Tutorial on Groovy

IS2150

Assoc Prof Kaushik Dutta
Department of Information Systems
School Computing
National University of Singapore

GROOVY STRINGS

In Groovy, a string can be defined three different ways: using double quotes, single quotes, or slashes (called "slashy strings").

Create a class called StringTesting1.groovy and copy paste the below code in your netbeans IDE.

```
public class StringTesting1

{

	public static void main(args)

	{

		// Quote

		def helloChris = "Hello, Chris"

		println helloChris.class.name // java.lang.String

		// Single quote

		def helloJoseph = 'Hello, Joseph'

		println helloJoseph.class.name //java.lang.String

		// Slashy string

		def helloJim = /Hello, Jim/

		println helloJim.class.name // java.lang.String

	}

}
```

Groovy also supports a more advanced string called a GString. A GString is just like a normal string, except that it evaluates expressions that are embedded within the string, in the form ${...}. When Groovy sees a string defined with double quotes or slashes and an embedded expression, Groovy constructs an org.codehaus.groovy.runtime.GStringImpl instead of a java.lang.String. When the GString is accessed, the expression is evaluated.

```
public class StringTesting2

{

	public static void main(args)

	{

		def name = "Jim"

		def helloName = "Hello, ${name}"

		println helloName // Hello, Jim

		println helloName.class.name

		// org.codehaus.groovy.runtime.GStringImpl

		def helloNoName = 'Hello, ${name}'

		println helloNoName // Hello, ${name}

		println helloNoName.class.name

		// java.lang.String

		def helloSlashyName = /Hello, ${name}/

		println helloSlashyName // Hello, Jim

		println helloSlashyName.class.name

		// org.codehaus.groovy.runtime.GStringImp

	}

}
```

String *interpolation* is the ability to substitute an expression or variable within a string. Strings defined using double quotes and slashes will evaluate embedded expressions within the string whereas strings defined with single quotes don't evaluate the embedded expressions.

Groovy supports strings that span multiple lines. A multiline string is defined by using three double quotes or three single quotes. String interpolation with multiline strings works in the same way as it does with regular strings: multiline strings created with double quotes evaluate expressions, and single-quoted strings don't.

```
public class StringTesting3

{

	public static void main(args)

	{

		def name = "Jim"

		def multiLineQuote = """

		Hello, ${name}

		This is a multiline string with double quotes

		"""

		println multiLineQuote

		println multiLineQuote.class.name

		def multiLineSingleQuote = '''

		Hello, ${name}

		This is a multiline string with single quotes

		'''

		println multiLineSingleQuote

		println multiLineSingleQuote.class.name

	}

}
```

CLOSURES

Closures are code fragments which can be used without being a method or a class.

A closure is defined via {para1, para2 -> code of the closure}. The values before the -> sign define the parameters of the closure. For the case that only one parameter is used you can use the implicit defined variable it.

The last statement of a closure is returned as return value.

The groovy collections have several methods which accept a closure as parameter, for example the each method.

```
public class ClosureTest1
{
	public static void main(args)
	{
		def name = "Chris"
		def printClosure = { println "Hello, ${name}" }
		printClosure()
		name = "Joseph"
		printClosure()
	}
}
```

```
public class ClosureTest2
{
	public static void main(args)
	{
		def printClosure = {name -> println "Hello,
		${name}"}
		printClosure("Chris")
		printClosure("Joseph")
		printClosure "Jim"
	}
}
```

Replace the code within the main function above with the below snippet.

```
def printClosure = {name1, name2, name3 -> println "Hello,
${name1},${name2}, ${name3}" }
printClosure "Chris", "Joseph", "and Jim"
```

```
public class ClosureTest4

{

	public static void main(args)

	{

		List<Integer> list = [5,6,7,8]

		list.each({line -> println line})

		list.each({println it})

	}

}
```

METHODS

We will now compare the difference between a Groovy method and a Java method, to illustrate how much more compact the Groovy version is.

```
public String hello(String name){

	Return "Hello, " + name;

}
```

```
def hello(name){

	"Hello, ${name}"

}
```

It is obvious which is which, isn't it? The Groovy version (the 2nd one if you haven't already guessed) uses a couple of Groovy's optional features:

- The **return type** and **return statement** are not included.
  Groovy always returns the **last** expression, in this case the GString "Hello, ${name}".
- The access modifier (public) is not defined. Unless specified, it is **public** by default for all classes, properties and methods.

LISTS

A Groovy List is an ordered collection of objects, as in Java. It is an implementation of the java.util.List interface.

Lets dive into creating and using Lists.

```
def emptyList = [] //empty list created
println emptyList.class.name //java.util.ArrayList
println emptyList.size //0

def list = ["Chris"] //List with one item in it
//Add items to list
list.add "Joseph" //optional () missing
list << "Jim " //overloaded left-shift operator
println list.size //3

//Iterate over the list like in last tutorial
//invokes a closure to print the contents
list.each{println it} //Chris Joseph Jim

//Access items in the list using index
println list[1] //Joseph, through indexed access
list[0] = "Christopher"
println list.get(0) //Christopoer, another way of access

list.set(0, "Chris") //Another way of setting
println list.get(0) //Chris

list.remove 2 //Removes Jim
list -= "Joseph" //Overloaded - operator
list.each {println it} //Chris
```

```
list.add "Joseph"

list += "Jim" //Overloaded + operator

list.each {println it} //Chris Joseph Jim

println list[-1] //Jim
```

Everything is pretty straight-forward, except the last line. It uses the index value -1. Using a negative index value causes the list to be accessed in the **opposite order**, from last to first.

Try accessing the above list with indexes **-2**, **-3** and **-4**, observing the output.