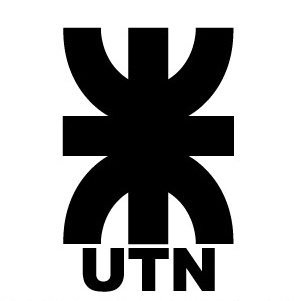
******

***Universidad Tecnológica Nacional***

**Sintaxis y Semántica de los Lenguajes**

**2023**

**Curso:** K2055

**Profesora:** Roxana Leituz

**Fecha Estipulada de Entrega: 25/08**

**Grupo N° 32**

|  |  |
| --- | --- |
| **Alumno** | **Legajo** |
| **Gonzalo Menese** | **173.087-3** |
| **Jean Pierre Luna Rumualdo** | **206.431-5** |
|  |  |
|  |  |

[Breve Historia Java 3](#_Toc143807918)

[Breve Historia Haskell 4](#_Toc143807919)

[BNF Java 6](#_Toc143807920)

[BNF Haskell 7](#_Toc143807921)

[Comparación de funciones básicas 8](#_Toc143807922)

[Bibliografía 12](#_Toc143807923)

# Breve Historia Java

Fue desarrollado en 1991 por James Gosling bajo el nombre “Oak”,con el propósito de ser un lenguaje para un control universal(Star 7), pero este proyecto no tuvo un buen recibimiento y Java fue lanzado en 1995 de forma gratuita por Sun mycrosistem con el objetivo que se volviera un estándar en la plataforma web y a la vez Microsoft (que dominaba el mercado tecnológico) no fuera capaz de responder.

Java se destacó como el único lenguaje en su tipo, debido a que no se compila a código máquina, sino a un lenguaje intermedio llamado BYTECODE que es interpreta por una JVM, esto hacía que, si nosotros instalásemos la JVM en cualquier sistema operativo, este tendría la posibilidad de ejecutar código Java

Combinaba los lenguajes de bajo nivel C y C++, pero con una sintaxis más sencilla.

# Breve Historia Haskell

Basada en cálculo lambda (de donde proviene su logo), consiste en una regla de transformación simple (sustituir variables) y un esquema simple para definir funciones.

En particular, es un lenguaje de tipos polimórficos (puede tomar mas de un tipo), de evaluación perezosa (no evalua una expresion o una funcion hasta que sea necesario), puramente funcional (o sea que priorizan el uso de funciones y recursidad para resolver problemas cuando otros lenguajes lo resolverían mediante estructuras de control (if, for , while )), por lo que es muy diferente de la mayoría de los otros lenguajes de programación.

Bases para la creación de Haskell

* En 1930 donde se remontan los orígenes teóricos del modelo funcional en los cuales Church propuso el cálculo lambda que se basó puramente en teorías de funciones.
* En 1950 McCarthy desarrollo el primer lenguaje funcional(Lisp) básado en cálculo lambda
* En 1960 Peter Landin desarrolló ISWIM que fue el primer lenguaje funcional puro totalmente basado en cálculo lambda
* En 1970 Jhon Backus desarrolló FP un lenguaje funcional con funciones de orden superior(funciones que aceptan funciones como argumentos) y razonamiento sobre programas
* En 1970 Robin Milner desarrolla el lenguaje ML, primer lenguaje funcional moderno, utiliza la inferencia de tipos y tipos polimórficos
* De 1970-1980 David Turner desarrolló varios lenguajes funcionales perezosos.
* En 1987 el comité internacional de investigadores inician el desarrollo de Haskell con el propósito de crear un lenguaje funcional, perezoso y estandarizado.
* En 2003 se consolido su versión mas estable y en 2010 se lanzó ya con mejores estándares, una actualización del compilador y un mayor soporte de librerías.

# BNF Java

<type declaration> ::= <class declaration> | <interface declaration> | ;

package declaration> ::= package <package name> ;

<constant expression> ::= <expression>

<if then statement>::= if ( <expression> ) <statement>

<while statement> ::= while ( <expression> ) <statement>

<for statement> ::= for ( <for init>? ; <expression>? ; <for update>? ) <statement>

# BNF Haskell

vars :: [context =>] type (type signature)

reservedid -> case | class | data | default | deriving | do | else | if | import | in | infix | infixl | infixr | instance | let | module | newtype | of | then | type | where | \_

pat1 | pat2 choice

topdecl -> type simpletype = type | data [context =>] simpletype = constrs [deriving] | newtype [context =>] simpletype = newconstr [deriving] | class [scontext =>] tycls tyvar [where cdecls] | instance [scontext =>] qtycls inst [where idecls] | default ( type1 , ... , typen) | decl

# Comparación de funciones básicas

**Factorial**

En Java…

Texto

Descripción generada automáticamente

Mientras que en Haskell…

Texto

Descripción generada automáticamente

Y ahora comparamos tiempo y memoria que usan en programa con una entrada de “1000”

Primero vamos con el tiempo y el promedio

|  |  |
| --- | --- |
| **Haskell** | **Java** |
| 1.03 secs | 89.460700 miliseg |
| 0.99 secs | 89.465000 miliseg |
| 0.99 secs | 162.534200 miliseg |
| 0.98 secs | 88.205700 miliseg |
| 1.02 secs | 90.790300 miliseg |
| Promedio: 1.002 | Promedio: 104.09118 |

Y ahora vamos con la memoria

|  |  |
| --- | --- |
| **Haskell** | **Java** |
| 108,455,400 bytes | 63918072 bytes |
| 108,455,144 bytes | 63901672 bytes |
| 108,455,176 byte | 63983576 bytes |
| 108,455,144 bytes | 63950840 bytes |
| 108,455,176 bytes | 63934440 bytes |
| Promedio: 108455208 bytes | Promedio:63937720 |

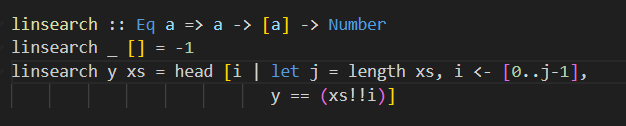
**Búsqueda**

En Java…

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Mientras que en Haskell…



Y ahora comparamos tiempo y memoria que usan en programa con una entrada de 5000 elementos

Primero vamos con el tiempo

|  |  |
| --- | --- |
| **Haskell** | **Java** |
| 0.02s | 1.1391ms |
| 0.01s | 1.1523ms |
| 0.01s | 1.1112ms |
| 0.01s | 1.1221ms |
| 0.01s | 1.1031ms |
| Promedio: 0.012s | Promedio: 1.14534ms |

Y ahora con la memoria que va a utilizar cada uno

|  |  |
| --- | --- |
| **Haskell** | **Java** |
| 427096 bytes | 3775872 bytes |
| 426248 bytes | 3775872 bytes |
| 426272 bytes | 3775872 bytes |
| 426248 bytes | 3775872 bytes |
| 425272 bytes | 3775872 bytes |
| Promedio: 426,227 bytes | Promedio: 3775872 bytes |

Sintaxis extra que necesita Java (una clase con un main donde arrancar a ejecutar)  
  


Una diferencia importante entre ambos lenguajes es la necesidad de tipado que posee Java frente a Haskell.

Como se ve claramente, Haskell no necesita que declaremos los tipos de funciones o variables ya que los deduce y agrega implícitamente por nosotros, mientras que Java requiere de que escribamos explicitamente el tipo de dato que vamos a utilizar tanto en funciones como en variables.

Otra gran diferencia como se puede ver es la necesidad de una clase con main por parte de Java para que la JVM sepa por donde empieza a ejecutar el codigo, mientras que Haskell ejecuta independiéntemente función a función sin necesidad de que le digamos por donde empieza.

Otras comparaciones importantes

Además de lo anteriormente mencionado, ambos lenguajes se utilizan para diferentes ambitos.

Mientras que Haskell es a menudo la elección clara en los campos que requieren un mapeo muy preciso de los algoritmos matemáticos, entre los que se incluyen las aplicaciones para la seguridad de red, los marcos de especificaciones para sistemas integrados o los programas para modelar cálculos matemáticos complejos, Java por otro lado, se utiliza mayormente en el ambito del Desarrollo web, videojuegos y aplicaciones de escritorio, elegido mayormente por su fuerte tipado, su nivel de confianza en estos ambitos (dado por sus años en ese mercado)

Preguntas para la clase

¿Por qué Java y Haskell no son comparables?

¿A qué se debió la popularización de Java?

¿En qué ámbito se utiliza mayormente Haskell?

# Bibliografía

<https://aws.amazon.com/es/what-is/java/#:~:text=Java%20se%20usa%20para%20motores,de%20datos%20en%20tiempo%20real>.

<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-haskell/#:~:text=Haskell%20es%20a%20menudo%20la,para%20modelar%20c%C3%A1lculos%20matem%C3%A1ticos%20complejos>.

<https://www.haskell.org/onlinereport/syntax-iso.html>

<https://cs.au.dk/~amoeller/RegAut/JavaBNF.html>

<https://ed.team/blog/que-es-y-para-que-sirve-java-guia-completa-para-programadores>

<https://labsys.frc.utn.edu.ar/ppr-2011/Unidad%20IV%20-%20Paradigma%20funcional/Unidad%20V%20-%20Paradigma%20Funcional.pdf>

<https://stackoverflow.com/questions/52518817/haskell-linear-search-returning-index>