* **Closures** – A closure in Groovy is an open, anonymous, block of code that can take arguments, return a value and be assigned to a variable
* Poate defini variabile in scope al sau si sa acceseze variabile din alte scopes.
* Sintaxa:

{ parametri -> instructiuni }

* Oricate instructiuni nu am avea, nu mai e necesar { }

{ parametri -> { instructiuni } }



* Orice closure este un obiect de tip **groovy.lang.Closure**
* Closure e apelata exact ca o metoda, odata ce e atribuita unei variabile:

Closure closure = **{ ->** println("Hello")**}**closure()

* Putem specifica si un generic la closure ca tip de return
* Closure<Boolean> closure = **{**String text **->** text.startsWith("Hello")**}**println closure("Hello world")
* Un closure poate sa aiba si 0 argumente
* Closure closure = **{**println("Hello")**}**
* Closure object mereu apeleaza metoda call() cand noi folosim ()
* Cand closure nu specifica niciun parametru si deci nu foloseste ->, el mereu va avea un parametru default “it”
* Closure closure = **{**println(it)**}**closure("Eduard")
* Daca vrem sa nu fie “it” disponibil, folosim doar -> fara argumente
* Closure closure = **{ ->** println(it)**}**closure("Eduard")



**owner, delegate and this**

* Orice closure are metoda getThisObject(), getOwner() si getDelegate()
* **this** – in closure, this mereu are referinta la obiectul in care se gaseste, ca orice metoda ce apeleaza this. E vorba de obiectul original, care a fost creat cu new. Daca e vorba de inner class, this e referinta la obiectul la clasa inner:
* class Class1{  
   class Class2{  
   def getThis = **{** return (this instanceof Class2);  
   **}** }  
  }  
  Class1 obj1 = new Class1();  
  Class1.Class2 obj2 = new Class1.Class2(obj1);  
  println(obj2.getThis())
* **owner –** returneaza primul obiect in care e definita closure, fie ca e o clasa, sau alta closure
* class Class1{  
   class Class2{  
   def getOwner = **{** def getThis = **{** return (owner == getOwner);  
   **}** return getThis();  
   **}** }  
  }  
  Class1 obj1 = new Class1();  
  Class1.Class2 obj2 = new Class1.Class2(obj1);  
  println(obj2.getOwner())
* **delegate –** permite sa setam orice obiect ca delegate. In mod default, delegate e setat ca owner, delegate = owner
* class Class1{  
   String name = "Eduard"  
  }  
  class Class2{  
   String name = "Alex"  
  }  
    
  Class1 obj1 = new Class1()  
  Class2 ob2 = new Class2()  
  def closure = **{** println(delegate.name)**}**closure.delegate = obj1  
  closure()  
  closure.delegate = ob2  
  closure()

**Eduard**

**Alex**

**Closures in Gstring**

* **Gstring** are o problema, si anume ca folosind ${variabila}, Gstring e format la momentul declararii lui. Odata ce varibila isi modifica valoarea, Gstring ramane inca cu valoarea veche
* def x = "10"  
  def string = "$**{**x**}**"  
    
  println(x == string)  
    
  x = "100"  
    
  println(x == string)

**true**

**false**

* **Solutia** este sa folosim closure, deci in loc de ${x} folosim ${ -> x} si asa la fiecare folosire a variabilei cu referinta catre gstring cu closure, closure mereue va fi executat
* def x = "var = 10"  
  def string = "$**{ ->** x**}**"  
    
  println(x == string)  
    
  x = "var = 100"  
    
  println(x == string)

**Abstract methods**

* Closure, exact ca lambda, pot fi folosite pentru a suprascrie metodele abstracte:
* @FunctionalInterface  
  interface Test{  
   boolean isMillion(int number);  
  }  
    
  Test test = **{**it == 1\_000\_000**}**println(test.isMillion(1000000))

**Map**

* Putem suprascrie metodele unei interfete prin a crea o mapa unde key e numele la metoda abstracta si value e closure pentru ea
* interface Test{  
   boolean isMillion(int number);  
   boolean isPositive(int number);  
  }  
    
  def map = [isMillion : **{**it === 1000000**}**, isPositive : **{**it > 0**}**]  
  Test test = map as Test  
  println(test.isMillion(1000000))  
  println(test.isPositive(10))