

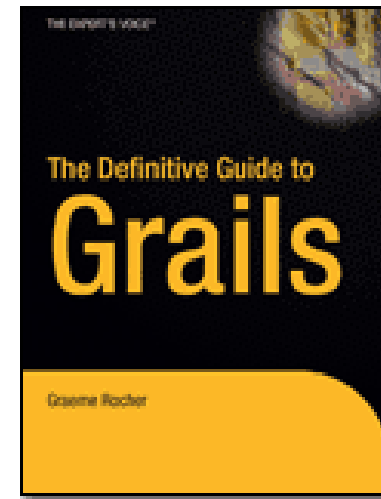
# Introduction To Groovy For Java Programmers



Jeff Brown  
Principal Software Engineer  
Object Computing Inc.  
<http://www.ociweb.com/>  
brown\_j@ociweb.com

# Our Sponsors

- Books To Give Away
- Thanks To Manning And Apress
  - <http://manning.com/>
  - <http://apress.com/>



# About Me

- Jeff Brown
- Principal Engineer - Object Computing Inc.
- Software Engineering For 15 Years
- Mostly Java For 10 Years
- Java/OO Instructor For 7 Years
- Grails Core Development Team Member
- St. Louis Java SIG Steering Committee Member

# What Is Groovy?

- Open Source
- Agile Dynamic Language
  - Others...
    - JavaScript
    - Ruby
    - Python
- Integrates Very Well With Java
  - Runs On The JVM
  - Call Groovy From Java
  - Call Java From Groovy
  - Leverage Powerful Existing Java Libraries

# Why Groovy?

- Familiar Syntax For Java Programmers
- Leverage The Wealth Of Java Libraries
- Easy Integration With Your Existing Infrastructure
  - App Servers
  - Servlet Containers
  - Loads Of Databases With JDBC Drivers
  - All Your Homegrown Java Infrastructure

# Momentum

- In Recent Weeks...
  - Manning published Groovy In Action
  - Apress published The Definitive Guide To Grails
  - AboutGroovy.com went live
  - Groovy 1.0-RC-01 was released

# Installing Groovy

- Download Release
  - <http://groovy.codehaus.org/>
- Extract The Archive
- Set GROOVY\_HOME
- Add \$GROOVY\_HOME/bin to PATH

# Hello Groovy

- Give It A Spin...

```
$ groovy -version  
Groovy Version: 1.0-RC-01 JVM: 1.4.2_13-b06
```

```
$ groovy -e " println 'Hello From Groovy' "  
Hello From Groovy
```

```
$ groovy -e "a=10;b=4;c=a*b;println c"  
40
```

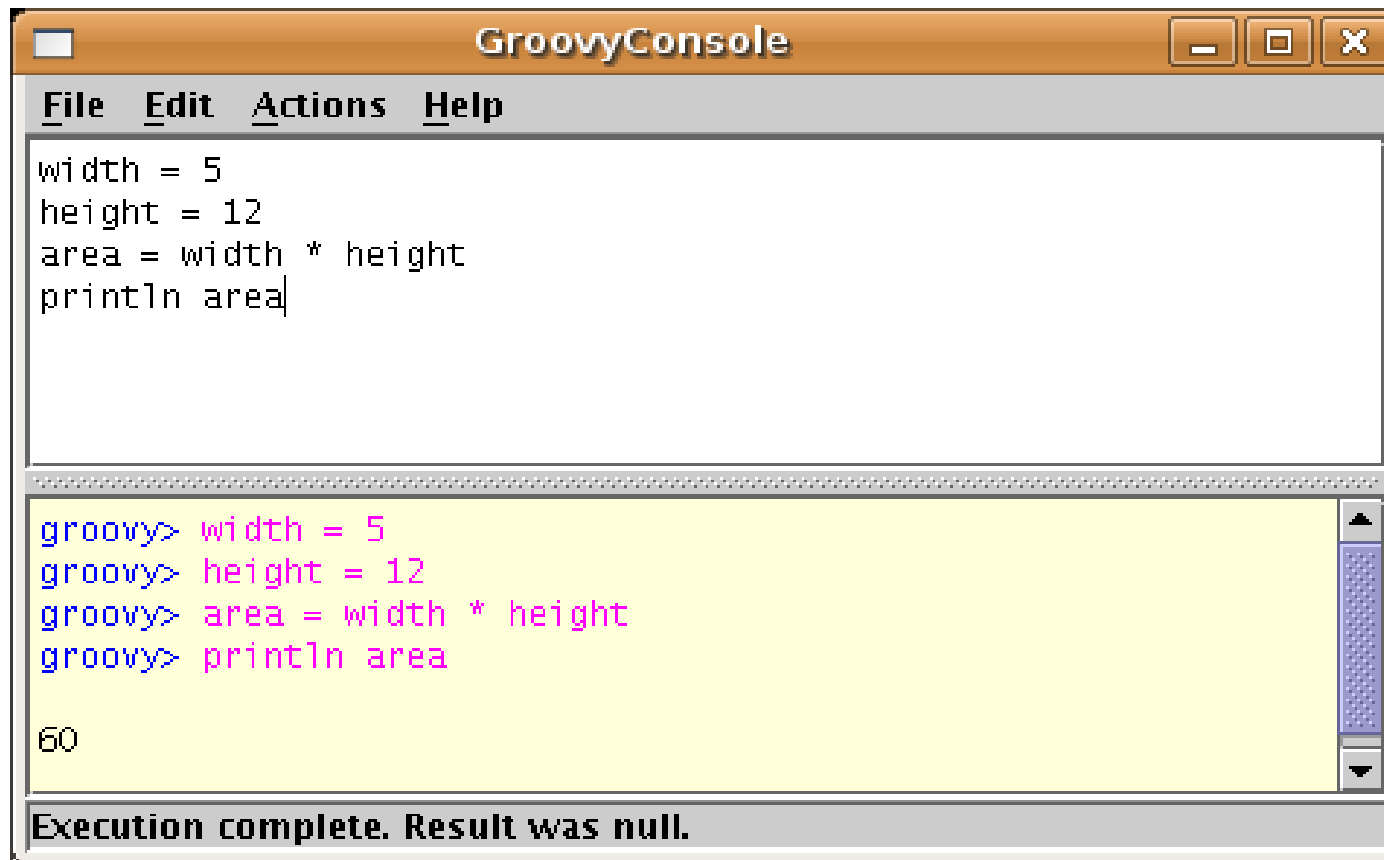
# Groovy Shell

```
$ groovysh
Let's get Groovy!
=====
Version: 1.0-RC-01 JVM: 1.4.2_13-b06
Type 'exit' to terminate the shell
Type 'help' for command help
Type 'go' to execute the statements

groovy> width = 5
groovy> height = 12
groovy> area = width * height
groovy> println area
groovy> go
60
```

# Groovy Console

\$ groovyConsole



The screenshot shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main text area contains the following Groovy code:

```
width = 5
height = 12
area = width * height
println area
```

Below the code, the execution output is displayed in a yellow background:

```
groovy> width = 5
groovy> height = 12
groovy> area = width * height
groovy> println area

60
```

At the bottom of the window, a status bar indicates: "Execution complete. Result was null."

# Groovy Scripts

```
// mygroovyscript.groovy  
println 'Hello From My Groovy Script'
```

```
groovy mygroovyscript.groovy  
Hello From My Groovy Script
```

# Groovy Classes

```
// MyGroovyTest.groovy
class MyGroovyTest {
    def sayHello() {
        println 'Hello From MyGroovyTest'
    }

    static void main(args) {
        def mgt = new MyGroovyTest()
        mgt.sayHello()
    }
}
```

```
groovy MyGroovyTest.groovy
Hello From MyGroovyTest
```

# groovyc

- groovyc Compiles Groovy To Bytecode
- Compiled Code Runs As Normal Java Code
- CLASSPATH
  - groovy-all-[version].jar
  - in \$GROOVY\_HOME/embeddable/

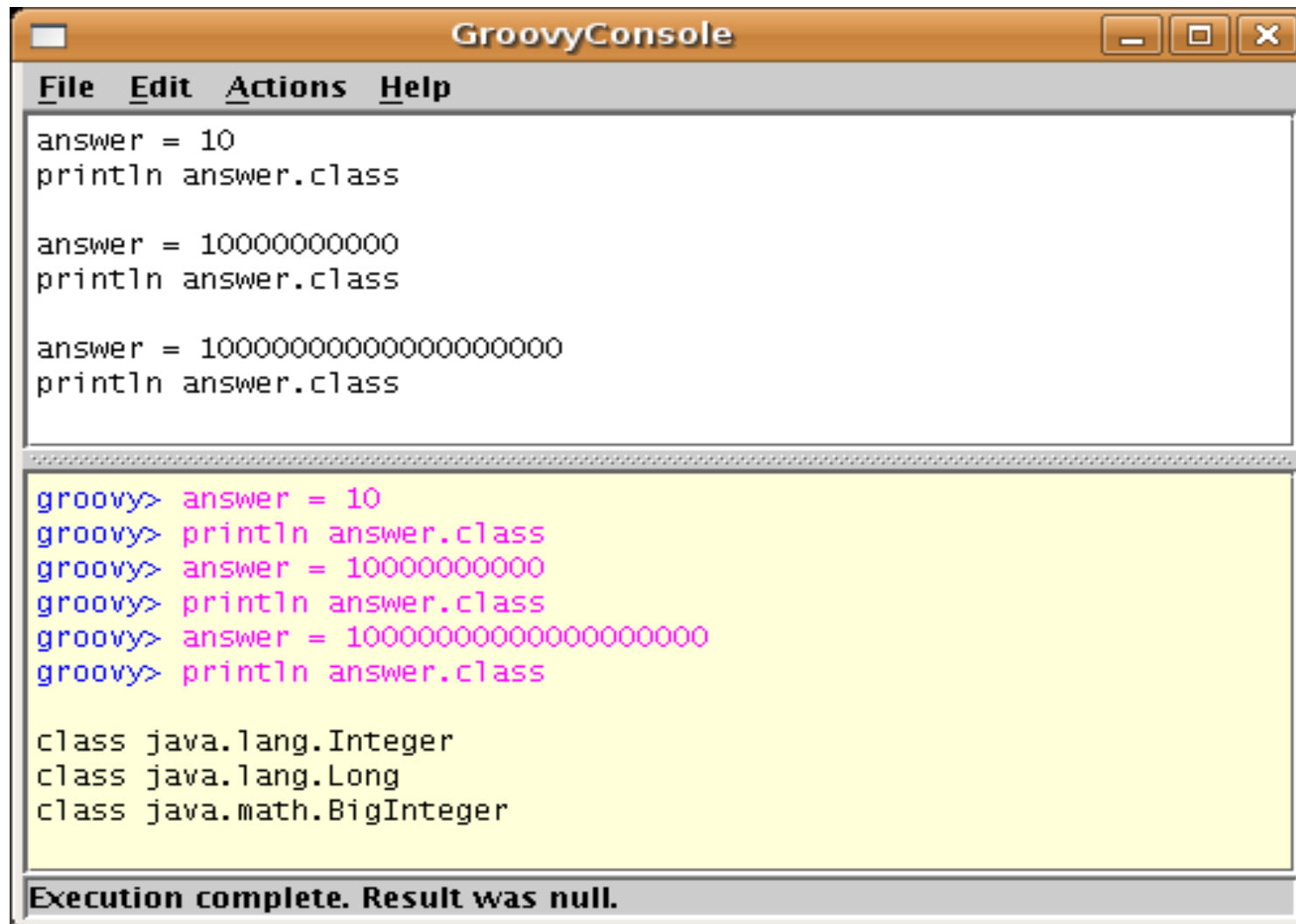
```
groovyc MyGroovyTest.groovy
java MyGroovyTest
Hello From MyGroovyTest
```

# Some Language Basics

- Everything Is An Object
- GString
- Closures
- Collections
- Categories
- Ranges
- Groovy Beans
- Builders
- Meta Programming

The following slides are a bunch of code snippets run in groovysh and groovyConsole to accompany live demo and discussion.

# Everything Is An Object



The screenshot shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main area is divided into two sections. The top section contains three lines of Groovy code: `answer = 10`, `println answer.class`, `answer = 10000000000`, `println answer.class`, `answer = 1000000000000000000000`, and `println answer.class`. The bottom section, which has a yellow background, shows the execution results: `groovy> answer = 10`, `groovy> println answer.class`, `groovy> answer = 10000000000`, `groovy> println answer.class`, `groovy> answer = 1000000000000000000000`, and `groovy> println answer.class`. Below this, the output shows the class names: `class java.lang.Integer`, `class java.lang.Long`, and `class java.math.BigInteger`. At the bottom of the window, a status bar reads "Execution complete. Result was null."

```
answer = 10
println answer.class

answer = 10000000000
println answer.class

answer = 1000000000000000000000
println answer.class

groovy> answer = 10
groovy> println answer.class
groovy> answer = 10000000000
groovy> println answer.class
groovy> answer = 1000000000000000000000
groovy> println answer.class

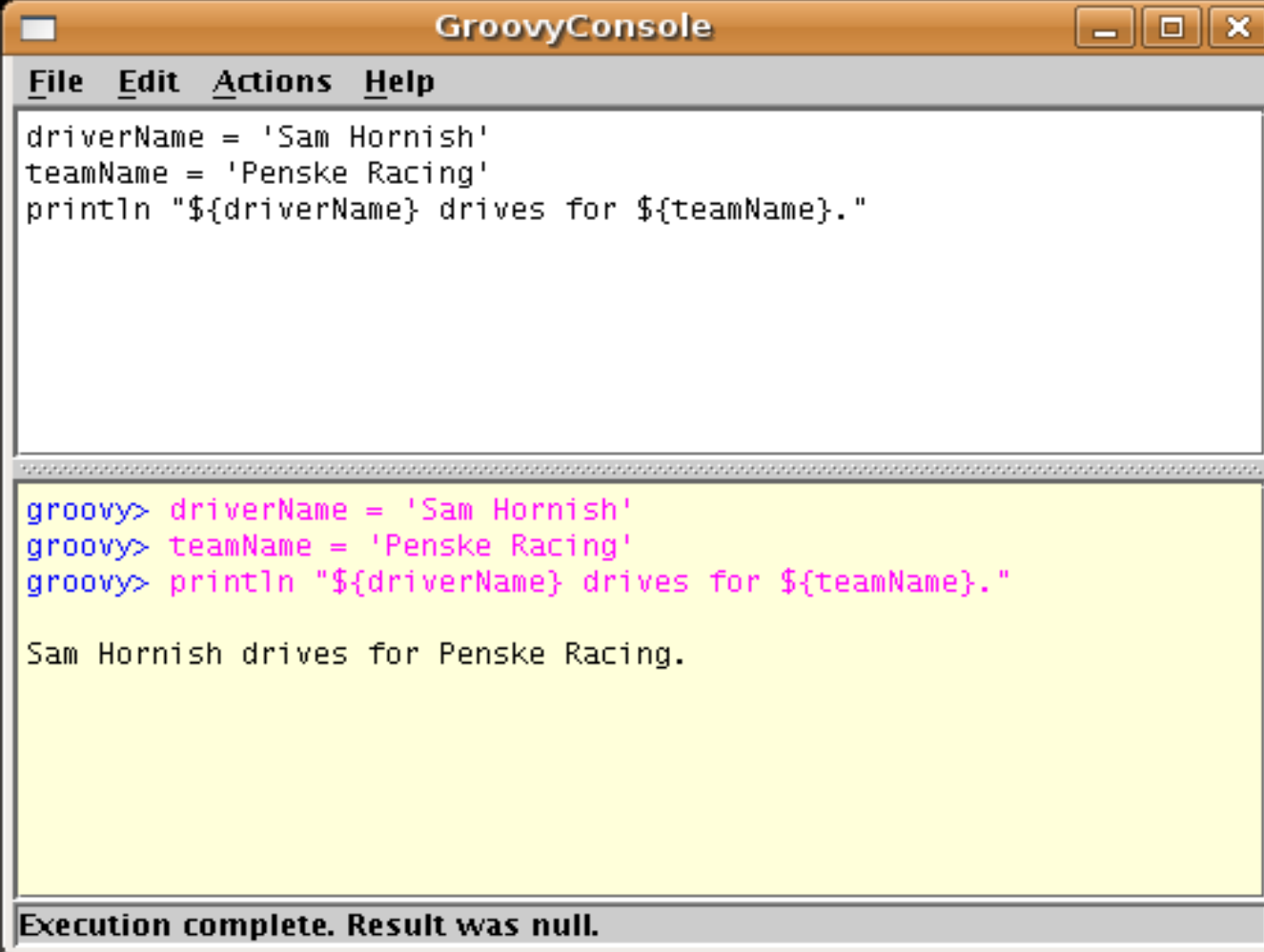
class java.lang.Integer
class java.lang.Long
class java.math.BigInteger

Execution complete. Result was null.
```

# Groovy Strings

- Known As GStrings
- GStrings Are Surrounded By Double Quotes
  - single quotes are used for regular strings
- May Contain Groovy Expressions
  - expressions surrounded by `${ }`
  - evaluated and substitution takes place at runtime
- Square Bracket Syntax May Be Applied
  - like `charAt(i)`
  - more complicated range related example later

# Groovy Strings



The image shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main area is divided into two sections. The top section contains Groovy code: `driverName = 'Sam Hornish'`, `teamName = 'Penske Racing'`, and `println "${driverName} drives for ${teamName}."`. The bottom section, which has a yellow background, shows the execution of this code: `groovy> driverName = 'Sam Hornish'`, `groovy> teamName = 'Penske Racing'`, `groovy> println "${driverName} drives for ${teamName}."`, followed by the output `Sam Hornish drives for Penske Racing.`. At the bottom of the window, a status bar reads "Execution complete. Result was null."

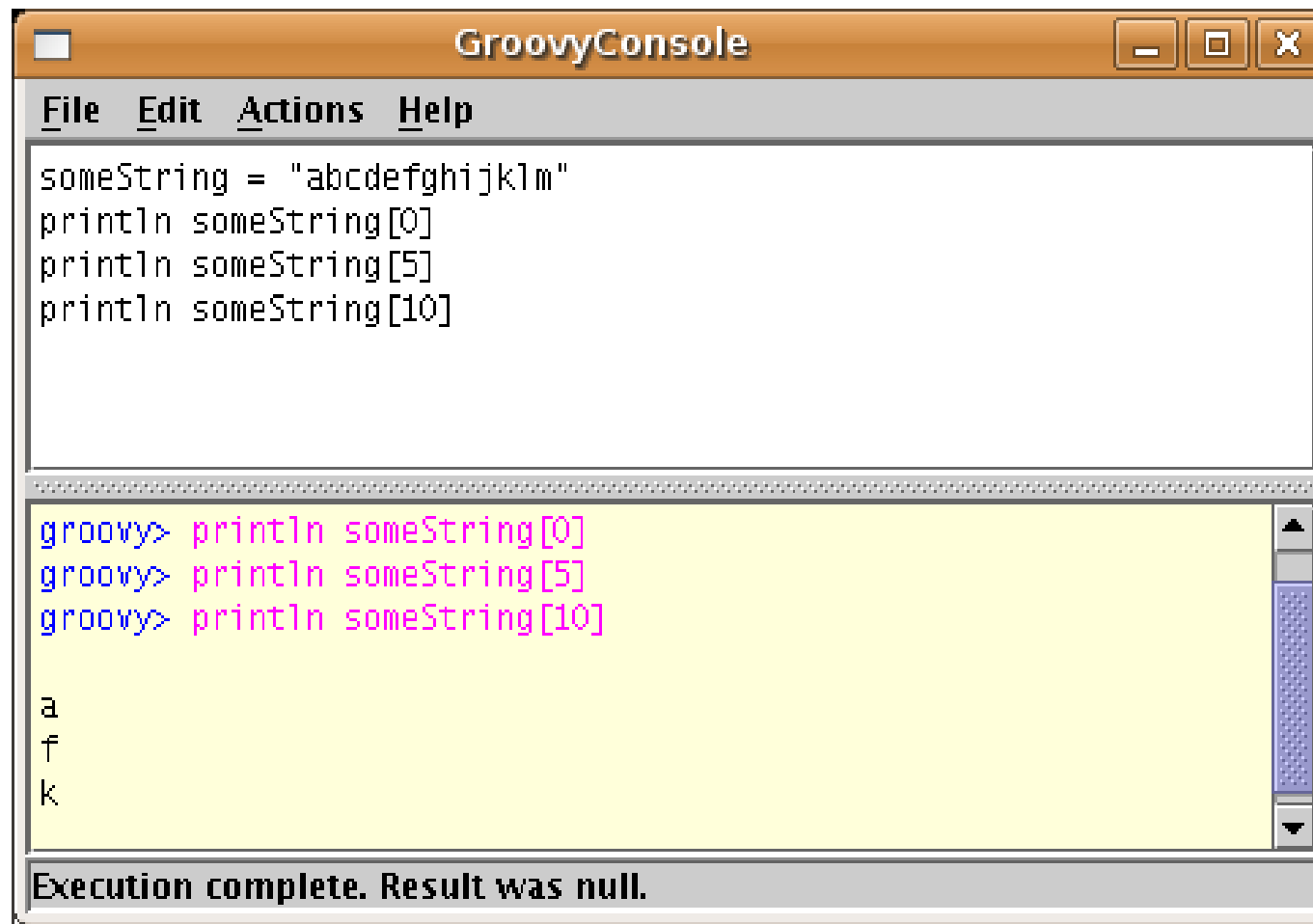
```
driverName = 'Sam Hornish'
teamName = 'Penske Racing'
println "${driverName} drives for ${teamName}."
```

```
groovy> driverName = 'Sam Hornish'
groovy> teamName = 'Penske Racing'
groovy> println "${driverName} drives for ${teamName}."
```

Sam Hornish drives for Penske Racing.

Execution complete. Result was null.

# Groovy Strings



The screenshot shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main text area contains the following Groovy code:

```
someString = "abcdefghijklm"  
println someString[0]  
println someString[5]  
println someString[10]
```

Below the code, the execution output is shown on a yellow background:

```
groovy> println someString[0]  
groovy> println someString[5]  
groovy> println someString[10]  
  
a  
f  
k
```

At the bottom of the window, a status bar indicates: "Execution complete. Result was null."

# Closures

- A Block Of Code
- May Be Passed Around
- May Accept Arguments
- Always Return Something
  - not necessarily explicitly
- More Flexible Than Anonymous Inner Class

# Closures

- Groovy Adds A times Method to Number
- The times Method Accepts A Closure

```
groovy> 3.times { println 'Hello' }  
groovy> go  
Hello  
Hello  
Hello
```

# Closures

- Closures Are First Class Objects
- References May Point To Closures

```
groovy> cl = { println 'Closures Are Cool' }  
groovy> 3.times cl  
groovy> go  
Closures Are Cool  
Closures Are Cool  
Closures Are Cool
```

# Closures

- Closures May Accept Arguments
- The times Method Passes An Argument To The Closure

```
groovy> 3.times { index -> println "index is ${index}" }
groovy> go
index is 0
index is 1
index is 2
```

# Closures

- Closures Have An Implicit “it” Argument

```
groovy> 3.times { println "index is ${it}" }
groovy> go
index is 0
index is 1
index is 2
```

# Closures

- Multiple Arguments

```
groovy> myMap = [name:'Jeff', location:'St. Louis']
groovy> myMap.each { key, value ->
    println "${key} is ${value}"
}
groovy> go
location is St. Louis
name is Jeff
```

# Collections

- Lists Are Simple To Declare

```
groovy> kids = ['Zack', 'Jake']  
groovy> println kids.class  
groovy> go  
class java.util.ArrayList
```

# Collections

- Adding To A List

```
groovy> albums = ['Rush']
groovy> albums << 'Fly By Night'
groovy> albums += 'Caress Of Steel'
groovy> albums.add '2112'
groovy> println albums
groovy> go
["Rush", "Fly By Night", "Caress Of Steel", "2112"]
```

# Collections

- Removing From A List

```
groovy> instruments =  
    ['drums', 'keyboards', 'guitars', 'lutes']  
groovy> instruments.remove 'keyboards'  
groovy> instruments -= 'lutes'  
groovy> println instruments  
groovy> go  
["drums", "guitars"]
```

# Collections

- Closure To Iterate Over A List

```
groovy> tracks = ['Custard Pie',  
                 'The Rover',  
                 'In My Time Of Dying']  
groovy> tracks.each { println it }  
groovy> go  
Custard Pie  
The Rover  
In My Time Of Dying
```

# Collections

- Iterating With An Index

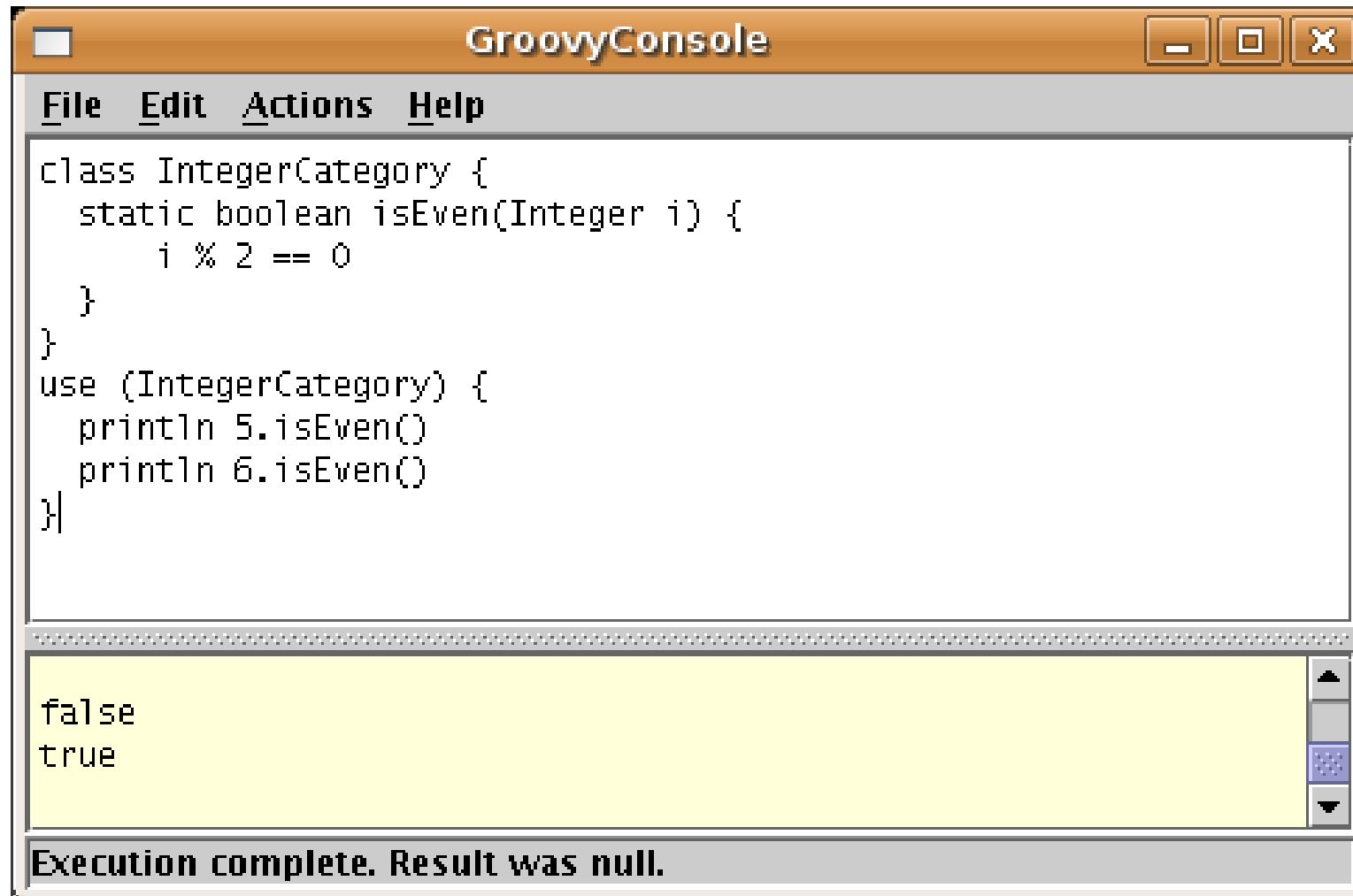
```
groovy> tracks = ['Custard Pie',  
                 'The Rover',  
                 'In My Time Of Dying']  
groovy> tracks.eachWithIndex { track, index ->  
    println "Track ${index + 1}: ${track}"  
    }  
groovy> go  
Track 1: Custard Pie  
Track 2: The Rover  
Track 3: In My Time Of Dying
```

# Collections

- Map Manipulation

```
groovy> myMap = [bass:'Geezer', drums:'Bill']
groovy> myMap['vocals'] = 'Ozzy'
groovy> myMap.guitar = 'Tony'
groovy> println myMap
groovy> go
["drums":"Bill", "vocals":"Ozzy",
 "bass":"Geezer", "guitar":"Tony"]
```

# Groovy Categories



The screenshot shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main text area contains the following Groovy code:

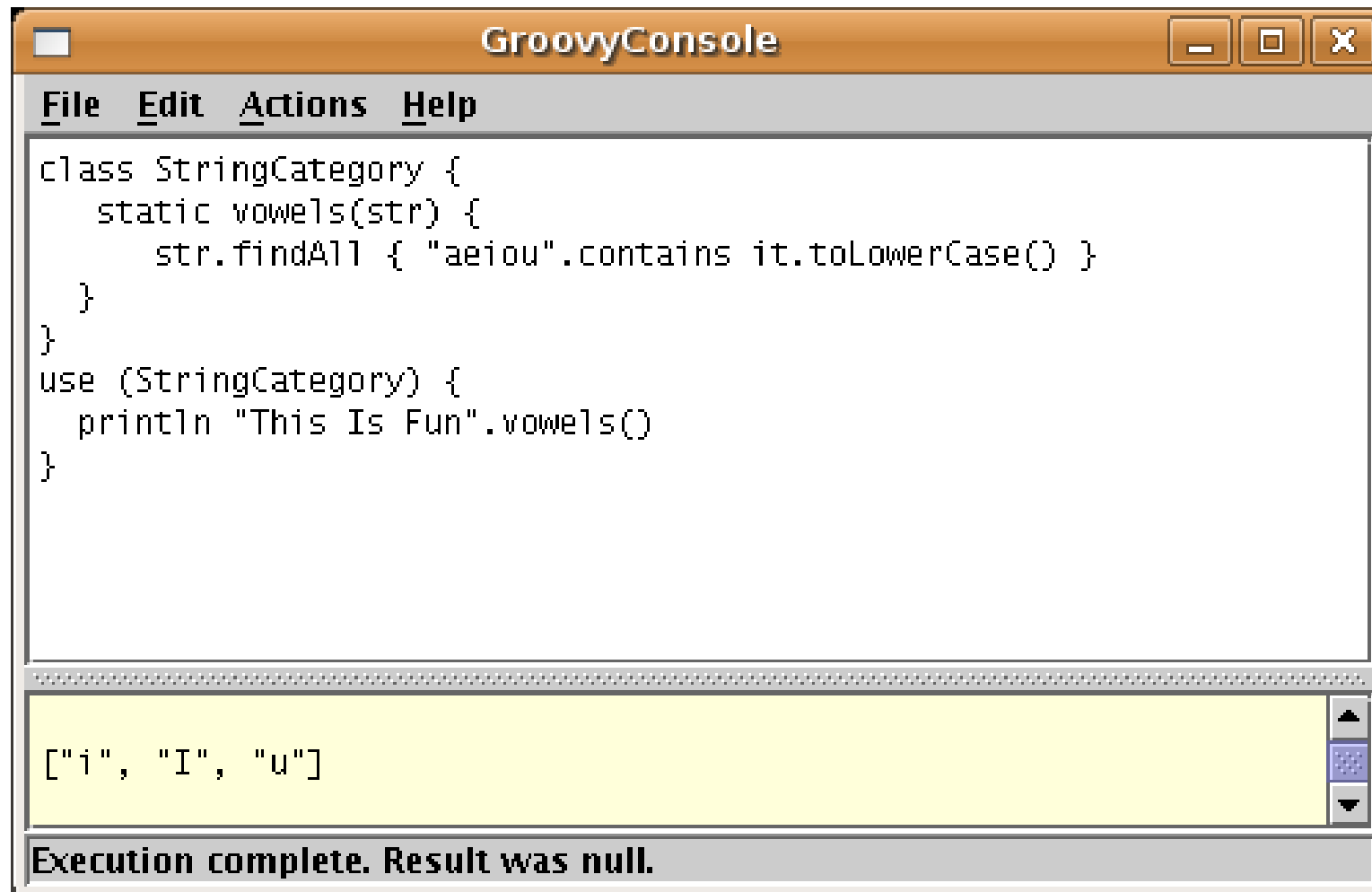
```
class IntegerCategory {  
    static boolean isEven(Integer i) {  
        i % 2 == 0  
    }  
}  
use (IntegerCategory) {  
    println 5.isEven()  
    println 6.isEven()  
}
```

Below the code, the output of the execution is displayed on a yellow background:

```
false  
true
```

At the bottom of the window, a status bar indicates: "Execution complete. Result was null."

# Groovy Categories

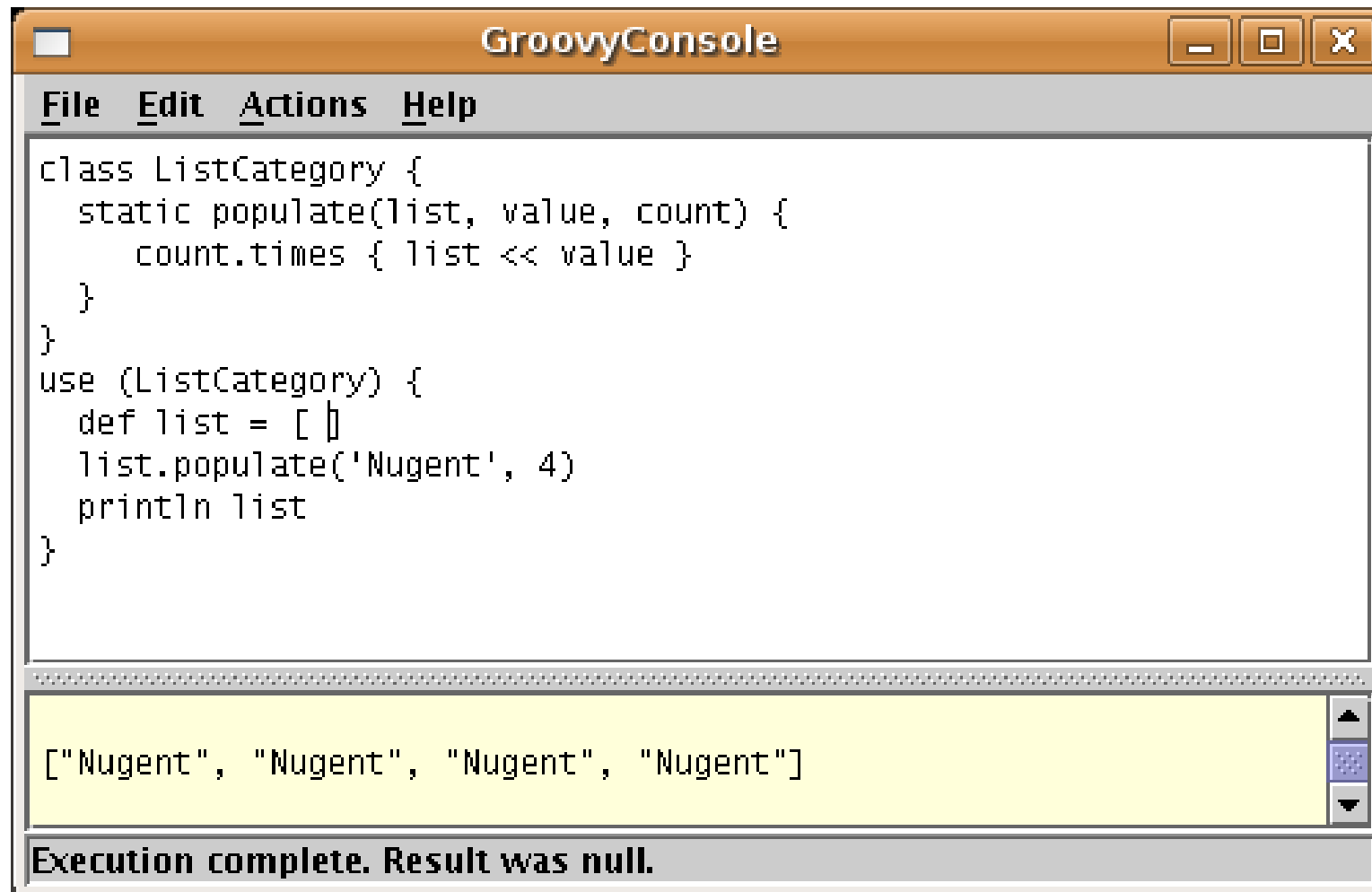


The screenshot shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main text area contains the following Groovy code:

```
class StringCategory {  
    static vowels(str) {  
        str.findAll { "aeiou".contains it.toLowerCase() }  
    }  
}  
use (StringCategory) {  
    println "This Is Fun".vowels()  
}
```

Below the code, the output is displayed in a yellow-highlighted area: ["i", "I", "u"]. At the bottom of the window, a status bar indicates "Execution complete. Result was null."

# Groovy Categories



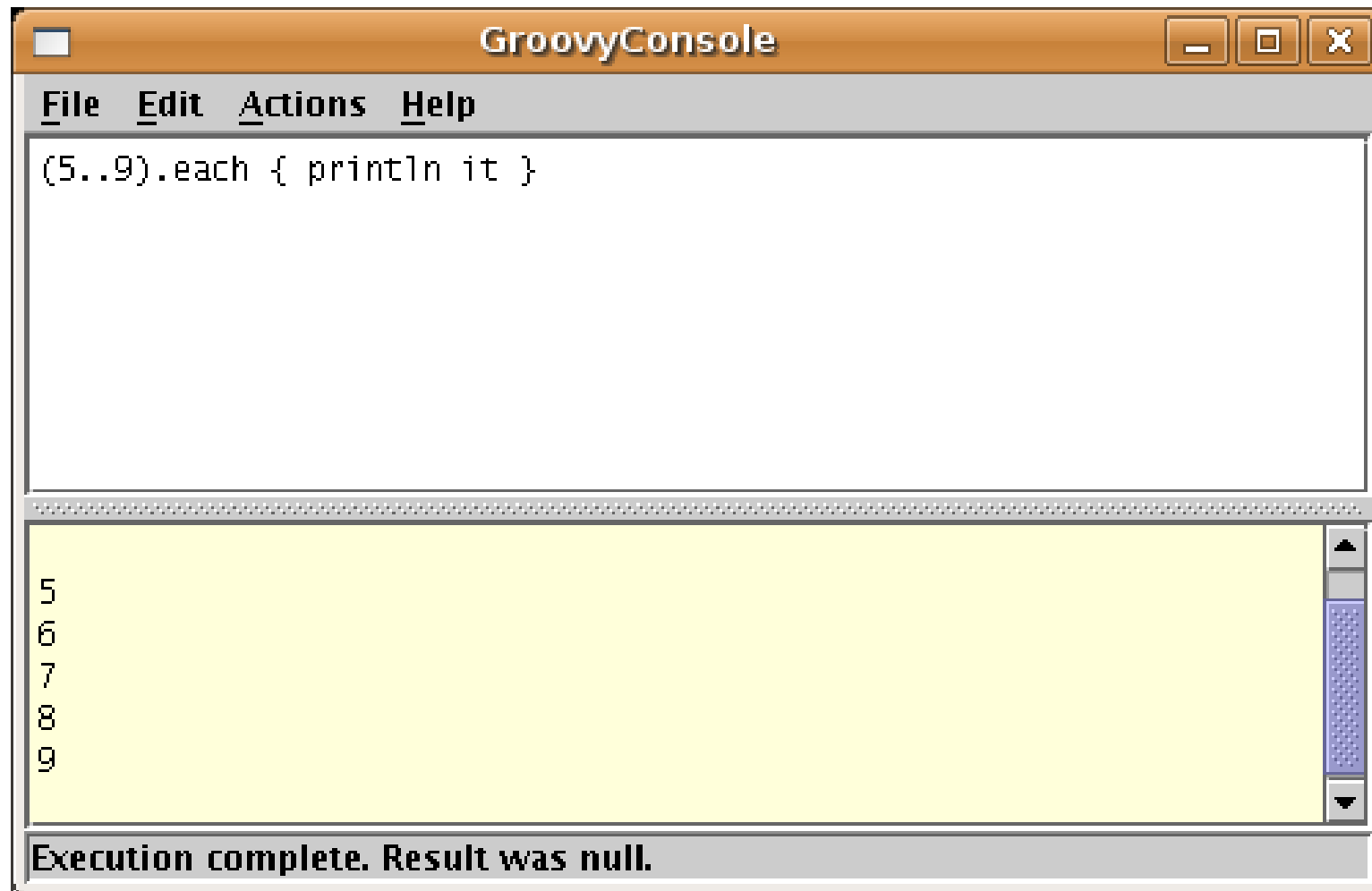
The image shows a screenshot of a window titled "GroovyConsole". The window has a menu bar with "File", "Edit", "Actions", and "Help". The main area contains Groovy code defining a class and using it. Below the code is a yellow output area showing the result of the execution, and a status bar at the bottom.

```
class ListCategory {
    static populate(list, value, count) {
        count.times { list << value }
    }
}
use (ListCategory) {
    def list = []
    list.populate('Nugent', 4)
    println list
}
```

["Nugent", "Nugent", "Nugent", "Nugent"]

Execution complete. Result was null.

# Ranges



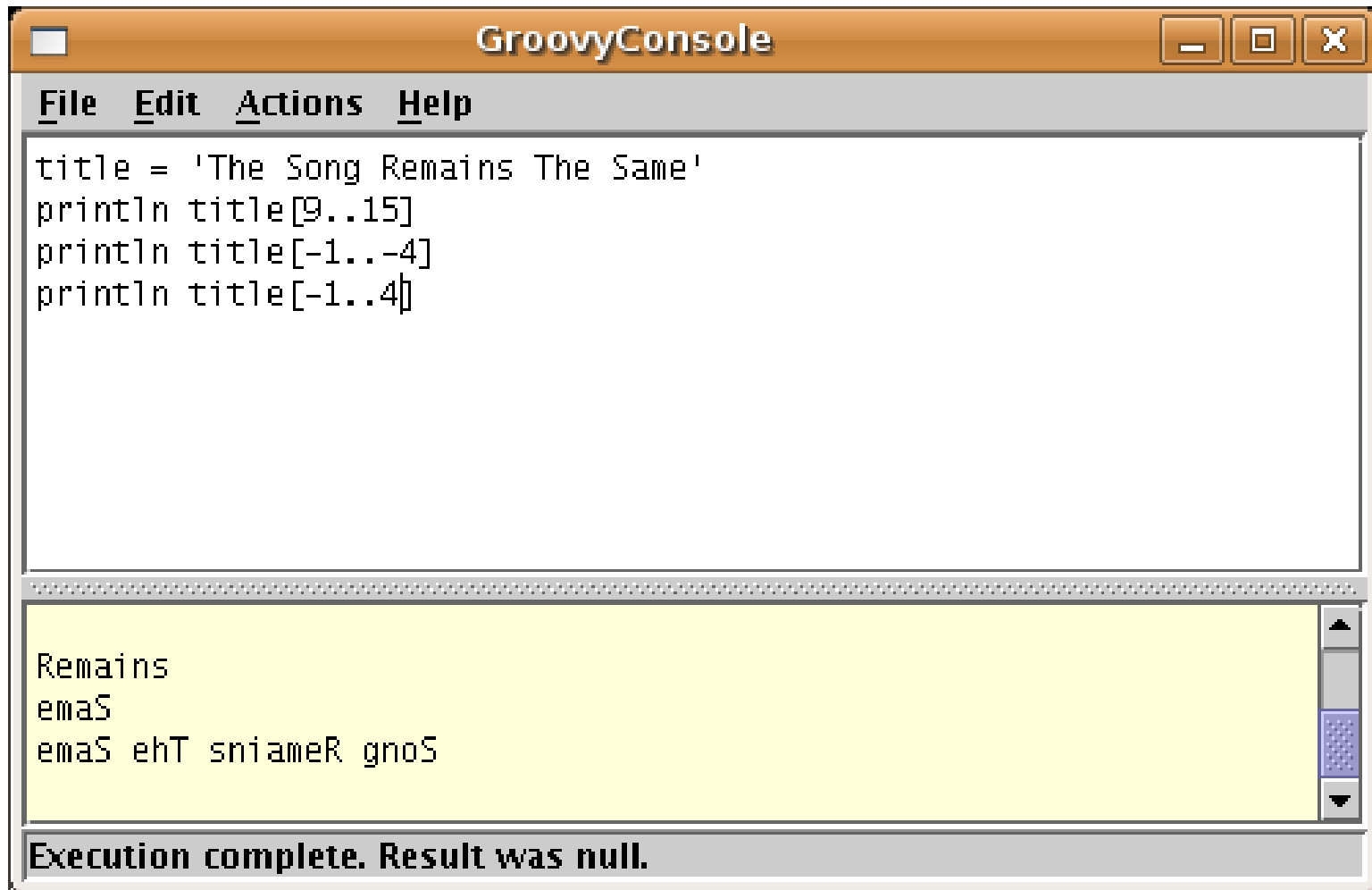
The image shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main text area contains the Groovy code `(5..9).each { println it }`. Below the code is a yellow output area displaying the numbers 5, 6, 7, 8, and 9 on separate lines. At the bottom of the window, a status bar reads "Execution complete. Result was null."

```
File Edit Actions Help
(5..9).each { println it }

5
6
7
8
9

Execution complete. Result was null.
```

# Ranges



The screenshot shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main text area contains the following Groovy code:

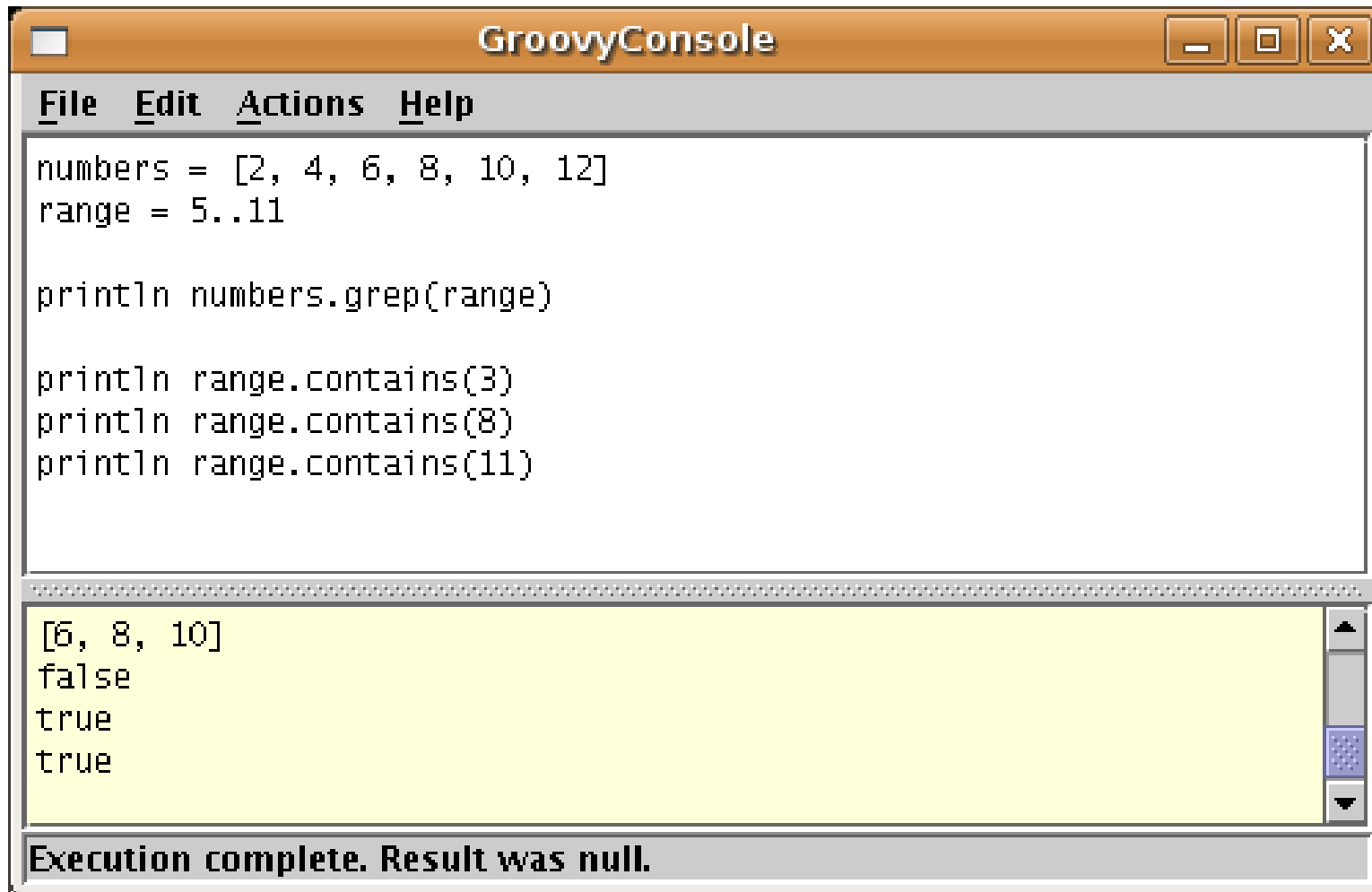
```
title = 'The Song Remains The Same'  
println title[9..15]  
println title[-1..-4]  
println title[-1..4]
```

The output area, which has a yellow background, displays the results of the code execution:

```
Remains  
emaS  
emaS ehT sniameR gnoS
```

At the bottom of the window, a status bar indicates: "Execution complete. Result was null."

# Ranges



The screenshot shows a window titled "GroovyConsole" with a menu bar containing "File", "Edit", "Actions", and "Help". The main text area contains the following Groovy code:

```
numbers = [2, 4, 6, 8, 10, 12]
range = 5..11

println numbers.grep(range)

println range.contains(3)
println range.contains(8)
println range.contains(11)
```

The output area, which has a yellow background, displays the following results:

```
[6, 8, 10]
false
true
true
```

At the bottom of the window, a status bar indicates: "Execution complete. Result was null."

# Groovy Beans

- Groovy Beans Are POGOs
- Similar To POJOs
- Boilerplate Code Is Eliminated

# POJO

```
// Person.java
public class Person {

    private String firstName;
    private String lastName;

    public Person() {}

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

# POJO

```
public String getFirstName() {  
    return firstName;  
}  
  
public void setFirstName(String firstName) {  
    this.firstName = firstName;  
}  
  
public String getLastName() {  
    return lastName;  
}  
  
public void setLastName(String lastName) {  
    this.lastName = lastName;  
}  
}
```

# POJO

- I Wrote Code To Declare The Fields
- I Let Eclipse Generate Constructors
- I Let Eclipse Generate Getters/Setters

If all of that code can be generated by the IDE, why can't it be generated by the compiler or the runtime environment?

# Groovy Beans

```
// BaseballTeam.groovy
class BaseballTeam {
    def cityName
    def teamName
}
```

- No Need To Write Constructors
- No Need To Write Getters/Setters
- May Declare Types – Don't Need To

# Groovy Beans

```
cardinals = new BaseballTeam(cityName:'St. Louis',  
                             teamName:'Cardinals')  
println "City Name: ${cardinals.cityName}"  
println "Team Name: ${cardinals.teamName}"
```

```
City Name: St. Louis  
Team Name: Cardinals
```

# Groovy Beans

- Property Access Looks Like Field Access
  - name = cardinals.teamName
  - name = cardinals.getTeamName()
- Assignment Works The Same Way
  - cardinals.teamName = 'Saint Louis'
  - cardinals.setTeamName('Saint Louis')

# Groovy Beans

- Properties Aren't Necessarily Declared As Fields

```
class BaseballTeam {  
  
    def cityName  
    def teamName  
  
    def getDisplayName () {  
        "${cityName} ${teamName}"  
    }  
  
}
```

# Groovy Beans

```
cardinals = new BaseballTeam(cityName:'St. Louis',  
                              teamName:'Cardinals')  
println cardinals.displayName
```

St. Louis Cardinals

# Groovy Beans

- Properties Are Public By Default
  - private field
  - public getter/setter
- Properties May Be Private Or Protected
- No 'package' Level

# MarkupBuilder

```
builder = new groovy.xml.MarkupBuilder()  
builder.baseball {  
    league(name:"National") {  
        team("Cardinals")  
        team("Cubs")  
        team("Mets")  
    }  
    league(name:"American") {  
        team("Angels")  
        team("Yankees")  
        team("Royals")  
    }  
}
```

# MarkupBuilder

```
<baseball>
  <league name='National'>
    <team>Cardinals</team>
    <team>Cubs</team>
    <team>Mets</team>
  </league>
  <league name='American'>
    <team>Angels</team>
    <team>Yankees</team>
    <team>Royals</team>
  </league>
</baseball>
```

# MarkupBuilder

```
builder = new groovy.xml.MarkupBuilder()
builder.html() {
  head() {
    title('Markup Builder Demo')
  }
  body {
    h1('Bands')
    ul {
      li('Rush')
      li('King Crimson')
      li('Opeth')
    }
  }
}
```

# MarkupBuilder

```
<html>
  <head>
    <title>Markup Builder Demo</title>
  </head>
  <body>
    <h1>Bands</h1>
    <ul>
      <li>Rush</li>
      <li>King Crimson</li>
      <li>Opeth</li>
    </ul>
  </body>
</html>
```

# Meta Programming

```
class MyGroovyThing {  
    Object invokeMethod(String name, Object args) {  
        // do something cool  
    }  
}
```

Live Meta Programming Demo...

# Calling Groovy From Java

- GroovyShell
- GroovyClassLoader
- Bean Scripting Framework
- Java 6 (and beyond...)

Live Demo...

# Links

- Main Groovy Site
  - <http://groovy.codehaus.org/>
- Main Grails Site
  - <http://grails.org/>
- Groovy Portal
  - <http://aboutgroovy.com/>
- My Blog
  - <http://javajeff.blogspot.com/>
- Java News Brief
  - <http://www.ocinweb.com/jnb/>

# The End

Thank You For Coming!

Q&A