**Scala en Java**

**Autores:**

* Saldaña Bartra Jean Pierre
* Francisco Yauli Vanesa
* Gutierrez Ortega Junnior Jhair

**Nombre del docente:**

* Coronel Castillo Gustavo Erick

**Tema :** Scala

**Fecha de entrega :** 20/10/2017

**Ciclo:** 02

**Turno:** noche

**Escuela**: Ing. De Sistemas e Informática

**Universidad**: UCH



**Resumen:**

Este documento provee una rápida introducción al lenguaje Scala como también a su compilador. Está pensado para personas que ya poseen cierta experiencia en programación y quieren una vista rápida de lo que pueden hacer con Scala. Se asume como un conocimiento básico de programación orientada a objetos, especialmente en Java.

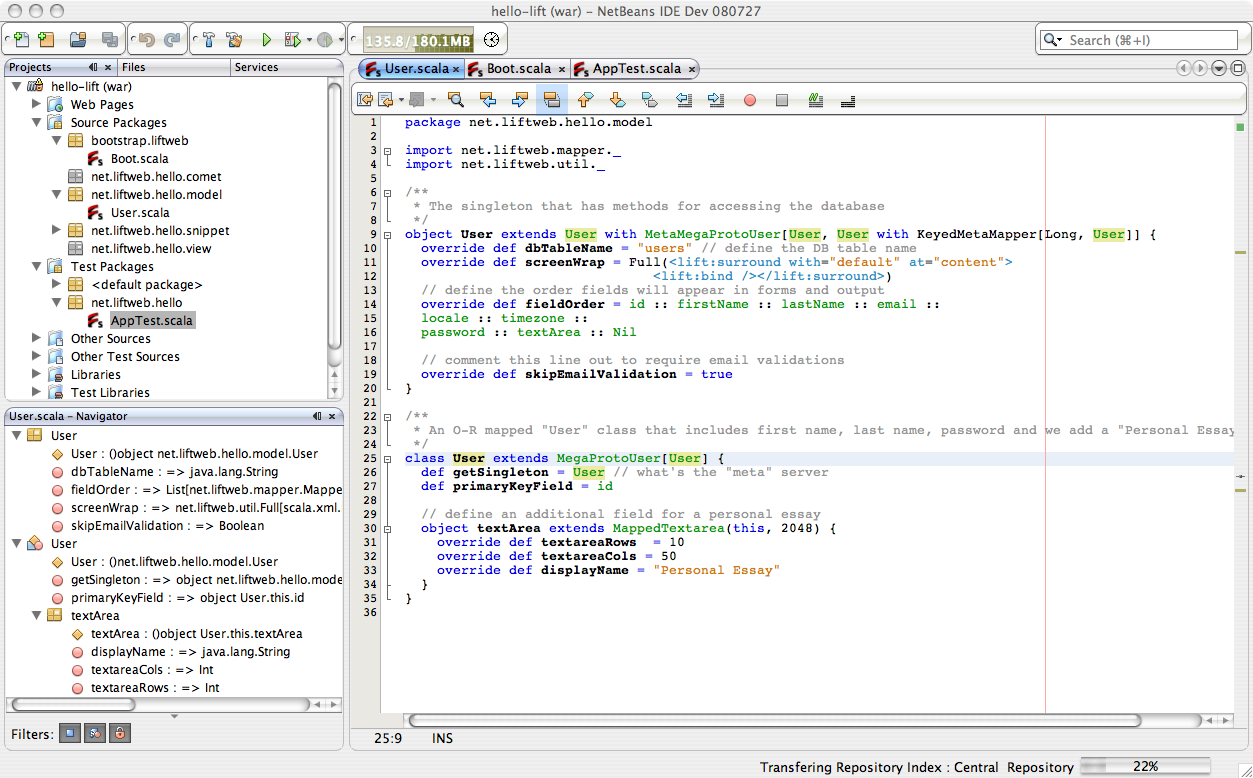
Además de tocar los puntos fundamentales como características , importancia , ventajas al utilizar Scala, entre otros beneficios además de que es accesible a todos.

Las clases case son uno de los mecanismos que Scala utiliza para hacer "pattern matching" sobre objetos, sin la necesidad de escribir el código de boilerplate , En Scala, las clases son plantillas estáticas que pueden ser instanciadas por muchos objetos en tiempo de ejecución. Scala es un lenguaje de programación orientado a objetos puro, en el sentido de que cada valor es un objeto. Scala también posee características propias de los lenguajes funcionales, Scala está equipado con un sistema de tipos expresivo que refuerza a que las abstracciones de tipos se usen en forma coherente y segura. Scala se diseñó teniendo en mente el hecho de que en la práctica el desarrollo de aplicaciones requiere a menudo de extensiones específicas del lenguaje.



**ANTECEDENTES:**

El tema nos interesó ya que está relacionando con nuestra carrera además de darnos ventajas para hacer nuetros programas en Neatbeans 8.1 , se sabe que Scala es de gran ayuda para cualquier programador en Java ya que nos da otras ventajas y que incluso es gratuito . Apareció en el 2013 , su creador Martin Odersky dio a conocer este tipo de lenguaje de programación y tuvo influencia de [Smalltalk](https://es.wikipedia.org/wiki/Smalltalk), [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java), [Haskell](https://es.wikipedia.org/wiki/Haskell), [Standard ML](https://es.wikipedia.org/wiki/Standard_ML), [OCaml](https://es.wikipedia.org/wiki/Ocaml) , tiene licencia BSD fue diseñado para expresar patrones comunes de programación en forma concisa, elegante y con tipos seguros, con características de lenguajes funcionales orientada a objetos además de haberlo creado para ser compatible con Java y todos sus componentes.

****

INDICE

RESUMEN

ANTECEDENTES

DESARROLLO DEL TEMA:

I. CONCEPTO……………………………………………………………………….

II.CARACTERISTICAS……………………………………………………………..

III.VENEFICIOS………………………………………………………………………

IV. OBJETIVOS…........................................................................................................

VI. EJEMPLOS……………………………………………………………………

CASOS

RECOMENDACIONES

BIBLIOGRAFIA

ANEXOS

SCALA

1. CONCEPTO

¿Qué es Scala?

El nombre de Scala viene de “scalable” y “language”, esto indica cual es el propósito de este lenguaje. Scala se trata de un lenguaje de programación moderno multi-paradigma , el cual esta diseñado para expresar patrones de programación comunes de una forma concisa, elegante, y de tipado seguro. Combina fácilmente características de lenguajes orientados a objetos y funcionales; En otros lenguajes los objetos y las funciones son dos conceptos distintos, pero en Scala son dos conceptos entrelazados, por ejemplo, una función valor es un objeto. Este aspecto es muy útil para mejorar las escalabilidad de las aplicaciones.

Fue desarrollado por Martin Odersky , aproximadamente en 2004

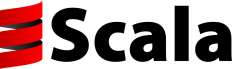
Ademas permite ejecutar programas tanto en modo compilado como en modo scripting.

Scala es un lenguaje orientado a objetos puro, donde todo es un objeto y todas las operación se realizan mediante una llamada a un método. Pero mientras Java tiene excepciones como los tipos primitivos o los statics, Scala no pose dichas excepciones.

Scala es funcional

En la programación orientada a objetos la composición y la herencia son los dos mecanismos básicos de rehúso. Scala ofrece un soporte excelente para el diseño correcto de los principios de abstracción fundamentales de la programación orientada a objetos: Clasificación/Instanciación, Agregación/Descomposición, Generalización/Especialización y Agrupamiento/Individualización.

Escala esta hecho para el lado del servidor; transacional, servicio, web; además es de mucho para interés para las empresas grandes las cuales tienen múltiples de usuarios, quienes realizan operaciones trasnacionales grandes



1. **CARACTERISTICAS**

Scala añade algunas características en la programación orientadas a objetos que no posee Java:

**Composición modular de mixin**. Este mecanismo que permite la composición de clases para el diseño de componentes reutilizables evitando los problemas presentados por la herencia múltiple. Similar a los interfaces Java y las clases abstractas. Por una parte se pueden definir múltiples “contratos” (del mismo modo que los interfaces). Por otro lado, se podrían tener implementaciones concretas de los métodos.

**Self-type.** Los mixin no dependen de ningún método y/o atributo de aquellas clases con las que se está entremezclando aunque en determinadas ocasiones será necesario hacer uso de las mismas. Esta capacidad es conocida en Scala como self-type.

**Abstracción de tipos.** Existen dos mecanismos principales de abstracción en los lenguajes de programación: la parametrización y los miembros abstractos. Scala soporta ambos estilos de abstracción de manera uniforme para tipos y valores.

Integra sutilmente características de lenguajes funcionales y orientados a objetos.

**Scala es también un lenguaje funcional** en el sentido que toda función es un valor. Scala provee una sintaxis ligera para definir funciones anónimas. Soporta funciones de primer orden, permite que las funciones sean anidadas, y soporta currying. Las clases caso de Scala y las construcciones incorporadas al lenguaje para reconocimiento de patrones modelan tipos algebráicos usados en muchos lenguajes de programación funcionales.

Además, la noción de reconocimiento de patrones de Scala se puede extender naturalmente al procesamiento de datos XML con la ayuda de patrones de expresiones regulares. En este contexto, seq comprehensions resultan útiles para formular consultas. Estas características hacen a Scala ideal para desarrollar aplicaciones como Web Services.

**Scala estáticamente tipado**

Scala cuenta con un expresivo sistema de tipado que fuerza estáticamente las abstracciones a ser usadas en una manera coherente y segura. En particular, el sistema de tipado soporta:

- Clases genéricas

- anotaciones variables,

- límites de tipado superiores e inferiores,

- clases internas y tipos abstractos como miembros de objetos,

tipos compuestos

- auto-referencias explicitamente tipadas

- implicit conversions

- métodos polimórficos

**El mecanismo de inferencia de tipos locales** se encarga de que el usuario no tengan que anotar el programa con información redundante de tipado. Combinadas, estas características proveen una base poderosa para el reuso seguro de abstracciones de programación y para la extensión segura (en cuanto a tipos) de software.

**Scala es extensible**

En la práctica el desarrollo de aplicaciones específicas para un dominio generalmente requiere de “Lenguajes de dominio específico” (DSL). Scala provee una única combinación de mecanismos del lenguaje que simplifican la creación de construcciones propias del lenguaje en forma de librerías:

cualquier método puede ser usado como un operador de infijo o postfijo

las closures son construidas automáticamente dependiendo del tipo esperado (tipos objetivo).

El uso conjunto de ambas características facilita la definición de nuevas sentencias sin tener que extender la sintaxis y sin usar facciones de meta-programación como tipo macros.

**Scala está diseñado para interoperar** bien con el popular Entorno de ejecución de Java 2 (JRE). En particular, la interacción con el lenguaje orientado a objetos Java es muy sencillo. Scala tiene el mismo esquema de compilación (compilación separada, carga de clases dinámica) que java y permite acceder a las miles de librerías de gran calidad existentes.

1. **VENTAJAS**

* En otros lenguajes los objetos y las funciones son dos conceptos distintos, pero en Scala son dos conceptos entrelazados, por ejemplo, una función valor es un objeto. Este aspecto es muy útil para mejorar las escalabilidad de las aplicaciones.
* Además permite ejecutar programas tanto en modo compilado como en modo scripting.
* Mientras Java tiene excepciones como los tipos primitivos o los statics; Scala no pose dichas excepciones.
* Scala ofrece un soporte excelente para el diseño correcto de los principios de abstracción fundamentales de la programación orientada a objetos: Clasificación/Instanciación, Agregación/Descomposición, Generalización/Especialización y Agrupamiento/Individualización.

**IV . EJEMPLOS:**

* CLASES
* Clases (estilo Java)



**CASOS:** Las clases son plantillas estáticas que pueden ser instanciadas por muchos objetos en tiempo de ejecución. Esta clase define la clase Point:

1. class Point(xc: Int, yc: Int) {
2. var x: Int = xc
3. var y: Int = yc
4. def move(dx: Int, dy: Int) {
5. x = x + dx
6. y = y + dy
7. }
8. override def toString(): String = "(" + x + ", " + y + ")";
9. }

Esta clase define dos variables x e y, y dos métodos: move y toString ,

Las clases son parametrizadas con argumentos constructores (inicializadores.En nuestro ejemplo son utilizados para inicializar las variables x e y.

Para instanciar una clase es necesario usar la primitiva new, como se muestra en el siguiente ejemplo:

1. object Classes {
2. def main(args: Array[String]) {
3. val pt = new Point(1, 2)
4. println(pt)
5. pt.move(10, 10)
6. println(pt)
7. }
8. }

El programa define una aplicación ejecutable a través del método main del objeto singleton Classes. El método main crea un nuevo Point y lo almacena en pt. *Note que valores definidos con la signatura*val*son distintos de los definidos con*var*(véase la clase*Point*arriba) ya que los primeros (*val*) no permiten reasignaciones; es decir, que el valor es una constante.*

Aquí se muestra la salida del programa:

1. (1, 2)
2. (11, 12)

**RECOMENDACIONES:**

**10 motivos por los que todo desarrollador necesita aprender Scala.**

## 1.Escalabilidad: ¿qué es eso? En resumidas cuentas, es la posibilidad de hacer crecer un negocio o proyecto sin tener que volver a invertir en infraestructura.

## 2.Funcional:  Esto hace que Scala sea especialmente una gran opción para las CPU con varios núcleos y para cargas de trabajo distribuidas

## 3.Orientado a objetos: Se puede  aprovechar las poderosas características orientadas a objetos como *traits* y *mixin-based composition*.

## 4. Productivo: dando la posibilidad a los desarrolladores de reducir el tamaño de código fuente

## 5. Lo tienes gratis: Lo gratis siempre tiene un sabor especial.

**BIBLIOGRAFIA:**

<http://www.scala-lang.org/docu/files/ScalaTutorial-es_ES.pdf> autores :Michel Schinz y Philipp Haller

<http://docs.scala-lang.org/es/tutorials/scala-for-java-programmers.html> fuente : wed

<https://en.wikipedia.org/wiki/Scala_(programming_language)> fuente : wed

<http://www.codecommit.com/blog/scala/scala-for-java-refugees-part-1> autor: Daniel Spiewak

<http://www.javamexico.org/blogs/ezamudio/un_caso_interesante_de_scala_vs_java> fuente : anónima

<https://keepcoding.io/es/10-motivos-por-los-que-debes-aprender-scala/> fuente : wed

**ANEXOS:**

**Scala vs Java**

**En Java:**

Un ejemplo :El alfabeto está compuesto de las letras A a la Z mayúsculas y luego minúsculas, los dígitos y los signos + y /. Existe además un símbolo especial = para indicar la ausencia de datos, cuando se codifica un bloque de menos de 3 bytes.

Cuando lo implementé en Java, lo primero que hice fue definir el alfabeto, como un arreglo de **char** Para ello, dentro de la clase que iba a contener los dos métodos (codificar y decodificar), hice esto:

**public** **class** Base64 {  
  **private** **static** **char**[] HEX\_CHARS = **new** **char**[]{  
    '0', '1', '2', '3', '4', '5', '6', '7',  
    '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'  
  };  
  ***/\*\* El arreglo de 64 caracteres que componen el 'alfabeto' de base 64. \*/***  
  **static** **final** **char**[] B64\_CHARS = **new** **char**[64];

**static** {  
    for (**int** i = 0; i < 26; i++) {  
      B64\_CHARS[i] = (**char**)(i + 65);  
    }  
    for (**int** i = 26; i < 52; i++) {  
      B64\_CHARS[i] = (**char**)(i + 71);  
    }  
    for (**int** i = 52; i < 63; i++) {  
      B64\_CHARS[i] = (**char**)(i - 4);  
    }  
    B64\_CHARS[62] = '+';  
    B64\_CHARS[63] = '/';  
  }  
}

Sesupone que era un algoritmo relativamente sencillo? ¿Por qué 27 líneas de código entonces? Este es uno de los problemas de la programación en estilo imperativo... estamos expresando exactamente no sólo lo que se tiene que hacer, sino cómo se tiene que hacer.

**Ahora en Scala:** Después de estar experimentando con varias opciones, ese tiempo fue el menor que pude lograr. La versión de Scala tarda 10 veces el tiempo que la de Java. Como mencioné anteriormente, este es uno de esos algoritmos en los que sí se tiene un beneficio muy notorio al poder manejar los tipos nativos de Java y hacer las operaciones de bits directamente. No es el caso típico de una aplicación; es de hecho el caso marginal en el que conviene tener el código en Java, por el performance.

**En conclusión:** No esque Java sea mejor, o que Scala sea mejor o peor, o que sea lento en general, etc.Creo que Java es una buena opción para implementar este tipo de algoritmos, pero Scala y Groovy, JRuby, etc son mucho mejores opciones para desarrollo general de aplicaciones.