**Default imports**

All these packages and classes are imported by default, i.e. you do not have to use an explicit import statement to use them:

* java.io.\*
* java.lang.\*
* java.math.BigDecimal
* java.math.BigInteger
* java.net.\*
* java.util.\*
* groovy.lang.\*
* groovy.util.\*

**Multi methods**

* In groovy, metoda de executat e aleasa la runtime, nu la compile time ca Java
* De ex, in Java:
* public class Test {  
   public static void main(String[] args) {  
   Object string = "test";  
   *met1*(string);  
   }  
   public static void met1(Object o){  
   System.*out*.println("Object");  
   }  
   public static void met1(String o){  
   System.*out*.println("String");  
   }  
  }

Object

In groovy, nu se va lua tipul obiectului dupa upcasting, ci se va lua tipul lui general

class Main {  
 static void main(String[] args) {  
 Object string = "test";  
 *met1*(string);  
 }  
 static void met1(Object o){  
 System.*out*.println("Object");  
 }  
 static void met1(String o){  
 System.*out*.println("String");  
 }  
}

String

**Array**

* {} sunt pentru closures, deci nu se folsoesc la array!
* Asa ceva totusi e posibil

def array = new int[] {1,2,3,4}

**public and default**

* In groovy, orice membru este automat public
* Protected si Private sunt mai mult doar sugestii, nu au nicio valoare, caci macar si membrii privati pot fi accesati:
* class MainClass{  
   static void main(String[] args) {  
   println(SecondClass.*a*)  
   }  
  }  
    
  class SecondClass{  
   private static def *a* = 100;  
  }

vom primi doar un avertisment de la compilator si gata

**inner class instance**

* In Java doar asa putem crea un obiect de tip local inner class:

Clasa.clasaInner variabila = obiect.new clasaInner()

* In groovy, putem si asa:

new clasaInner(obiect)

class Y {  
 class X {}  
 X foo() {  
 return new X()  
 }  
 static X createX(Y y) {  
 return new X(y)  
 }  
}

**Method reference**

* Groovy suporta ::, dar mai are si .&
* list.forEach(System.*out*.&println)

**String vs Gstring**

* Gstring nu mosteneste de la String, caci String e final
* Cand folosim ‘’, mereu avem string
* Daca folosim “” sau “”” “””, fie avem string, fi gstring
* Daca folosim “” sau “”” “””fara ${}, se creaza String
* Daca folosim “” sau “”” “””cu ${}, groovy va crea Gstring
* **hashcode la String si Gstring diferentiaza chiar si pentru acelasi string!**
* def a = "Hello $**{**10**}**"  
    
  println(a.getClass().getName())

GstringImplementation

def a = "Hello"  
  
println(a.getClass().getName())

String

* Daca o metoda are ca parametru un String si noi oferim un Gstring, asupra la Gstrign se va apela automat toString() si va fi un string in parmetrul metodei
* String takeString(String message) {  
   assert message instanceof String  
   return message  
  }  
    
  def message = "The message is $**{**'hello'**}**"  
  assert message instanceof GString  
    
  def result = takeString(message)  
  assert result instanceof String  
  assert result == 'The message is hello'

**“”” “””**

* Daca folosim ${}, avem Gstring, daca nu, String
* Daca vom folosi \ la final de rand, \n va fi ignorat.
* De ex:
* println( """  
  Line1  
  Line2  
  Line3  
  """  
  )

avem 4 linii, caci “”” mereu incepe cu \n

Dar acum avem doar 2 :

println( """\  
Line1\  
Line2  
Line3  
"""  
)

Line1Line2

Line3

* .**stripMargin()** – folosind |, aratam de unde propriu zis incepe linia
* println( """\  
   |Line1  
  Line2   
   |Line3  
  """.stripMargin()  
  )

Line1

Line2

Line3

au fost ignorate space de pana la |

**Slashy string**

* Reg exp folosesc multe \

**def** pattern = \\d{1,3}\\s\\w+\\s\\w+\\\\\\w+

asta arata urat de tot

* Slashy string vede \ ca simplu string, nu ca simbol special, de accea
* println(/\t/)

va afisa \t

* De aceeam putem crea un string pentru reg exp cu / /

**def** pattern = /\d{3}\s\w+\s\w+\\\w+/

* / / tot este de tip string sau gstring
* **.matches(regexp)**

Fiecare string are metoda .matches(regexp) care ia un regexp pentru

println("A test message".matches(/\w\s.{20}/))

* \\ va fi luat ca comentariu, nu empt string!
* tot suporta ${}

**Dolar slashy string**

* Are forma $\ \$
* Tot suporta ${}, dar daca vom pune doi dolari, $$, atunci nu vom mai avea un placholder, ci un $ simplu
* println($/$$a/$)

$a

**char**

* ‘’ nu este char!
* Daca vrem sa transforma un string in char, folosim

- (char)

- as char

- autocasting

**char** b = 'B' as **char**

**char** c = (**char**) 'C'

char a = 'a';

**==**

* == nu compara referintele, ci chiar valorile
* == foloseste .equals() automat, daca obiectul nu implementeaza Comparable
* package test.pack2  
    
  def obj1 = new Test("test",20);  
  def obj2 = new Test("test",20);  
    
  println(obj1 == obj2)  
    
  class Test{  
   String name;  
   int age;  
    
   Test(String name, int age) {  
   this.name = name  
   this.age = age  
   }  
    
   boolean equals(o) {  
   if (this.is(o)) return true  
   if (getClass() != o.class) return false  
    
   Test test = (Test) o  
    
   if (age != test.age) return false  
   if (name != test.name) return false  
    
   return true  
   }  
    
   int hashCode() {  
   int result  
   result = (name != null ? name.hashCode() : 0)  
   result = 31 \* result + age  
   return result  
   }  
  }

true

* Daca foloseste comparable, va folosi compare()
* Pentru a compara referintele, folosim **.is() sau ===**

println(obj1.is(obj2))

false

**Boxing**

* Groovy la necesitate transforma automat primitivele in wrapper
* println(5.compareTo(10))

**extra keywords**