Kotlin.

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Maestría en Computación
MC-8812 Teoría de los Lenguajes de Programación

CARLOS MARTÍN FLORES GONZÁLEZ, Carné: 2015183528

Índice

1. INTRODUCCIÓN

2. HISTORIA

La idea de Kotlin se concibió en el 2010 en la compañía JetBrains[1], fabricantes de herramientas de desarrollo para muchos lenguajes incluido `Java`, `C#`, `JavaScript`, `Python`, `Ruby` y `PHP`[1]. El Proyecto Kotlin fue revelado en Julio del 2011. Era un nuevo lenguaje para la *Java Virtual Machine* (JVM) que había estado en desarrollo por un año[2]. Dentro de la compañía se acusaba que muchos lenguajes de programación no contaban con las características que ellos estaban buscando, con la excepción de `Scala`, pero el lento tiempo de compilación de `Scala` era una deficiencia obvia. Uno de los objetivos de `Kotlin` era contar con un tiempo de compilación tan rápido como en el de `Java`.

De acuerdo con Dmitry Jemerov, ingeniero en Jetbrains [1]:

"La experiencia en la construcción de herramientas para una conjunto diverso de lenguajes nos ha dado una perspectiva y entendimiento único en el diseño de lenguajes pero, a pesar de esto varios de nuestros entornos de desarrollo estaban construidos en Java. Teníamos cierta envidia de nuestro equipo de .Net quienes desarrollaban en `C#`, un lenguaje rápido, moderno y de rápida evolución. Pero no encontrábamos ningún lenguaje que pudiéramos usar en lugar de Java.

¿Cuáles eran nuestro requerimientos para este lenguaje? Lo primero y lo más obvio era tipos estáticos. No conocemos ninguna otra forma de desarrollar miles de millones de líneas de código sin volvernos locos. Segundo, necesitábamos total compatibilidad con el código Java existente. El código es un gran activo para JetBrains, y no nos podíamos permitir perderlo o devaluarlo debido a dificultades en interoperatibilidad. En tercer lugar, no queríamos aceptar ningún compromiso en términos de calidad de herramientas. La productividad del desarrollador es el valor más importante para una compañía como JetBrains, y el contar con buenas herramientas es esencial para lograr esto. Por último, necesitábamos un lenguaje que fuera fácil de leer y de razonar. Con esto en mente nos decidimos a embarcarnos en un proyecto para crear un nuevo lenguaje: Kotlin".

`Kotlin v1.0` fue lanzado en Febrero del 2016[3]. En Google `I/O` 2017, Google anunció soporte para `Kotlin` en `Android`[4]. `Kotlin v1.2` se lanzó en Noviembre del 2017. Compartir código entre la JVM y `JavaScript` fue una característica agregada a esta version[5].

Kotlin lleva el nombre de una isla cerca de San Petersburgo[2], Rusia, donde se encuentra la mayor parte del equipo de desarrollo de Kotlin[1].

3. CARACTERÍSTICAS DEL LENGUAJE

3.1. Valores y tipos de datos base

En Kotlin, todo es un objeto en el sentido que se pueden invocar funciones y propiedades en cualquier variable. Algunos de los tipos pueden tener una representación interna especial (por ejemplo, números, caracteres y booleanos pueden ser representados como tipos primitivos en tiempo de ejecución) pero que para el usuario lucen como una clase ordinaria.

3.1.1. Números. Kotlin proporciona las siguientes tipos para representar números:

```
Double 64bit, Int 32bit, Float 32, Short 16bit, Long 64bit, Byte 8bit
```

3.1.2. Caracteres. Son representados por el tipo `Char`. No pueden ser tratados directamente como números. Los literales van en comillas simples: `'1'`. Caracteres especiales pueden ser escapados usando un *backslash*. Las siguientes secuencias de escape son soportadas:

```
\t, \b, \n, \r, \', \", \\, \$
```

3.1.3. Booleanos. El tipo `Boolean` representa booleanos y tiene dos tipos `true` y `false`.

[1] https://www.jetbrains.com/
[2] https://en.wikipedia.org/wiki/Kotlin_Island

3.1.4. Arrays. Los *Arrays* en Kotlin se representan por la clase `Array`, que tiene la propiedad `size` y funciones `get`, `set`, entre otras que son útiles para trabajar con este tipo de estructura.

3.1.5. Strings. Se representan con la clase `String`. Los *String* son inmutables. Los elementos de un `String` son caracteres que puede ser accedidos por la operación de indexación `s[i]`. Un `String` puede ser recorrido por medio de un ciclo `for-loop`.

3.1.6. String Literals. Kotlin tiene dos tipo de *string literals*: *strings* escapados que pueden tener caracteres escapados en él y *strings* crudos (*raw*) que pueden contener varias líneas y texto arbitrario.

```
val s = "Hello, world!\n"
val text = """
    for (c in "foo")
        print(c)
"""
```

3.1.7. String Templates. Los *strings* pueden contener expresiones emplantilladas, piezas de código que puede ser evaluadas y cuyos resultados se concatenan dentro del *string*.

```
val i = 10
print("i = \$i") //imprime "i = 10"
```

3.2. Variables y almacenamiento

Para declarar una variable en Kotlin, se inicia con el nombre del identificador se puede o no poner el tipo luego del nombre

```
val question = "The Ultimate Question of Life, the Universe, and Everything"
val answer: Int = 42
```

Si no se especifica el tipo, el compilador analiza la expresión y su inicializador, y usa el tipo como el tipo de variable[1]. En el ejemplo anterior, `42`, el inicializador tiene un tipo `Int`, por lo tanto la variable va a tener el mismo tipo. Si la variable no se inicializa, se necesita especificar su tipo explícitamente:

```
val answer: Int
answer = 42
```

Existen dos palabras reservadas para declarar una variable:

1. **`val`** (de valor – *value*). Referencia Inmutable. Una variable declarada cn `val` no puede ser reasignada luego de ser inicializada. Corresponde a una variable final en Java.
2. **`var`** (de variable). Referencia mutable. El valor de una variable puede ser cambiado. Esta declaración corresponde a un variable regular (no final) en Java.

Almacenamiento. Debido a que Kotlin es un lenguaje construido por encima de la JVM delega el manejo del almacenamiento de variables, estructuras de datos y de control a esta. Una breve introducción sobre el manejo de la memoria en la JVM se puede encontrar en Apéndice 1.

3.3. Estructuración de datos

3.4. Declaraciones, manejo del alcance de los identificadores

3.4.1. Declaración de tipos. Nuevos tipos son introducidos en Kotlin a través de clases, `Data Classes`, `Enum Classes`, interfaces y objetos.

Declaración de una clase.

```
class Invoice {
```

```
}
// Clase con constructor primario
class User(_nickname: String) {
    val nickname = _nickname
}
//
val invoice = Invoice()
val user = User("Luigi")
```

Data Classes. Clases cuyo propósito principal es contener datos. En una clase de este tipo, algunas funcionalidades estándar y funciones de utilidad a menudo se derivan mecánicamente de los datos.

```
data class User(val name: String, val age: Int)
val admin = User("John", 40)
```

El compilador automáticamente deriva los siguientes miembros par todas las propiedades declaradas en el constructor primario: `equals()/hashCode()`, `toString()`, `copy()`. Para asegurar la consistencia y significado del código generado, las *data classes* tiene que tener los siguietes requisitos:

- El constructor primario necesita tener al menos un parámetro
- Todos los parámetros del constructor primario necesitan estar marcados con `val` o `var`
- *Data classes* no pueden ser abstractas, internas o selladas (*sealed*).

Enum Classes.

```
enum class Color {
    RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET
}
//
val primaryColor = Color.GREEN
```

Interfaces. Interfaces en Kotlin son similares a las de Java8: pueden contener definiciones de métodos abstractos así como implementaciones de métodos no abstractos (similar a los métodos *default* de Java8), pero no pueden contener ningún estado.

```
interface Clickable {
    fun click()
}

class Button : Clickable {
    override fun click() = println("I was clicked")
}
val submitBtn : Clickable = Button()
submitBtn.click()

// Interfaz con implementaciones
interface Focusable {
    fun setFocus(b: Boolean) =
        println("I \${if (b) "got" else "lost"} focus.")

    fun showOff() = println("I'm focusable!")
}
```

Objetos.

3.4.2. Declaración de constantes. Las propiedades cuyo valor se conoce en tiempo de compilación se pueden marcar como constantes de tiempo de compilación (*compile time constants*) utilizando el modificador `const`. Estas propiedades necesitan tener los siguientes requerimientos:

- Tiene que ser un miembro de primer niven en un *object*
- Inicializado con un valor de tipo `String` o un valor primitivo
- Sin un *getter* personalizado

```
const val COUNTRY: String = "Costa Rica"
```

3.4.3. Declaración de variables.

3.4.4. Declaración de funciones y procedimientos.

- La palabra reservada `fun` se utiliza para declarar una función.
- El tipo de parámetro se escribe luego de su nombre. Esto aplica también para declaraciones.
- La función se puede declarar en cualquier parte del archivo, no se necesita poner dentro dentro de una clase.

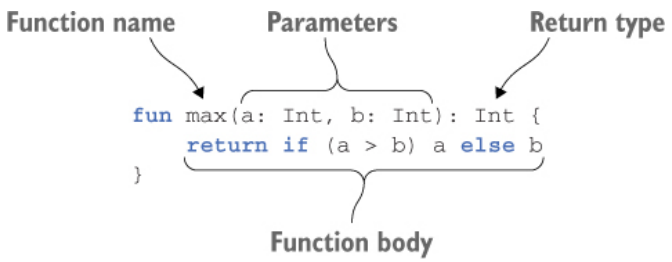

Fig. 1. Declaracion de una función en Kotlin. Fuente [1]

La declaración de la función inicia con la palabra clave `fun`, seguida del nombre de la función: `max` en en caso del ejemplo de la figura 1. El tipo de retorno viene luego de la lista de parámetros, separado por dos puntos. Nótese que en Kotlin, `if` es una expresión con un valor como resultado. Es similar como al operador ternario de Java:

```
(a > b) ? a : b
```

3.5. Estructuras de control

Expresión `if`. Uso tradicional

```
var max = a
if (a < b) max = b
```

Utilizando `else`.

```
var max : Int
if (a > b) {
    max = a
} else {
    max = b
}
```

Como expresión.

```
val max = if(a > b) a else b
```

3.5.1. Expresión `when`. reemplaza al operador *switch* de lenguajes basados en `C`. Evalua el argumento contra todos los caso de manera secuencial hasta que se cumpla una de las condiciones. El caso `else` se evalua si ninguno de los otros casos se pudo satisfacer. Puede utilizar expresiones arbitrarias(no solo constantes) y también se puede evaluar si un valor se encuentra dentro de un rango.

```
when (x) {
    1          -> print("x == 1")
    2          -> print("x == 2")
    3, 4       -> print("x == 3 or x == 4")
    in 5..10   -> print("x is between 5 and 10")
    !in 20..30 -> print("x is outside the range")
    else -> {
        print("none of the above")
    }
}
```

3.5.2. Ciclos `for`. Tiene una forma equivalente al ciclo `for-each` de Java.

```
for (item: Int in ints) {
    print(item)
}
```

Se puede iterar sobre un rango de números:

```
for (i in 1..3) {
    println(i)
}
for (i in 6 downTo 0 step 2) {
    println(i)
}
```

3.5.3. Ciclos `while`. Tanto el ciclo `while` como el `do..while` trabajan de la misma forma que en Java:

```
while (x > 0) {
    x--
}

do {
    val y = retrieveData()
} while (y != null) // y es visible aquí
```

3.6. Secuenciadores

Kotlin tiene tres expresiones para saltos/continuaciones:

- `return`: retorna de la función más cercana
- `break`: termina un ciclo
- `continue`: continua con el siguiente paso de un ciclo

`break` *y* `continue` *con etiquetas.* Cualquier expresión en Kotlin puede ser etiquetada. Las etiquetas tienen un identificador seguido del signo `@`. Para etiquetar una expresión, se punta una etiqueta en frente de ella

```
loop@ for (i in 1..100) {
    //...
}
```

Se puede marcar un `break` o un `continue` con una etiqueta:

```
loop@ for (i in 1..100) {
    for (j in 1..100) {
        if (...) break@loop
    }
}
```

Un `break` marcado con una etiqueta salta al punto de ejecución justo después del ciclo que fue marcado con esa etiqueta. Un `continue` prosigue con la próxima iteración del ciclo.

Excepciones. Todas las excepciones en Kotlin son descendientes de la clase `Throwable`. Todas las excepciones tienen un mensaje, un *stack trace* y una causa (opcional). Para lanzar una excepción se usa la expresión `throw`

```
throw MyException("Hi There!")
```

Para atrapar una excepción, se usa la expresión `try`:

```
try {
    // algún código
} catch (e: SomeException) {
    // código manejador de excepción
} finally {
    // bloque finalizador opcional
}
```

Pueden haber uno o varios bloques `catch`. Los bloques `finally` pueden ser omitidos.

3.7. Mecanismos de abstracción

3.8. Mecanismos de modularización

3.9. Soporte a concurrencia, paralelismo, distribución

3.10. Sistemas de tipos

Como se señalo en la sección 3.4, nuevos tipos son introducidos en Kotlin a través de clases, Data Classes, Enum Classes, interfaces y objetos.

Modificadores de acceso para clases: `open`, `final` y `abstract`.

- `final`: No puede ser sobre escrita. Utilizada por defecto para los miembros de clase
- `open`: Puede ser sobre escrita. Tiene que se especificada explícitamente
- `abstract`: Debe ser sobre escrita. Puede ser utilizada solamente en clases abstractas, los miembros abstractos no pueden tener una implementación
- `override`: sobre escribe un miembro de una super clase o interfaz. El miembro sobreescrito está abierto por defecto, sino está marcado como `final`.

Modificadores de visibilidad:

- `public` (por defecto). Visible en todas partes
- `internal`. Visible en un módulo

- `protected`. Visible en subclases
- `private`. Visible en una clase

3.11. Genericidad

3.12. Soporte a paradigmas

3.13. Soporte a "programación en grande"

3.14. Pecularidades

4. EJEMPLOS

5. ANÁLISIS Y CONCLUSIONES

REFERENCIAS

[1] Dmitry Jemerov, Svetlana Isakova. *Kotlin in Action*. Manning Publications Company. 2017. ISBN 9781617293290

[2] Krill, Paul *JetBrains readies JVM language Kotlin*. infoworld.com. InfoWorld. 2011. Obtenido el 11 de Abril del 2018 de https://www.infoworld.com/article/2622405/java/jetbrains-readies-jvm-based-language.html.

[3] Kotlin Blog. *Kotlin 1.0 Released: Pragmatic Language for JVM and Android*". Febrero, 2015. Obtenido el 11 de Abril del 2018 de https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/

[4] Kotlin Blog. *Kotlin on Android. Now official*. Mayo, 2017. Obtenido el 11 de Abril del 2018 de https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/.

[5] Kotlin Blog. *Kotlin 1.2 Released: Sharing Code between Platforms*. Noviembre, 2017. Obtenido el 11 de Abril del 2018 de https://blog.jetbrains.com/kotlin/2017/11/kotlin-1-2-released/.