

---

# Essential Groovy



# Contents

1. Groovy essentials
2. From Java to Groovy



Demo project:  
**Demo-GettingStartedwithGroovy**

# 1. Groovy Essentials

- What is Groovy?
- Groovy features
- Creating Groovy applications using Eclipse
- Creating a Groovy project
- Adding a class
- Building and running the class

# What is Groovy?

- Groovy is an agile and dynamic language for the Java Virtual Machine
- Builds upon the strengths of Java
  - Also has additional power features inspired by languages such as Python, Ruby, and Smalltalk
  - Makes modern programming features available to Java developers with almost-zero learning curve
- Supports Domain Specific Languages and other compact syntax
  - So your code is a lot easier to read and maintain

# Groovy Features

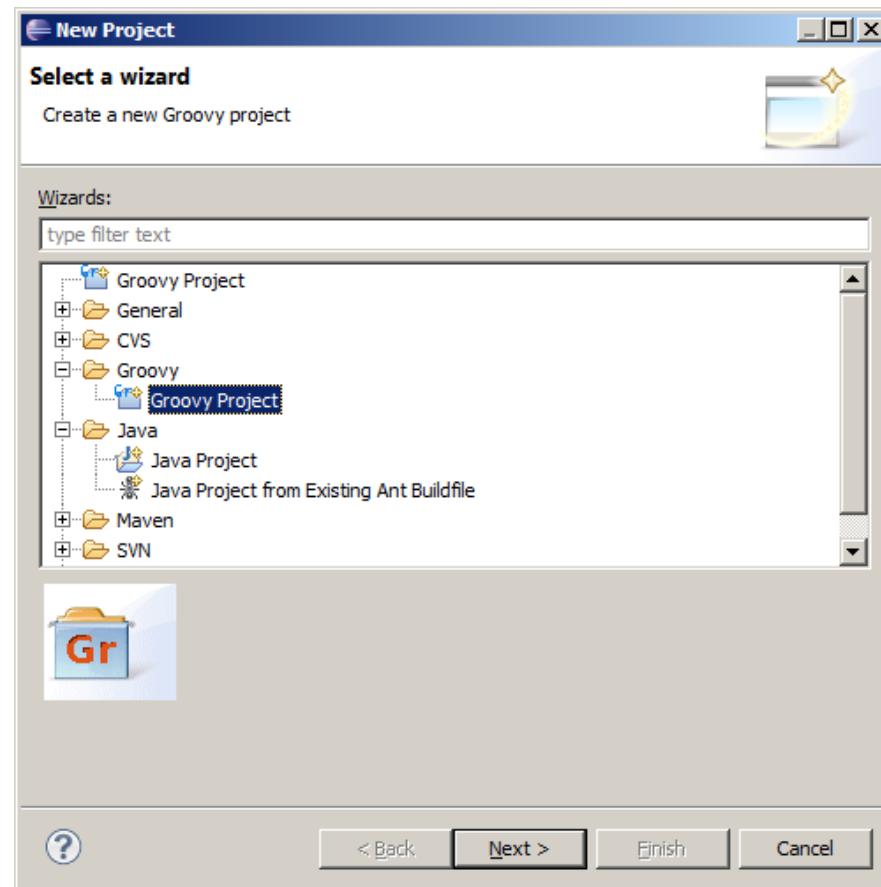
- Increases developer productivity
  - Reduces scaffolding code when developing UI, web, database, XML, and console apps
- Simplifies testing
  - Supports unit testing and mocking out-of-the-box
- Compiles straight to Java byte code
  - You can use it anywhere you can use Java
  - Integrates seamlessly with all existing Java objects and libraries

# Creating Groovy Applications using Eclipse

- There's an Eclipse plug-in for Groovy
- In Eclipse, invoke Help | Install New Software and add the following update site (for Juno):
  - <http://dist.springsource.org/release/GRECLIPSE/e4.2/>
- Choose which components to install (e.g. all of them!)
  - Then follow the wizard steps to install the Groovy plugin
- Then restart Eclipse for the changes to take effect

# Creating a Groovy Project (1 of 2)

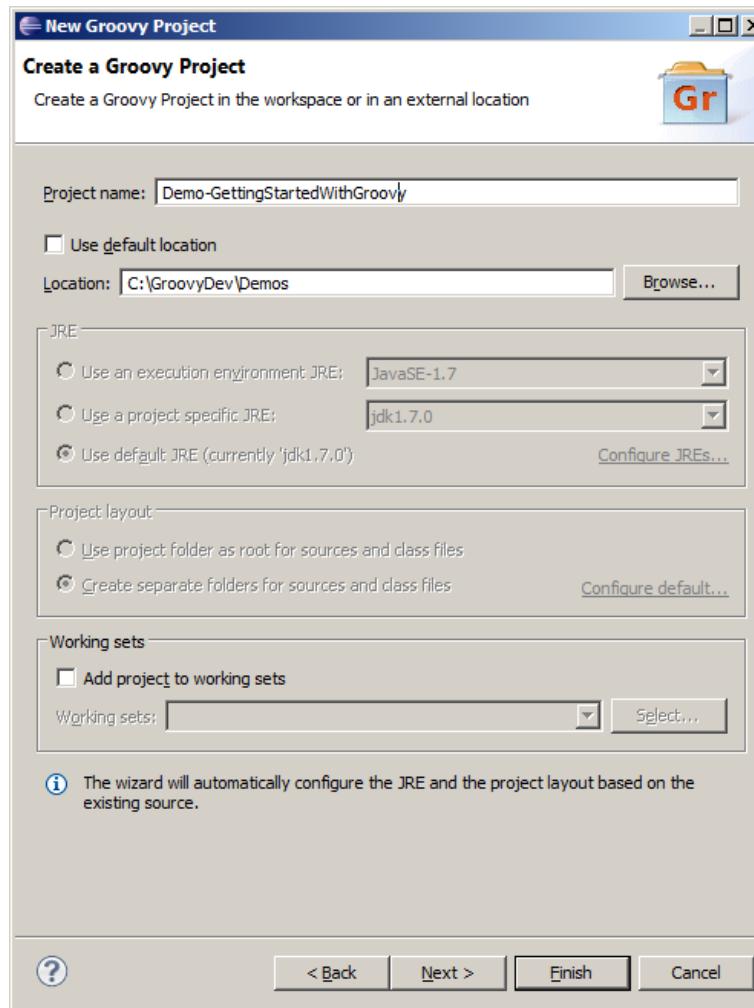
- In Eclipse, invoke File | New | Project
  - Choose Groovy Project



- Then click Next

# Creating a Groovy Project (2 of 2)

- Enter a project name and location, and then click Finish



# Adding a Class

- Right-click the `src` folder, and invoke `New | Groovy Class`
  - Specify a package name and class name
- Here's the skeleton code
  - Similar to Java, but simpler!

```
package mypackage

class MyClass {

    static main(args) {

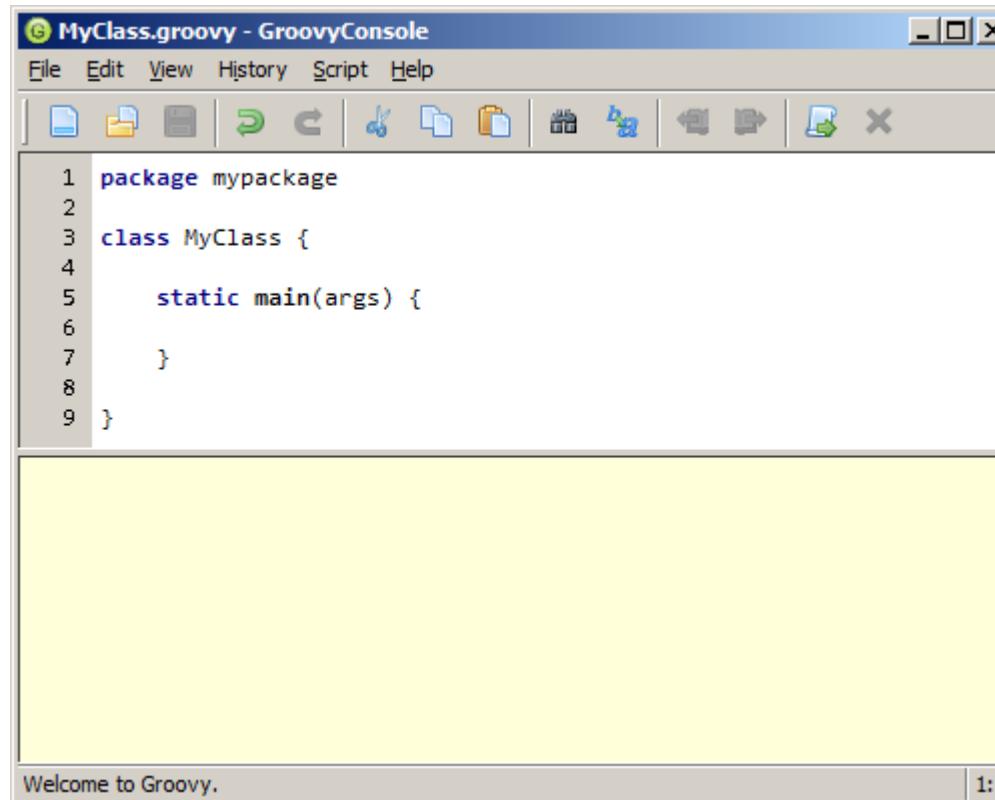
    }

}
```

`MyClass.groovy`

# Building and Running the Class

- To build and run the class (in the JVM!):
  - Right-click the class, and invoke Run as | Groovy Console
  - Compiles the class to Java byte codes
  - Runs in the Groovy Console (select Script | Run)



The image shows a screenshot of the GroovyConsole application window. The title bar reads "MyClass.groovy - GroovyConsole". The menu bar includes File, Edit, View, History, Script, and Help. The toolbar contains various icons for file operations like Open, Save, and Run. The code editor pane displays the following Groovy script:

```
1 package mypackage
2
3 class MyClass {
4
5     static main(args) {
6
7     }
8
9 }
```

The bottom status bar says "Welcome to Groovy." and "1:1".

## 2. From Java to Groovy

- Step 0: vanilla Java
- Step 1: public and semicolons
- Step 2: getters, setters, types, and console IO
- Step 3: dynamic types
- Step 4: variable interpolation
- Step 5: unnecessary keywords
- Step 6: construction and properties
- Step 7: scripting

# Step 0: Vanilla Java

- About 95% of Java syntax is directly portable to Groovy
  - Just rename .java file to .groovy, and you're done!
  - E.g. the following code is legal in both Java and Groovy:

```
package mypackage

public class HelloWorld0 {

    String name;

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

    public String getName() { return name; }

    public String sayHello() { return "Hello " + name; }

    public static void main(String [] args){
        HelloWorld0 obj = new HelloWorld0();
        obj.setName("Groovy v0");
        System.out.println( obj.sayHello() );
    }
}
```

HelloWorld0.groovy

# Step 1: Public and Semicolons

- Everything in Groovy is `public`, unless defined otherwise
- Semicolons at the end of line are optional

```
package mypackage

class HelloWorld1 {

    String name

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

    String getName(){ return name }

    String sayHello() { return "Hello " + name }

    static void main(String [] args) {
        HelloWorld1 obj = new HelloWorld1()
        obj.setName("Groovy v1")
        System.out.println( obj.sayHello() )
    }
}
```

HelloWorld1.groovy

# Step 2: Getters, Setters, Types, Console IO

- Getters and setters are free in Groovy
- Type parameters are optional
- Console output is simplified

```
package mypackage

class HelloWorld2 {

    String name

    String sayHello() { return "Hello " + name }

    static void main(args) {
        HelloWorld2 obj = new HelloWorld2()
        obj.setName("Groovy v2")
        println( obj.sayHello() )
    }
}
```

HelloWorld2.groovy

# Step 3: Dynamic Types

- Groovy has dynamic types
  - Via the def keyword (similar to var in JS and dynamic in C#)
  - Groovy deduces the correct type dynamically

```
package mypackage

class HelloWorld3 {

    String name

    def sayHello() { return "Hello " + name }

    static void main(args) {
        def obj = new HelloWorld3()
        obj.setName("Groovy v3")
        println( obj.sayHello() )
    }
}
```

HelloWorld3.groovy

# Step 4: Variable Interpolation

- Groovy supports variable interpolation through via GStrings(!)
  - Prepend any Groovy expression with \$ inside a string literal

```
package mypackage

class HelloWorld4 {

    String name

    def sayHello() { return "Hello $name, do you earn more than \$1000000?" }

    static def main(args) {
        def obj = new HelloWorld4()
        obj.setName("Groovy v4")
        println( obj.sayHello() )
    }
}
```

HelloWorld4.groovy

# Step 5: Unnecessary Keywords

- The return keyword is optional
  - The return value of a method will be the last evaluated expression
- You don't need to use def in static methods

```
package mypackage

class HelloWorld5 {

    String name

    def sayHello() { "Hello $name, do you earn more than \$1000000?" }

    static main(args) {
        def obj = new HelloWorld5()
        obj.setName("Groovy v5")
        println( obj.sayHello() )
    }
}
```

HelloWorld5.groovy

# Step 6: Construction and Properties

- Groovy constructors support named parameters
- Groovy objects support property access via bean[property] and bean.property

```
package mypackage

class HelloWorld6 {

    String name

    def sayHello() { "Hello $name, do you earn more than \$1000000?" }

    static main(args) {
        def obj = new HelloWorld6(name:"Groovy v6-0")
        obj.name ="Groovy v6-1";
        obj["name"] = "Groovy v6-2"
        println( obj.sayHello() )
    }
}
```

HelloWorld6.groovy

# Step 7: Scripting

- Groovy compiles classes to Java byte code, but it also supports scripts!
  - They also compile down to Java byte code!
  - Scripts allow classes to be defined anywhere on them
  - Scripts support packages, after all they are also valid Java classes

```
class HelloWorld {  
    String name  
    def greet() { "Hello $name" }  
}  
  
def helloworld = new HelloWorld(name:"Groovy 7")  
println helloworld.greet()
```

HelloWorld7.groovy

# Any Questions?



# Exercises

1. Create a new Groovy project
  - Add a package
  - Add a Groovy class
  - Implement a minimal "Hello World" using Groovy syntax shortcuts where possible
2. Define methods
  - Add static methods to your class
  - For some methods, pass-in parameters
  - For some methods, return a value
3. (If time permits)
  - Investigate Groovy's support for properties

