**Groovy**

**Differences with Java**

**1. Deffaut imports**

Những packeage và class sau được import mặc định trong groovy:

- java.io.\*

- java.lang.\*

- java.math.BigDecimal

- java.math.BigInteger

- java.net.\*

- java.util.\*

- groovy.lang.\*

- groovy.util.\*

**2. Multi – methods**

Trong groovy, các phương thức sẽ được chọn kiểu trong khi runtime. Tức là nó sẽ chọn kiểu của dữ liệu trong khi runtime. Đối với java thì ngược lại.

**3. Khởi tạo Array**

Trong Java, có 2 kiểu khởi tạo array:

int[] array = {1, 2, 3} cấu trúc khởi tạo ngắn

int[] array = new int[] {1,2,3} cấu trúc khởi tạo dài

Trong groovy, cặp dấu {..} được dành cho closures nên cấu trúc khởi tạo ngắn sẽ sửa thành int[] array = [1,2,3]

Với groovy 3 trở lên, ta có thể khai báo khởi tạo dài:

def array = new int[] {1, 2, 3}

**4. Package Scope Visibility**

Không có khái niệm access modifier cho các thuộc tính.

Việc khai báo hay không khai báo access modifier thì không ảnh hưởng đến khả năng truy xuất đến các thuộc tính của class (mặc định là public)

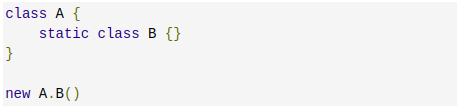
Để giới hạn khả năng truy cập, sử dùng annotating @PackageScope, tự động có get set.

**5. ARM blocks**

ARM được viết ngắn gọc bằng closure

**6. Inner Class**

**6.1. Static Inner Class**

****

**6.2 Create Instance của Inner class**

**7. Lambda Expresion và Method Refference**

**Trong groovy ta có thể thay băng closure:**

**Runable run = {println “run”}**

**8. GString**

Trong groovy, cặp dấu nháy kép được dùng cho String hoặc Gstring, GString sẽ chứa thêm cacus dấu $ để nối chuỗi thay vì dấu + trong java.

Trong java:

String name = "Nguyen Van Manh";

System.out.println("Xin chao " + name + "!");

Groovy:

String name = "Nguyen Van Manh";

println "Xin chao ${name}!"

**9. String and Character Literals**

Nháy đơn cũng dc dùng cho String thay vì char như java.lang

Kiểu cha chỉ được hiểu tường mình khi biến đó được khai báo là char.

Các cách cast từ String sang char

(char) “a”

“a” as char

“abc” as char - chri lấy ký tự đầu

“abc”.asType(char)

**10. Primitives and Wrapper**

Mọi thứ trong groovy đều là object → mọi biến primitives đều được autoboxing về wrapper class tương ứng

**11. Phép so sánh ==**

Trong groovy, so sánh == sẽ chuyển về a.compareTo(b) == 0 nếu là object Comparable, nếu không, sẽ chuyển về a.equal(b).

Để có thể xxem a và b cũng cùng trỏ về 1 object hay ko, dùng a.is(b)

**12.Groovy Conversion**

**13. Keywords**

as

def

in

trait

Từ khóa “as”: đặt bí danh cho import( import .. as ..), ép kiểu

Từ khóa “def”: khai bao biến / funtion

**14. Không dùng dấu ;**

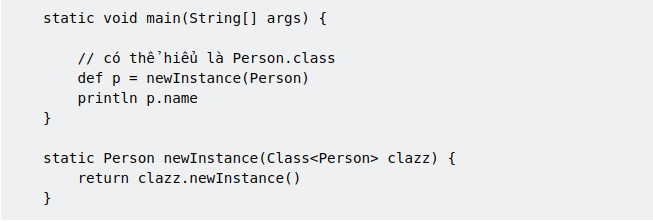
Groovy có thể có dấu ; hoặc không, không ảnh hưởng gì

**15. Từ khóa return không bắt buộc**

Việc dùng re turn để trả về kết quả là không bắt buộc, groovy tự động trả về kết quả cuối cùng của hàm

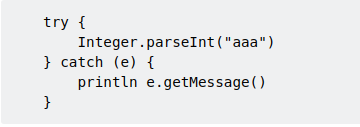
Tuy nhiên, nhiều TH return vẫn cần thiết

**16. Có thể bỏ qua đuôi .class khi class đc đưa như 1 parameter**



**17. Try catch**

Không cần khai tường mình loại exception trong catch



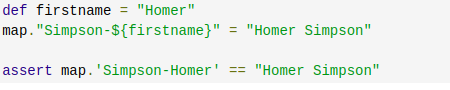
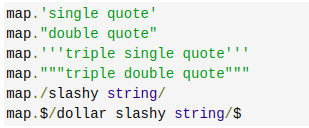
**Syntax**

**1. Quoted Identifier**

Có thể để phần sau dấu chấm trong dot expresstion ở trong một cặp dấy nháy đơn hoặc kép. Ví dụ person.name có thể để persion.’name’ hoặc persion.”name”

cách này có thể sử dụng khi phần sau dấu chấm là 1 cụm từ chứa các ký tự đặc biệt hoặc khoảng trắng

Groovy cung cấp nhiều dạng String Literals, tất cả đều có thể áp dụng vào phần sau dấu chấm



**2. String Interpolation**

Tất cả các biểu thức đều có thể được interpolated trong String literals, trừ single và triple-single-quoted string. Interpolation (nội suy) mô tả việc thay thế một placeholder (biểu thức giữ chỗ) bằng giá trị của nó khi đánh giá chuỗi. Một placeholder được bao quanh bơi ${}. Dấu ngoặc nhọn có thể dc bỏ qua với các dot expresstion.

Ngoài expresstion, ta có thể truyền vào expresstion các câu lệnh, tuy nhiên các câu lệnh bằng cách nào có nên trả về 1 giá trị, nếu ko, nó sẽ trả về null

Trường hợp đặc biết khi Interpolation một closure expresstion. Các closure lợi thế hơn các biểu thức thông thường - đó là đánh giá lười biếng. Nó sẽ được thực hiện chỉ khi GString được chuyển thành String để in ra

GString và String tương đương có hashCode khác nhau - lưu ý khi sử dụng trong map

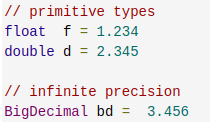
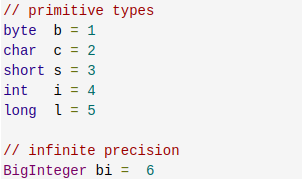
**3. Slashy String**

Groovy cung cấp chuỗi slashy được giới hạn bởi cặp dấu /, rất hữu ích trong biểu thức chính quy vì không phải escape dấu \.



4. **Number**

Groovy hỗ trợ khởi tạo các kiểu number.

Nếu khai bào bằng từ khóa def, tùy vào độ lớn của số mà sẽ dc hiểu vào các kiểu tương ứng.

Number type suffix:

**5. List**

List được khai báo mặc địch bởi cặp dấu ngoặc vuông và các phần tử ở trong. Mặc định, nó sẽ được triển khai theo ArrayList, tuy nhiên ta có thể thay đổi

**6. Arrays**

Array khai báo giống snhuw List nhưng phải được chỉ định rõ

**7.Maps**

Map giống như một List chứa các cặp khóa, giá trị. các khóa không trùng nhau, khóa và giá trị được phân cách bởi dấu :

Ví dụ

def map = [red: “#FF0000”, green: “00FF00”, blue: “0000FF”]

Khi truy cập ta dùng map.red hoặc map[“red”]

Map được tạo sẽ là LinkedHashMap

Khi khai báo như trên, ta mặc định các key là string - là tên mà ta truyền vào.

def key = “name”

def map = [key: “hung”]

trường hợp này map.containKey(“key”) = true, và nó ko chứa key nào là “name”

Nếu muốn đưa gia trị vào làm key, ta đưa biến đó trong cặp dấu ()

def key = “name”

def map = [(key): “hung”]

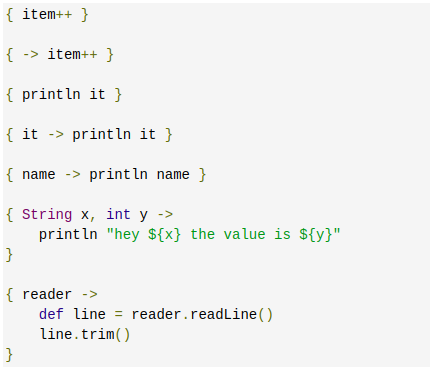
lúc này, map sẽ chưa key là “name” và không chưa key là “key”.

**Closures**

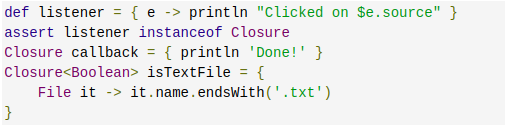
**1. Syntax**

Một closures được xác định bới ký hiệu {..}

{ [closureParameters -> ] statements }



Một closures là một instance của class groovy.lang.Closure



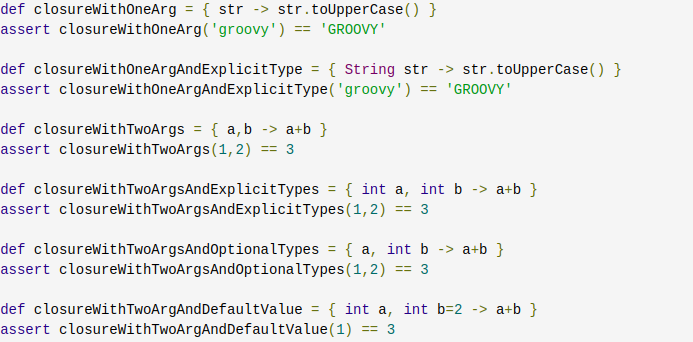
Closures có thể được gọi bởi một method khác. Những statements trong blocks code chỉ được gọi thì closures được gọi ví dụ

Ví dụ listener() hoặc listener.call()

Tương tự với các closures có tham số

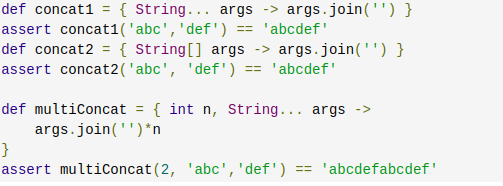
**2. Parameters**

Tham số của closures tuân theo nguyên tắc tham số thông thường



Mỗi một clusures luôn có một tham số ngầm định là có tên là it

Có thể truyền nhiều tham số giống nhau như sau:



**3. Delegation strategy**

Closures là một instance của lớp Closure, nên nó khác lambda expression. Delegation là một khái niệm trong closures mà lambda không có.

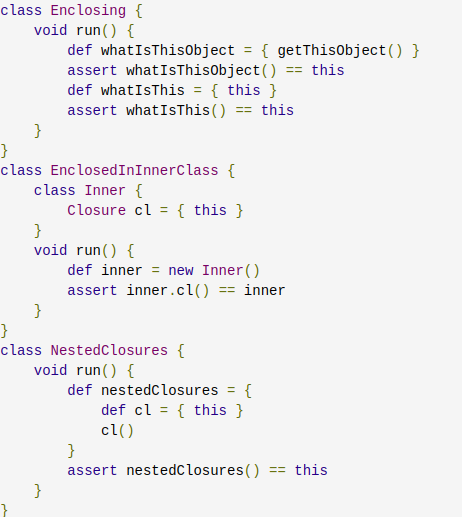
Một closures có 3 thứ:

- this: tương ứng với lớp chứa closures

- owner: tương ứng với object chứa closures, có thể là 1 class khác hoặc closures

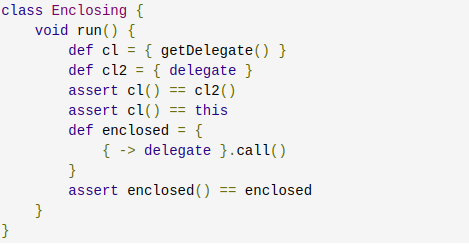
- delegate: tương ứng với object thứ 3 where methods calls or properties are resolved whenever the receiver of the message is not defined

Ví dụ về this:



Ví dụ về owner

Ví dụ delegate:







Currying - cho phép đặt sẵn giá trị của một vài tham số. Nó sẽ trả về một closures với các tham số còn lại

**4. Memoization**

Có thể lưu kết quả của closures bằng gọi .memoize()

Bọ nhớ đệm sẽ lưu kết quả thực vào cache, vì vậy lưu ý khi dùng với Object.

Hoạt động lưu có thể điều chình bằng các hàm khác:

memoizeAtMost: lưu trữ nhiều nhất n giá trị.

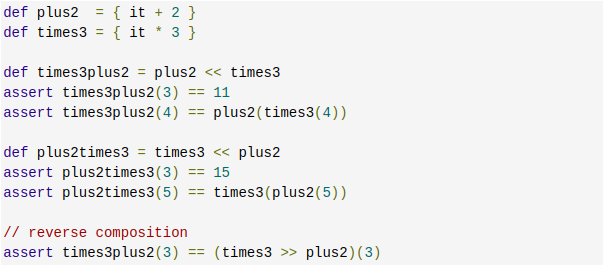
memoizeAtLeast: lưu trữ ít nhất n giá trị

memoizeBetween: lưu trữ ít nhất n và nhiều nhất m

Bộ đệm được dùng là LRU cache

**5. Composition**

Là cách tạo một closure mớibằng cách kết hợp nhiều closures



**6. Trampoline**