PROFUNDIZANDO EN KOTLIN

**Funciones de extensión**

### ¿De qué se trata?

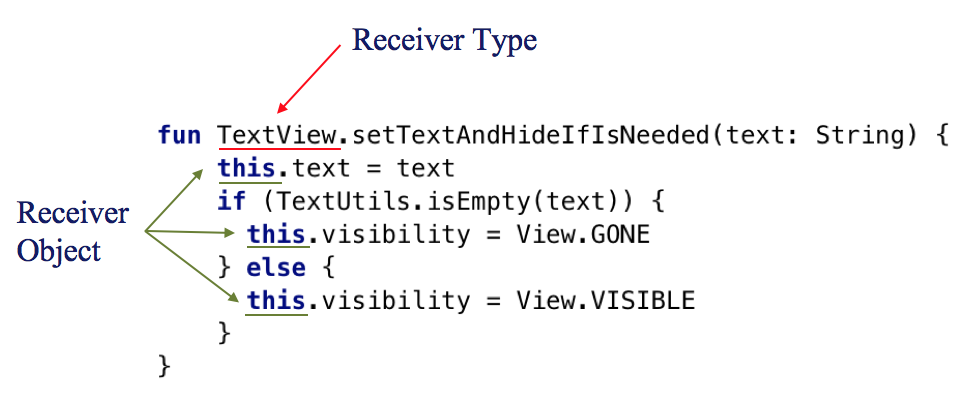
Es una función que agrega un nuevo comportamiento a una clase y lo mejor… ¡Hasta sin tener acceso al código fuente!

### Ventajas

* Añadimos funciones extras a clases ya existentes
* No es necesario pasar el objeto como argumento
* Actúa como si perteneciese a la clase, por lo que podemos usar sus métodos públicos
* Pueden añadir también propiedades

### Estructura

Dos elementos:

* **Receiver Type**. Clase a la que se añade la función
* **Receiver Object**. Valor al que llama la función  
  Ejemplo:  
  

### Imports

Deben ser importadas estáticamente una por una o utilizando el **\***.  
Normalmente se agrupan por contexto funcional en una clase con nomenclatura Extension. Ej: ViewExtension.kt  
No necesitan que sea una clase ni nada parecido.

### 

### ¿Se pueden usar desde Java?

Sí, simplemente tenemos que usarla como si fuese un método estático y pasarle como argumento el objeto que la llama.

**Colecciones**

### Métodos de creación

| **Tipos** | **Inmutables** | **Mutable** |
| --- | --- | --- |
| List | listOf | mutableListOf, arrayListOf |
| Set | setOf | mutableSetOf, hashSetOf, linkedSetOf, sortedSetOf |
| Map | mapOf | mutableMapOf, hashMapOf, linkedMapOf, sortedMapOf |

### 

### Interoperabilidad con Java

Al pasar una colección de Kotlin a Java, ésta puede ser editada e incluso se pueden introducir nulos en la misma.

### Arrays de tipos primitivos

Kotlin proporciona clases específicas para los arrays de tipo primitivo:  
IntArray, ByteArray, CharArray, BooleanArray, etc.

### Arrays de tipos primitivos

Formas de crear Arrays

* Pasándole sólo el tamaño del mismo.
* val ints = IntArray(10)
* Pasando directamente el valor de cada elemento.
* val ints: IntArray = intArrayOf(1, 2, 3)
* Pasando el tamaño y una lambda para inicializar cada elemento
* val ints = IntArray(10) {i -> i\*2}

### 

### Operaciones comunes con colecciones

Operaciones

* **Filter**. Filtra los elementos de una lista incluso eliminando los elementos pero no puede editarlos.
* **Map**. Igual que filter pero ésta si puede editar los elementos.
* **All**. Comprueba si todos los elementos cumplen o no un predicado.
* **Fold**. Acumula empezando con el valor inicial y aplicando la operación pasada mediante una lambda para cada elemento.
* **Any**. Comprueba si algún elemento cumple el predicado pasado por parámetro.
* **Count**. Retorna el número de elementos que cumple el predicado.
* **Find**. Retorna el primer elemento que cumple el predicado o null si no hay ninguno.
* **Max o min**. Retorna el valor máximo o mínimo de un listado (null si la lista está vacía).
* **Partition**. Retorna dos listas, una con los elementos que cumplen el predicado y otra con los que no.
* **ElementAtOrNull**. Retorna el elemento o null si el índice está fuera de rango.
* **Sort, sortBy, reverse**, etc. Métodos de ordenación de los elementos.

**Operador *when***

### Definición

Similar al switch/case pero mucho más potente y frecuente de ver en el código.  
Tanto en el argumento como en los case se puede usar cualquier cosa.  
Un uso muy frecuente son los smart cast.

### Características

* El caso por defecto se define con la estructura else
* Como es una expresión, puede retornar un resultado también
* Las condiciones pueden ser un conjunto de valores separados por comas
* El argumento es automáticamente casteado en cada condición
* En las condiciones se pueden usar rangos
* Se puede usar para sustituir cadenas de if / else

**Control de excepciones**

### Diferencias con Java

* No tiene checked exceptions
* **try** es una expresión, por lo que se puede usar para retornar valores

val count: Int = try {

number!!.count()

} catch (e: Exception) {

0

}

* **throw** también es una expresión por lo que se puede usar en las expresiones **elvis**

number?.let { println(number) } ?: error()

### 

### Tipo Nothing

El tipo de la expresión throw es el tipo Nothing. Se usa para marcar una función que nunca retorna, por ejemplo que siempre acaba dando una excepción.

fun error(): Nothing {

throw RuntimeException()

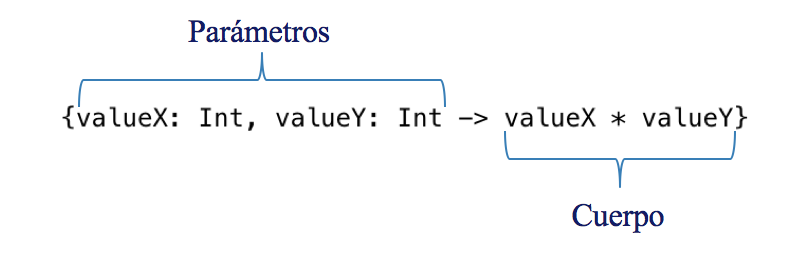
}

**Lambdas**

### ¿Qué es una función Lambda?

Es una forma simple de definir una función anónima. Es posible usar una función como parámetro de otra.

### Sintáxis

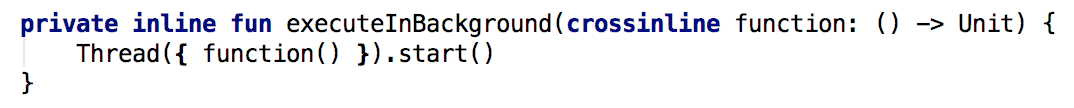
* Siempre va encapsulado entre llaves
* La lista de parámetros no necesita ir entre paréntesis
* La flecha separa la lista de parámetros del cuerpo de la lambda  
  

### Inline y crossline

* Función **Inline**. No consumen tantos recursos ya que en tiempo de compilación el compilador la sustituirá por el código y no creará clases anónimas.
* Función **Crossline**. Se usa cuando una función será llamada desde otra lambda.

### 

### Uso frecuente

Ejecutar tarea en segundo plano  
  
  
  
Invocar a la función  
Ejemplo

### Funciones de orden superior y lambdas (Higher-Order functions)

Se tratan de funciones que toman funciones como parámetros o retornan una función.