CoffeeScript vs Java

Bonne Année 2012

moi

```
Philippe =
    age : 43
    emplois :
        actuellement :
            poste : "Bid Manager"
            employeur : "Steria Lyon"

        avant : [
                "Technico-commercial"
                "Développeur"
                "Responsable informatique"
                "Chef de projet"
                "Architecte"
                "Directeur de projet"
                "Directeur technique"
                "Responsable avant-vente"
            ]

    technos :
        avant : ["Cobol", "Visual Basic", "Visual FoxPro", "DBase", "..."]
        puis : ["Flash", ".Net", "..."]
        maintenant : ["Java", "Javascript", "..."]

    signeParticulier : ["Mac Addict", "SmartPhone Addict"]
```

Objectif

Démontrer que CoffeeScript est orienté “Classes”

... du coup Javascript aussi

On va parler de

Classes

Héritage

Design Patterns

+ 2 ou 3 autres trucs

CoffeeScript ?

CoffeeScript

"Transpiler" Javascript

Jeremy Ashkenas / @jashkenas

V° 1.2.0

http://coffeescript.org/

https://github.com/jashkenas/coffee-script

Classes

Human.java

```java
class Human {
    public String firstName;
    public String lastName;

    public Human(String first, String last) {
        this.firstName = first;
        this.lastName = last;
    }

    public void hello() {
        System.out.println("Hello "+this.firstName+" "+this.lastName);
    }
}


public class Demo {

    public static void main(String[] args) {
        Human bob = new Human("Bob","Morane");
        bob.hello();
        bob.firstName = "BOB";
        bob.lastName = "MORANE";
        bob.hello();
    }
}
```

Human.coffee

```coffeescript
class Human
    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last

    hello:->
        console.log "Hello #{@firstName} #{@lastName}"

bob = new Human "Bob", "Morane"
bob.hello()
bob.firstName = "BOB"
bob.lastName = "MORANE"
bob.hello()

console.log typeof bob
console.log bob.constructor.name
```

Qu’est ce qui change ?

@, :->

```
class Human
    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last

    hello:->
        console.log "Hello #{@firstName} #{@lastName}"

bob = new Human "Bob", "Morane"
bob.hello()
bob.firstName = "BOB"
bob.lastName = "MORANE"
bob.hello()

console.log typeof bob
console.log bob.constructor.name
```

Human.coffee again

```
class Human
    constructor:(@firstName = "John", @lastName = "Doe")->

    hello:->
        console.log "Hello #{@firstName} #{@lastName}"

```

en js, ça donne quoi ?

```
var Human, bob, john;
Human = (function() {
  function Human(firstName, lastName) {
    this.firstName = firstName != null ? firstName : "John";
    this.lastName = lastName != null ? lastName : "Doe";
  }
  Human.prototype.hello = function() {
    return console.log("Hello " + this.firstName + " " +
     this.lastName);
  };
  return Human;
})();
```

étape suivante

En fait, on peut tout faire comme en Java !

... ou presque

Composition

Java

```
class Hand {
    public String whichOne = "";
    public Hand(String which_one){ this.whichOne = which_one; }
    public void take(String something) {
        System.out.println("Taking " + something + " with the " + this.whichOne +
" hand" );
    }
}

class Human {

    public String firstName = "???";
    public String lastName = "???";

    public Hand rightHand = new Hand("right");
    public Hand leftHand = new Hand("left");

    public Human(String first, String last) {
        this.firstName = first;
        this.lastName = last;
    }
}
```

CoffeeScript

```coffeescript
class Hand
    constructor:(which_one)->
        @whichOne = which_one

    take:(something)->
        console.log "Taking #{something} with the #{@whichOne} hand"

class Human

    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last
        @leftHand = new Hand "left"
        @rightHand = new Hand "right"


bob = new Human "Bob", "Morane"

bob.rightHand.take "a book"
bob.leftHand.take "a glass"
```

Association

Java

```
class Dog {
    public String name = "";
    public Dog(String name){ this.name = name; }
}

class Human {
    public String firstName = "???";
    public String lastName = "???";
    public Dog hisDog = null;

    public Human(String first, String last) {
        this.firstName = first;
        this.lastName = last;
    }

    public void adopt(Dog dog) {
        this.hisDog = dog;
        System.out.println(this.firstName+" "+this.lastName+" adopts "+ dog.name);
    }

    public void giveHisDogTo(Human human) {
        human.hisDog = this.hisDog;
        System.out.println(this.firstName+" "+this.lastName+" gives "+this.hisDog.name
          +" to "+human.firstName+" "+human.lastName);
        this.hisDog = null;
    }
}
```

CoffeeScript

```
class Dog
    constructor:(name)->
        @name = name

class Human

    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last
        @hisDog = null

    adopt:(dog)->
        @hisDog = dog
        console.log "#{@firstName} #{@lastName} adopts #{dog.name}"

    giveHisDogTo:(human)->
        human.hisDog = @hisDog
        console.log "#{@firstName} #{@lastName} gives #{@hisDog.name} to
#{human.firstName} #{human.lastName}"
        @hisDog = null
```

Encapsulation

Java

```
class Human {

    class Hand {
        public String whichOne = "";
        public Hand(String which_one){ this.whichOne = which_one; }
        public void take(String something) {
            System.out.println("Taking " + something + " with the " + this.whichOne +
             " hand" );
        }
    }

    public String firstName = "???";
    public String lastName = "???";

    public Hand rightHand = new Hand("right");
    public Hand leftHand = new Hand("left");

    public Human(String first, String last) {
        this.firstName = first;
        this.lastName = last;
    }
}
```

CoffeeScript

```
class Human

    class Hand
        constructor:(which_one)->
            @whichOne = which_one

        take:(something)->
            console.log "Taking #{something} with the #{@whichOne} hand"

    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last
        @leftHand = new Hand "left"
        @rightHand = new Hand "right"

bob = new Human "Bob", "Morane"

bob.rightHand.take "a book"
bob.leftHand.take "a glass"
```

Héritage

Java

```java
class Human {
    public String firstName = "???";
    public String lastName = "???";

    public Human(String first, String last) {
        this.firstName = first;
        this.lastName = last;
    }

    public void hello() {
        System.out.println("Hello "+this.firstName+" "+this.lastName);
    }
}

class SuperHero extends Human {

    public String name;

    public SuperHero(String first, String last, String name) {
        super(first, last);
        this.name = name;
    }
    public void secret() {
        System.out.println("Hello "+this.name);
    }
}
```

CoffeeScript

```coffeescript
class Human
    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last

    hello:->
        console.log "Hello #{@firstName} #{@lastName}"


class SuperHero extends Human
    constructor:(first, last, name)->
        super first, last
        @name = name

    secret:->
        console.log "Hello #{@name}"

clark = new SuperHero "Clark", "Kent", "SuperMan"
clark.hello()
clark.secret()
```

Static ?

Java

```java
class Human {
    public String firstName = "???";
    public String lastName = "???";

    public static Integer humanCounter = 0;

    public Human(String first, String last) {
        this.firstName = first;
        this.lastName = last;
        humanCounter +=1;
    }
}

class SuperHero extends Human {

    public String name;

    public static Integer superHeroCounter = 0;

    public SuperHero(String first, String last, String name) {
        super(first, last);
        superHeroCounter +=1;
        this.name = name;
    }
}
```

CoffeeScript

```coffeescript
class Human
    #Static variable
    humanCounter : 0

    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last

        Human::humanCounter += 1
class SuperHero extends Human
    #Static variable
    superHeroCounter : 0

    constructor:(first, last, name)->
        super first, last
        @name = name

        SuperHero::superHeroCounter += 1
```

```
bob = new Human "Bob", "Morane"
sam = new Human "Sam", "LePirate"

clark = new SuperHero "Clark", "Kent", "SuperMan"
peter = new SuperHero "Peter", "Parker", "SpiderMan"

console.log "Human Counter (from Human) : #{Human::humanCounter}
console.log "Human Counter (from SuperHero) : #{SuperHero::humanCounter}"
console.log "SuperHero Counter : #{SuperHero::superHeroCounter}"
```

```
Human Counter (from Human) : 4
Human Counter (from SuperHero) : 4
SuperHero Counter : 2
```

Static method ?

CoffeeScript

```
class Human
    #Static variable
    humanCounter : 0

    #Static Method
    @getHumanCounter:->
        Human::humanCounter

    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last
        Human::humanCounter += 1


bob = new Human "Bob", "Morane"
sam = new Human "Sam", "LePirate"

console.log "Human Counter : #{Human.getHumanCounter()}"
```

Static, attention !

CoffeeScript

```
class Human
    #Static variable ?!
    @humanCounter : 0

    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last
        Human.humanCounter += 1

class SuperHero extends Human
    #Static variable
    @superHeroCounter : 0

    constructor:(first, last, name)->
        super first, last
        @name = name
        SuperHero.superHeroCounter += 1

bob = new Human "Bob", "Morane"
sam = new Human "Sam", "LePirate"

clark = new SuperHero "Clark", "Kent", "SuperMan"
peter = new SuperHero "Peter", "Parker", "SpiderMan"

console.log "Human Counter (from Human) : #{Human.humanCounter}"
console.log "Human Counter (from SuperHero) : #{SuperHero.humanCounter}"
console.log "SuperHero Counter : #{SuperHero.superHeroCounter}"
```

```
bob = new Human "Bob", "Morane"
sam = new Human "Sam", "LePirate"

clark = new SuperHero "Clark", "Kent", "SuperMan"
peter = new SuperHero "Peter", "Parker", "SpiderMan"

console.log "Human Counter (from Human) : #{Human.humanCounter}
console.log "Human Counter (from SuperHero) : #{SuperHero.humanCounter}"
console.log "SuperHero Counter : #{SuperHero.superHeroCounter}"
```

```
Human Counter (from Human) : 4
Human Counter (from SuperHero) : 0
SuperHero Counter : 2
```

Design Patterns
... juste 2

Singleton

Singleton.java

```java
class SantaClaus {

    private static SantaClaus uniqueSantaClaus;

    public String name = "Santa Claus";

    private SantaClaus() {}

    public static SantaClaus getTheOne() {
        if(uniqueSantaClaus == null) {
            uniqueSantaClaus = new SantaClaus();
        } else {
            System.out.println("Bien essayé mais il n'existe qu'un seul "
             + uniqueSantaClaus.name);
        }
        return uniqueSantaClaus;
    }
}
```

Singleton.coffee

```
class SantaClaus
    uniqueSantaClaus:null
    constructor:->
        @name = "SANTA CLAUS"

    @getTheOne:->
        if SantaClaus::uniqueSantaClaus is null
            SantaClaus::uniqueSantaClaus = new SantaClaus()
        else
            console.log "BIEN ESSAYE MAIS IL N'EXISTE QU'UN SEUL
             #{SantaClaus::uniqueSantaClaus.name}"

        SantaClaus::uniqueSantaClaus
```

Singleton.bis.coffee

```
class SantaClaus
    uniqueSantaClaus:null
    constructor:()->
        @name = "SANTA CLAUS"
        if not arguments.length then return SantaClaus.getTheOne()

    @getTheOne:()->
        if SantaClaus::uniqueSantaClaus is null
            console.log "NEW"
            SantaClaus::uniqueSantaClaus = new SantaClaus(true)
        else
            console.log "BIEN ESSAYE MAIS IL N'EXISTE QU'UN SEUL
             #{SantaClaus::uniqueSantaClaus.name}"

        SantaClaus::uniqueSantaClaus
```

Factory

on transforme le Père-Noël en usine

Factory.java (1)

```
interface Toy {
    public void what();
}
class Car implements Toy {
    public void what() {
        System.out.println("this is a car");
    }
}
class Doll implements Toy {
    public void what() {
        System.out.println("this is a doll");
    }
}
```

Factory.java (2)

```java
class SantaClaus {

    private static SantaClaus uniqueSantaClaus;
    public String name = "Santa Claus";
    private SantaClaus() {}

    public static SantaClaus getTheOne() {
        if(uniqueSantaClaus == null) {
            uniqueSantaClaus = new SantaClaus();
        } else {
            System.out.println("Bien essayé mais il n'existe
             qu'un seul " + uniqueSantaClaus.name);
        }
        return uniqueSantaClaus;
    }

    public Toy offers(String toyName) {
        Toy toy = null;
        if(toyName=="car"){ toy = new Car(); }
        if(toyName=="doll"){ toy = new Doll(); }
        return toy;
    }
}
```

Factory.coffee

```
class Doll
    what:->
        console.log "this is a doll"

class Car
    what:->
        console.log "this is a car"

class SantaClaus
    uniqueSantaClaus:null
    constructor:->
        @name = "SANTA CLAUS"

    @getTheOne:->
        if SantaClaus::uniqueSantaClaus is null
            SantaClaus::uniqueSantaClaus = new SantaClaus()
        else
            console.log "BIEN ESSAYE MAIS IL N'EXISTE QU'UN SEUL
             #{SantaClaus::uniqueSantaClaus.name}"

        SantaClaus::uniqueSantaClaus

    offers:(toyName)->
        if toyName is "car" then return new Car
        if toyName is "doll" then return new Doll
```

Pas d’interface ?!

... ben non !

Factory.bis.coffee (1)

```
class FakeAbstractToy
    what:->

class Doll extends FakeAbstractToy
    what:->
        console.log "this is a doll"

class Car extends FakeAbstractToy
    what:->
        console.log "this is a car"

class Game extends FakeAbstractToy
```

Factory.bis.coffee (2)

```
class SantaClaus
    uniqueSantaClaus:null
    constructor:->
        @name = "SANTA CLAUS"

    @getTheOne:->
        if SantaClaus::uniqueSantaClaus is null
            SantaClaus::uniqueSantaClaus = new SantaClaus()
        else
            console.log "BIEN ESSAYE MAIS IL N'EXISTE QU'UN SEUL
             #{SantaClaus::uniqueSantaClaus.name}"

        SantaClaus::uniqueSantaClaus

    offers:(toyName)->
        if toyName is "car" then return new Car
        if toyName is "doll" then return new Doll
        if toyName is "game" then return new Game
```

Getters, Setters, private ?

Human.java

```java
class Human {
    private String firstName;
    private String lastName;

    public String getFirstName() {
        return this.firstName;
    }
    public void setFirstName(String value) {
        this.firstName = value;
    }

    public String getLastName() {
        return this.lastName;
    }
    public void setLastName(String value) {
        this.lastName = value;
    }

    public Human(String first, String last) {
        this.firstName = first;
        this.lastName = last;
    }

    public void hello() {
        System.out.println("Hello "+this.firstName+" "+this.lastName);
    }
}
```

Human.coffee

```
class Human

    constructor:(first, last)->
        #private variables
        firstName = first
        lastName = last

        #Getters Setters
        @getFirstName = ->
            firstName
        @setFirstName = (value)->
            firstName = value

        @getLastName = ->
            lastName
        @setLastName = (value)->
            lastName = value

    hello:->
        # !!! hello ne peut pas accéder aux variables firstName & lastName
        console.log "Hello #{@getFirstName()} #{@getLastName()}"
```

Mais ...

JSON.stringify bob

```
bob = new Human "Bob", "Morane"

console.log JSON.stringify bob

{} !!!
```

... Properties

Human.coffee

```coffee
class Human

    constructor:(first, last)->
        #private variables
        firstName = first
        lastName = last

        #properties
        Object.defineProperty @, "FirstName",
            get:->
                firstName
            set: (value)->
                firstName = value
            enumerable: true
            configurable: true

        Object.defineProperty @, "LastName",
            get:->
                lastName
            set: (value)->
                lastName = value
            enumerable: true
            configurable: true

    hello:->
        # !!! hello ne peut pas accéder aux variables firstName & lastName
        console.log "Hello #{@FirstName} #{@LastName}"
```

JSON.stringify bob

```
bob = new Human "Bob", "Morane"

console.log JSON.stringify bob
```

```
{
  "FirstName":"Bob",
  "LastName":"Morane"
}
```

=>

... un truc que Java n’a pas ...

Human.coffee

prototype

```
class Human
    constructor:(@firstName = "John", @lastName = "Doe")->

    hello:->
        console.log "Hello #{@firstName} #{@lastName}"

john = new Human
bob = new Human "Bob", "Morane"
sam = new Human "Sam", "LePirate"

#Human.prototype.hello
Human::hello = ->
    console.log "Salut #{@firstName} #{@lastName}"


john.hello()
bob.hello()
sam.hello()

angelina = new Human "Angelina", "Jolie"
angelina.hello()
```

```
Salut John Doe
Salut Bob Morane
Salut Sam LePirate
Salut Angelina Jolie
```

Human.coffee

prototype & =>

```
class Human
    constructor:(@firstName = "John", @lastName = "Doe")->

    hello:=>
        console.log "Hello #{@firstName} #{@lastName}"

john = new Human
bob = new Human "Bob", "Morane"
sam = new Human "Sam", "LePirate"

#Human.prototype.hello
Human::hello = ->
    console.log "Salut #{@firstName} #{@lastName}"

john.hello()
bob.hello()
sam.hello()

angelina = new Human "Angelina", "Jolie"
angelina.hello()
```

```
Hello John Doe
Hello Bob Morane
Hello Sam LePirate
Salut Angelina Jolie
```

One more thing

On peut ajouter du code exécutable entre les définitions des membres d’une classe

Human.coffee

```
class Human

    console.log "Hello world !"
    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last

    console.log "Hello world ! again"
    hello:->
        console.log "Hello #{@firstName} #{@lastName}"
```

Cela ne s’exécutera
qu’une seule fois

à quoi ça sert ?

Human.coffee

```
annotations = (what, member, value)->
    if not what.annotations then what.annotations = {}
    what.annotations[member] = value

class Human

    annotations @, "firstname",
       {placeholder : "First Name", inputtype : "text"}
    annotations @, "lastname",
       {placeholder : "Last Name", inputtype : "text"}

    constructor:(first, last)->
        #public variables
        @firstName = first
        @lastName = last
```

Human.coffee

```
class HumanForm
    constructor:(k)->
        @template = ""
        for m of k.annotations
            @template += """
              <input
                type='#{k.annotations[m].inputtype}'
                placeholder='#{k.annotations[m].placeholder}'/>\n
            """

F = new HumanForm Human
console.log F.template
```

```
<input
   type='text'
   placeholder='First Name'/>
<input
   type='text'
   placeholder='Last Name'/>
```

Conclusion

Orientation “Class”

Les “goodies” de Javascript

Un générateur de “bon code”