass Continued

- Metaclass Modification

Checkpoint

True or False: If we make a call from Java to Groovy that call will go through the Meta Object Protocol (MOP).

- False

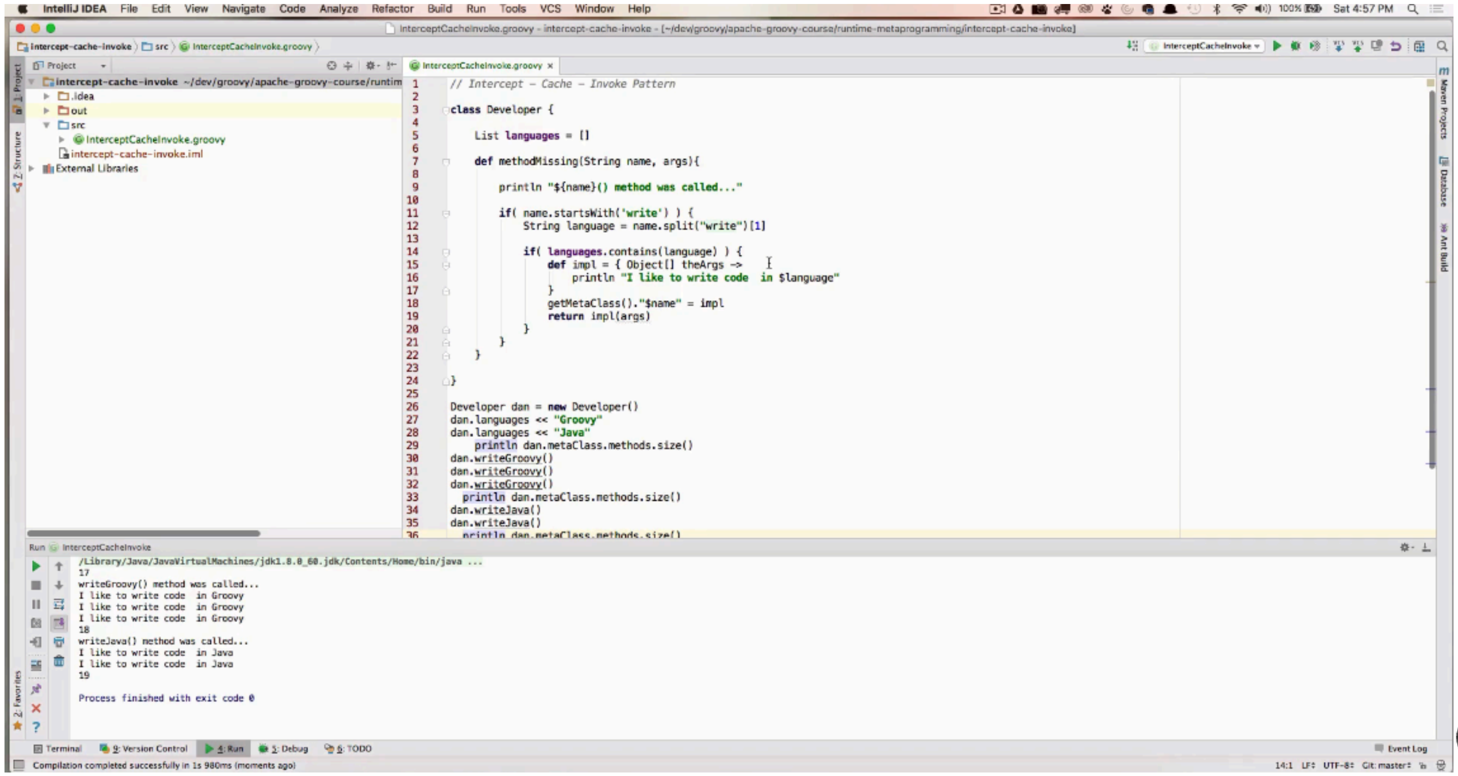
Groovy Metaprogramming – Categories

- Creating and Using a Category

Groovy Metaprogramming – Categories Continued

- Built-in Categories

Groovy Metaprogramming – Intercept Cache Invoke

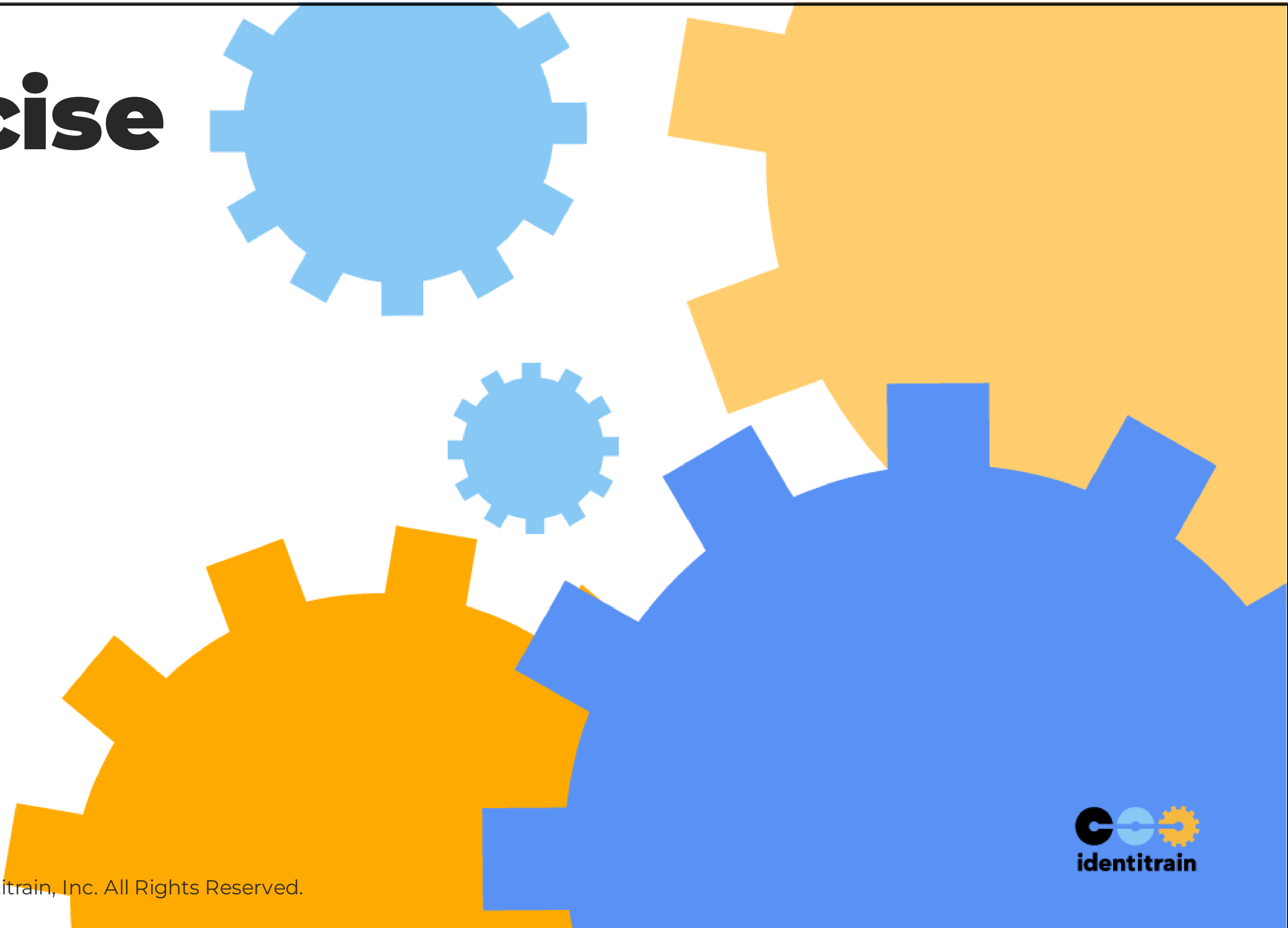


```groovy
// Intercept - Cache - Invoke Pattern

class Developer {

    List languages = []

    def methodMissing(String name, args){

        println "${name}() method was called..."

        if( name.startsWith('write') ) {
            String language = name.split("write")[1]

            if( languages.contains(language) ) {
                def impl = { Object[] theArgs ->
                    println "I like to write code  in $language"
                }
                getMetaClass()."$name" = impl
                return impl(args)
            }
        }
    }

}

Developer dan = new Developer()
dan.languages << "Groovy"
dan.languages << "Java"
    println dan.metaClass.methods.size()
dan.writeGroovy()
dan.writeGroovy()
dan.writeGroovy()
  println dan.metaClass.methods.size()
dan.writeJava()
dan.writeJava()
  println dan.metaClass.methods.size()
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
17
writeGroovy() method was called...
I like to write code  in Groovy
I like to write code  in Groovy
I like to write code  in Groovy
18
writeJava() method was called...
I like to write code  in Java
I like to write code  in Java
19

Process finished with exit code 0
```

Compilation completed successfully in 1s 980ms (moments ago)

Exercise



identitrain

Exercise

identitrain

Questions?

Compile Time Metaprogramming

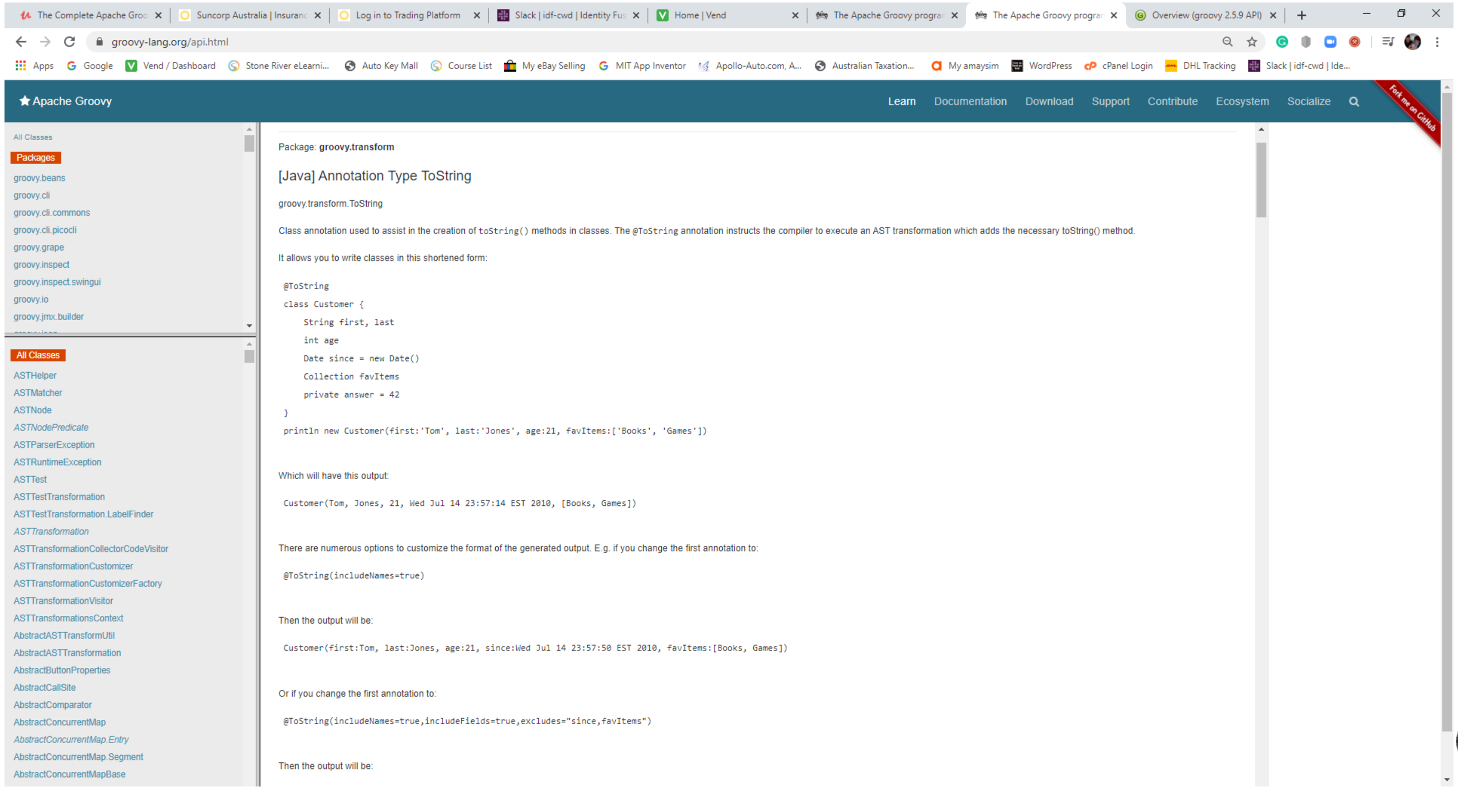
identitrain

Compile Time Metaprogramming

- AST Transformations

Compile Time Metaprogramming

- ToString Documentation



Package: **groovy.transform**

[Java] Annotation Type ToString

groovy.transform.ToString

Class annotation used to assist in the creation of `toString()` methods in classes. The `@ToString` annotation instructs the compiler to execute an AST transformation which adds the necessary toString() method.

It allows you to write classes in this shortened form:

```
@ToString
class Customer {
    String first, last
    int age
    Date since = new Date()
    Collection favItems
    private answer = 42
}
println new Customer(first:'Tom', last:'Jones', age:21, favItems:['Books', 'Games'])
```

Which will have this output:

```
Customer(Tom, Jones, 21, Wed Jul 14 23:57:14 EST 2010, [Books, Games])
```

There are numerous options to customize the format of the generated output. E.g. if you change the first annotation to:

```
@ToString(includeNames=true)
```

Then the output will be:

```
Customer(first:Tom, last:Jones, age:21, since:Wed Jul 14 23:57:50 EST 2010, favItems:[Books, Games])
```

Or if you change the first annotation to:

```
@ToString(includeNames=true,includeFields=true,excludes="since,favItems")
```

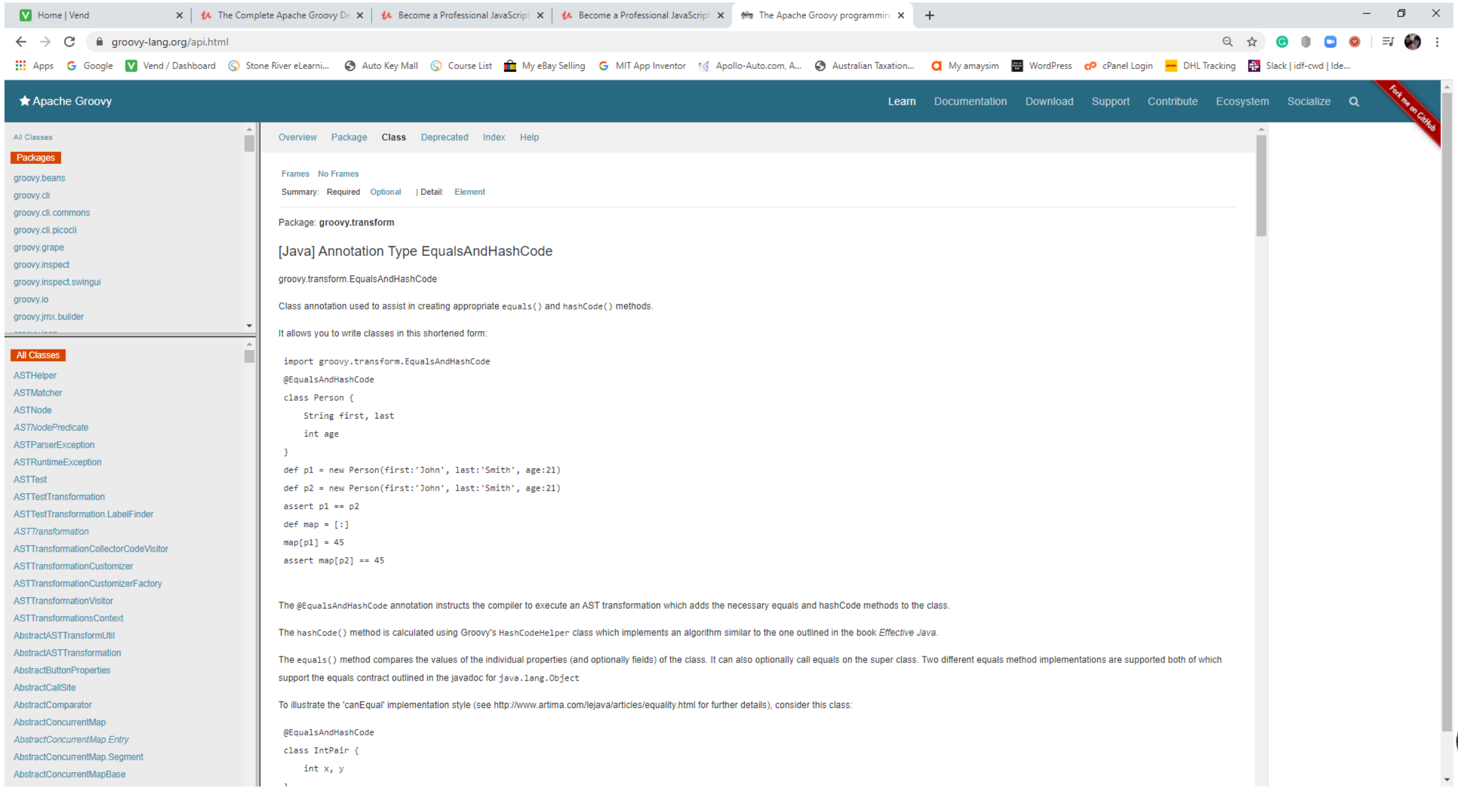
Then the output will be:

Compile Time Metaprogramming

- ToString Example

Compile Time Metaprogramming

- EqualsandHashCode Transformation



Package: **groovy.transform**

[Java] Annotation Type EqualsAndHashCode

groovy.transform.EqualsAndHashCode

Class annotation used to assist in creating appropriate `equals()` and `hashCode()` methods.

It allows you to write classes in this shortened form:

```
import groovy.transform.EqualsAndHashCode
@EqualsAndHashCode
class Person {
    String first, last
    int age
}
def p1 = new Person(first:'John', last:'Smith', age:21)
def p2 = new Person(first:'John', last:'Smith', age:21)
assert p1 == p2
def map = [:]
map[p1] = 45
assert map[p2] == 45
```

The `@EqualsAndHashCode` annotation instructs the compiler to execute an AST transformation which adds the necessary equals and hashCode methods to the class.

The `hashCode()` method is calculated using Groovy's `HashCodeHelper` class which implements an algorithm similar to the one outlined in the book *Effective Java*.

The `equals()` method compares the values of the individual properties (and optionally fields) of the class. It can also optionally call equals on the super class. Two different equals method implementations are supported both of which support the equals contract outlined in the javadoc for `java.lang.Object`

To illustrate the 'canEqual' implementation style (see http://www.artima.com/lejava/articles/equality.html for further details), consider this class:

```
@EqualsAndHashCode
class IntPair {
    int x, y
```

Compile Time Metaprogramming

- EqualsAndHashcode Demo

Compile Time Metaprogramming

- EqualsAndHashcode Demo Continued

Compile Time Metaprogramming

- @TupleConstructor Demo

Compile Time Metaprogramming

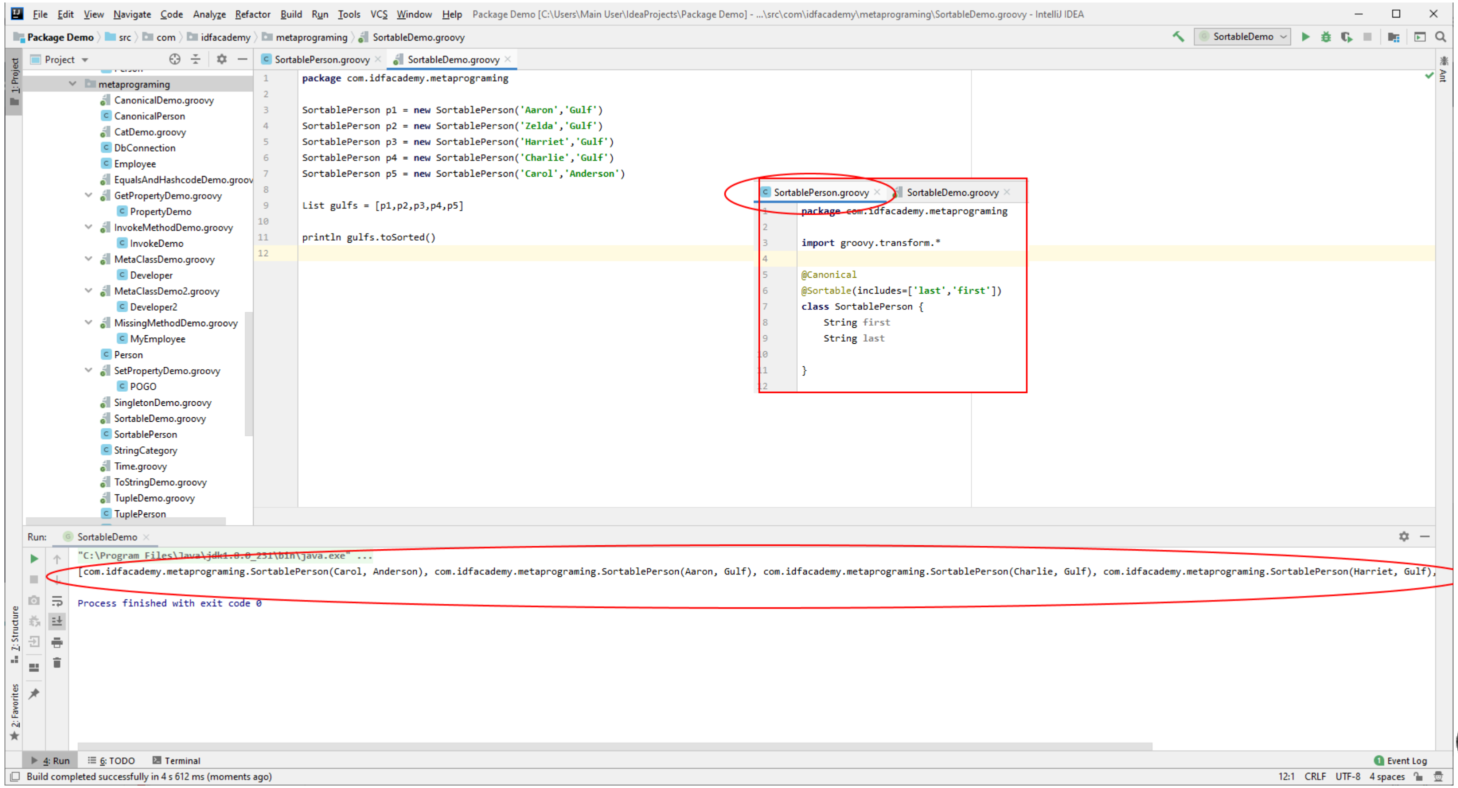
- Canonical Demo

Compile Time Metaprogramming

- Singleton

Compile Time Metaprogramming

- @Sortable



SortableDemo.groovy:

```groovy
package com.idfacademy.metaprograming

SortablePerson p1 = new SortablePerson('Aaron','Gulf')
SortablePerson p2 = new SortablePerson('Zelda','Gulf')
SortablePerson p3 = new SortablePerson('Harriet','Gulf')
SortablePerson p4 = new SortablePerson('Charlie','Gulf')
SortablePerson p5 = new SortablePerson('Carol','Anderson')

List gulfs = [p1,p2,p3,p4,p5]

println gulfs.toSorted()
```

SortablePerson.groovy:

```groovy
package com.idfacademy.metaprograming

import groovy.transform.*

@Canonical
@Sortable(includes=['last','first'])
class SortablePerson {
    String first
    String last

}
```

```
"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe" ...
[com.idfacademy.metaprograming.SortablePerson(Carol, Anderson), com.idfacademy.metaprograming.SortablePerson(Aaron, Gulf), com.idfacademy.metaprograming.SortablePerson(Charlie, Gulf), com.idfacademy.metaprograming.SortablePerson(Harriet, Gulf),

Process finished with exit code 0
```

Build completed successfully in 4 s 612 ms (moments ago)

Compile Time Metaprogramming

- @Immutable

Compile Time Metaprogramming

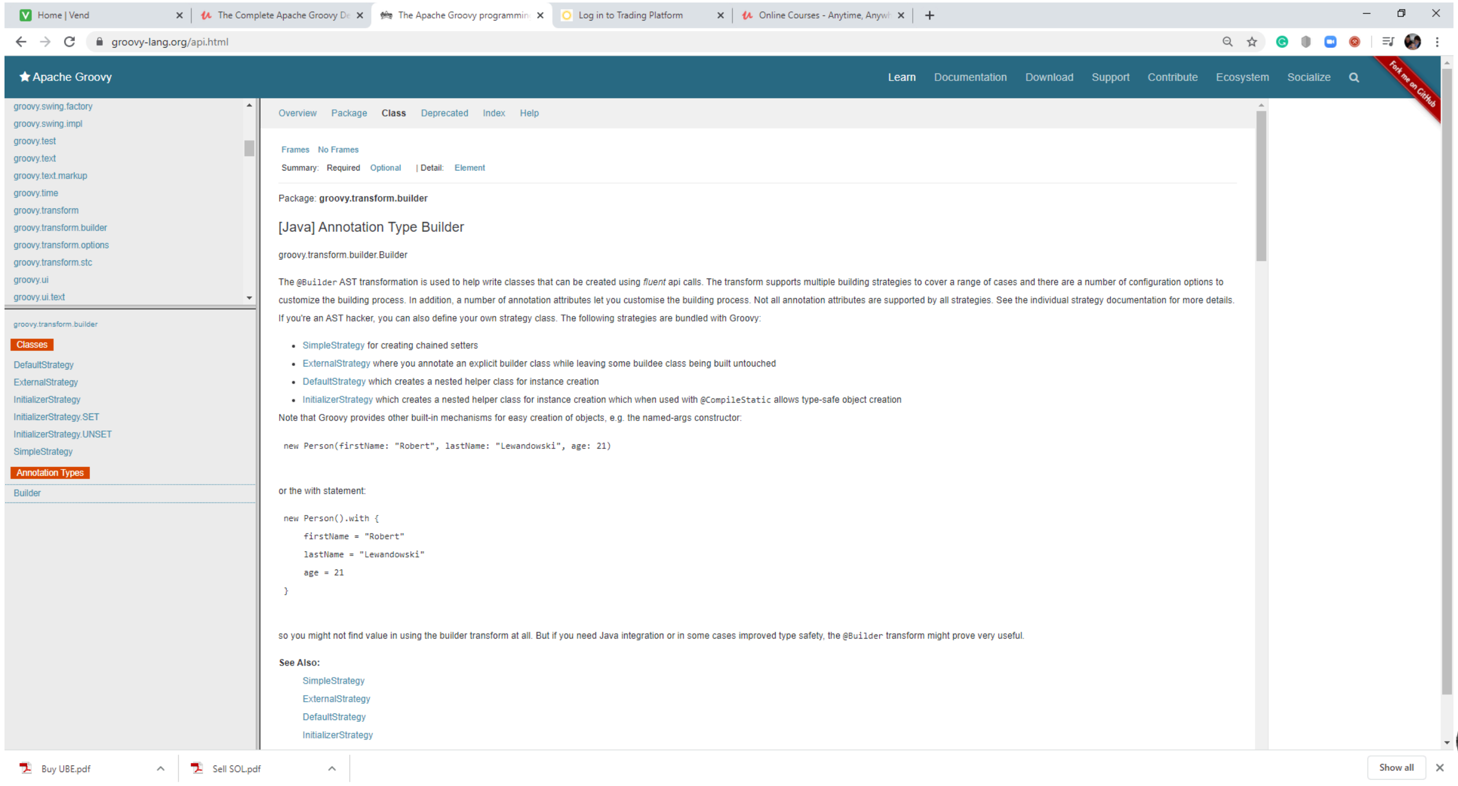
- @TypeChecked

Compile Time Metaprogramming

- @CompileStatic

Compile Time Metaprogramming

- @Builder



Package: **groovy.transform.builder**

[Java] Annotation Type Builder

groovy.transform.builder.Builder

The `@Builder` AST transformation is used to help write classes that can be created using *fluent* api calls. The transform supports multiple building strategies to cover a range of cases and there are a number of configuration options to customize the building process. In addition, a number of annotation attributes let you customise the building process. Not all annotation attributes are supported by all strategies. See the individual strategy documentation for more details. If you're an AST hacker, you can also define your own strategy class. The following strategies are bundled with Groovy:

- SimpleStrategy for creating chained setters
- ExternalStrategy where you annotate an explicit builder class while leaving some buildee class being built untouched
- DefaultStrategy which creates a nested helper class for instance creation
- InitializerStrategy which creates a nested helper class for instance creation which when used with `@CompileStatic` allows type-safe object creation

Note that Groovy provides other built-in mechanisms for easy creation of objects, e.g. the named-args constructor:

```
new Person(firstName: "Robert", lastName: "Lewandowski", age: 21)
```

or the with statement:

```
new Person().with {
    firstName = "Robert"
    lastName = "Lewandowski"
    age = 21
}
```

so you might not find value in using the builder transform at all. But if you need Java integration or in some cases improved type safety, the `@Builder` transform might prove very useful.

See Also:

SimpleStrategy
ExternalStrategy
DefaultStrategy
InitializerStrategy

Compile Time Metaprogramming

- @Builder demo

Exercise

identitrain

Exercise

identitrain

Questions?