



Programación multimedia y dispositivos móviles.
(PGL)
U.T. N° 1

INTRODUCCIÓN

- Kotlin es un lenguaje de programación, similar a Java que ha visto incrementada su fama desde que Google lo utiliza para su desarrollo Android. De tipado estático, para la JVM y desarrollado por JetBrains.
- Es 100% compatible con Java y cualquier librería Java se puede usar desde Kotlin, o al revés, hacer una librería en Kotlin y usarla desde Java.
- Y entre otros muchos beneficios, para el desarrollador, uno de los más destacables es que evita el NullPointerException.
- Puedes desarrollar con Kotlin en los IDEs Android Studio, IntelliJ IDEA, IntelliJ o Eclipse.



EJEMPLO HOLA MUNDO

- Las líneas del código anterior tienen el siguiente significado:
- La función `main()` es el punto de entrada de una aplicación Kotlin. Ella ejecuta todas las instrucciones que pongas en su interior. Toma como parámetros los argumentos de línea de comandos en un array de strings.
- La función `println()` imprime en consola el texto pasado como argumento, en este caso "Hello World!".
- La llave de cierre `}` especifica el final del cuerpo de `main()`
- A diferencia de Java, un archivo Kotlin, no necesita una clase declarada explícitamente, para definir piezas de código.
- La palabra reservada **fun** va a identificar una función.

```
fun main(args: Array<String>) {  
    println("Hello World!")  
}
```

Hola Mundo versión anterior a la 1.3.

```
fun main() {  
    println("Hello World!")  
}
```

Hola Mundo a partir de la versión 1.3.

FUNDAMENTOS BÁSICOS DE KOTLIN

● Operadores:

- ☑ Los ya conocidos `+`, `-`, `*` y `/`, con similar funcionamiento si se opera con enteros o decimales.
- ☑ Kotlin considera los números como primitivas, aunque se permite acceder a métodos como si fueran objetos.
- ☑ En Kotlin se utiliza `val` y `var` para conseguir que nuestras variables sean de solo lectura (lo que era `final` en Java) o mutables (el resto de variables a los que se modifica el valor).
- ☑ `==` es el operador de comparación. Existen también el `greater`, `not equal`... similares a otros lenguajes.

● Strings: de similar funcionamiento a otros lenguajes.

FUNDAMENTOS BÁSICOS DE KOTLIN

- If - else y When:

- ☑ If testea una condición. Además permite rangos.
- ☑ When es similar al switch-case:

- Array: Existen muchas alternativas para trabajar con Collections. Vemos la más común de ArrayList.

- ☑ Los array pueden contener elementos de diferente tipo.

```
var nota=8

when (nota) {
    in 0..4 -> println("suspendido")
    in 5..9 -> println("aprobado")
    10 -> println("Mención honorífica")
    else -> println("no presentado")
}

aprobado
|Ctrl+Intro> to execute
```

```
var milista= arrayOf("Ana","Luis")

println(milista.asList())
[Ana, Luis]
|Ctrl+Intro> to execute
```


```
var listamixta=arrayOf("Uno",2,"Tres",4)

println(listamixta.asList())
[Uno, 2, Tres, 4]
```

FUNDAMENTOS BÁSICOS DE KOTLIN

- Bucles:

- ☑ Puede implementar los for, while y do while como los conoces.
Ejemplo para el array milista con 2 elementos .

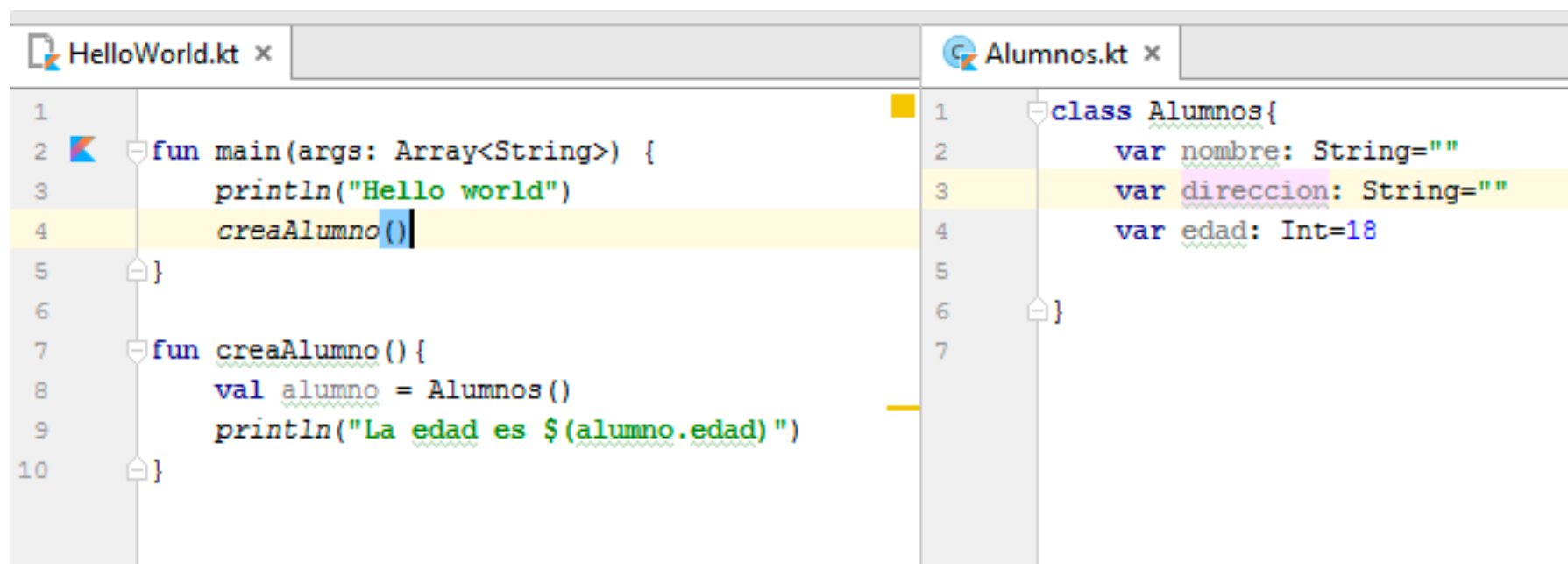


```
for (item in milista)
    println(item+"---")
Ana---Luis---
```

FUNDAMENTOS BÁSICOS DE KOTLIN

● Objetos:

- ☑ En un fichero Kotlin puede existir más de un main y se seleccionará cuál ejecutar. El desarrollo de clases en Kotlin es similar a lo conocido en Java, con algunas excepciones. Veamos algunos conceptos de la POO.
- ☑ En Kotlin, no se instancia un objeto con new()
- ☑ Habrá que declarar la clase, atributos, constructor principal y/o secundarios.



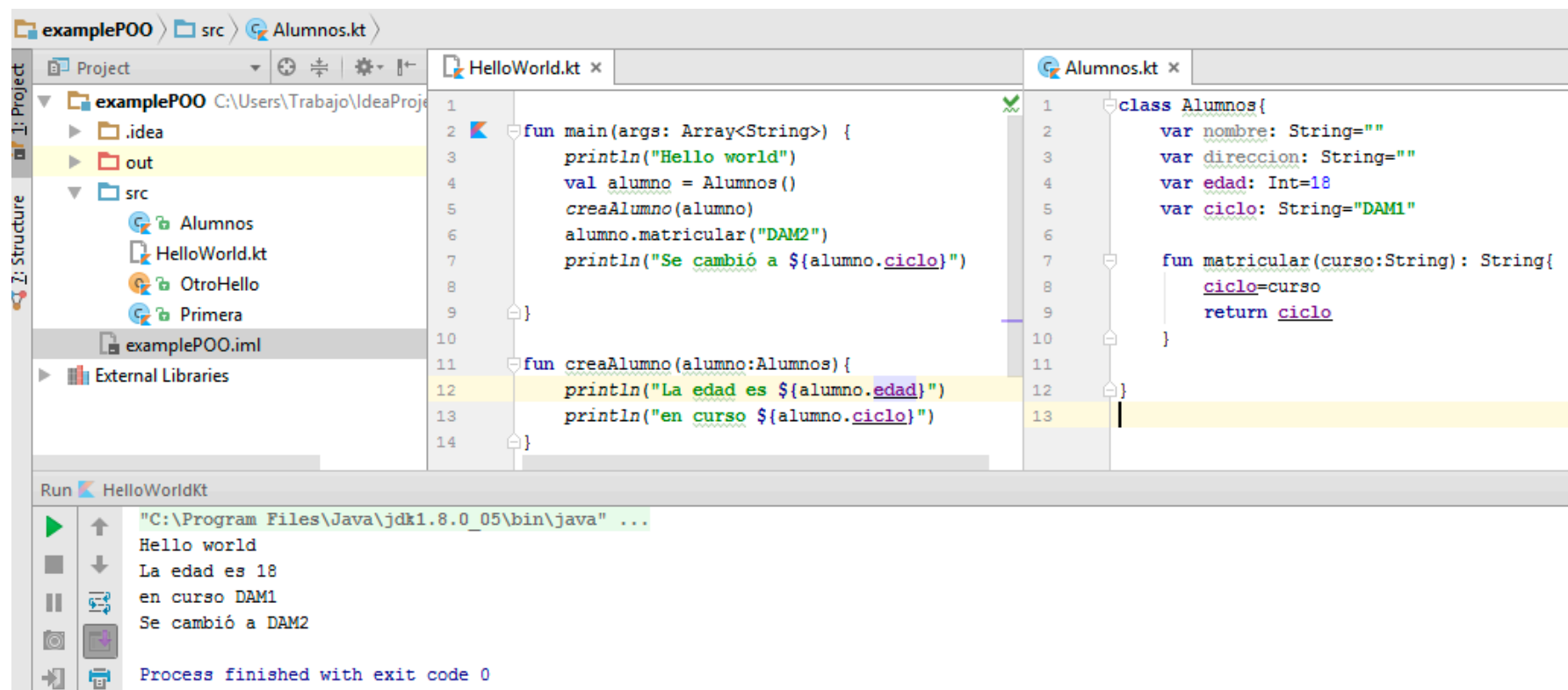
```

HelloWorld.kt x
1
2 fun main(args: Array<String>) {
3     println("Hello world")
4     creaAlumno()
5 }
6
7 fun creaAlumno() {
8     val alumno = Alumnos()
9     println("La edad es ${(alumno.edad)}")
10 }

Alumnos.kt x
1 class Alumnos{
2     var nombre: String=""
3     var direccion: String=""
4     var edad: Int=18
5
6 }
7
```

FUNDAMENTOS BÁSICOS DE KOTLIN

- Cuando definimos las variables o propiedades en una clase, se declaran la revés... es decir, primero el nombre, luego el tipo y siempre deben estar inicializadas o ser abstractas.
- Ejemplo:
 - ☑ `var nombre: String = ""`
- La declaración de métodos, similar, poniendo al final el tipo de datos que se devuelve (nada si es void)



```
examplePOO > src > Alumnos.kt
Project
  examplePOO C:\Users\Trabajo\IdeaProje
    .idea
    out
    src
      Alumnos
      HelloWorld.kt
      OtroHello
      Primera
    examplePOO.iml
  External Libraries

1
2 fun main(args: Array<String>) {
3     println("Hello world")
4     val alumno = Alumnos()
5     creaAlumno(alumno)
6     alumno.matricular("DAM2")
7     println("Se cambió a ${alumno.ciclo}")
8 }
9
10
11 fun creaAlumno(alumno: Alumnos) {
12     println("La edad es ${alumno.edad}")
13     println("en curso ${alumno.ciclo}")
14 }

Alumnos.kt
1 class Alumnos {
2     var nombre: String = ""
3     var direccion: String = ""
4     var edad: Int = 18
5     var ciclo: String = "DAM1"
6
7     fun matricular(curso: String): String {
8         ciclo = curso
9         return ciclo
10    }
11 }
12
13

Run HelloWorldKt
"C:\Program Files\Java\jdk1.8.0_05\bin\java" ...
Hello world
La edad es 18
en curso DAM1
Se cambió a DAM2
Process finished with exit code 0
```


HERENCIA

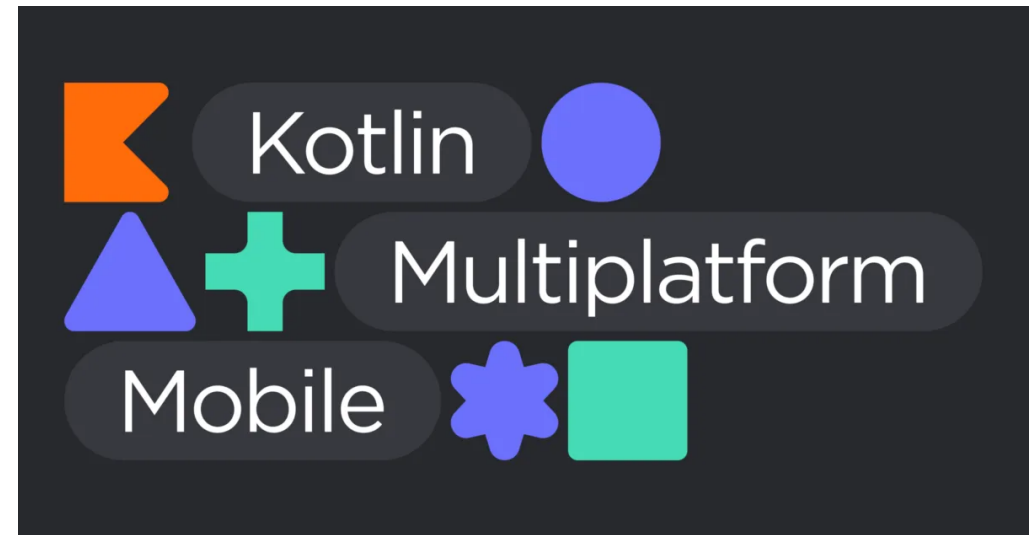
- La clase de la que se derivan atributos o propiedades debe ser declarada como Open. Ésta deriva de Any (lo que es Object en Java), pero no es necesario incluirlo de forma explícita.
- La clase que extiende sólo debe añadir el nombre de la clase padre, como tipo.
- Por ejemplo, Personas y estudiantes que son personas
- Usamos constructores primarios y quedaría así:

```
open class Personas (dni: String, nombre: String, direccion: String){  
}  
  
class Estudiantes(dni: String, nombre: String, direccion: String, ciclo: String, cial:String): Personas(dni, nombre, direccion){  
}
```

CLASES ABSTRACTAS E INTERFACES.

- La cláusula `abstract` se coloca delante del nombre de la clase y el funcionamiento es similar a Java.
- Y lo mismo para la implementación de interfaces.

```
open class Personas (dni: String, nombre: String, direccion: String){  
}  
  
class Estudiantes(dni: String, nombre: String, direccion: String, ciclo: String, cial:String): Personas(dni, nombre, direccion), Tareas {  
    override fun horaslibres() {  
        throw UnsupportedOperationException("not implemented") //To change body of created functions use File | Settings | File Templates.  
    }  
}  
  
interface Tareas{  
    fun horaslibres()  
}
```



KOTLIN MULTIPLATFORM

- Es una tecnología open source creada por JetBrains para simplificar los desarrollos multiplataforma. Gracias a KMP vamos a poder realizar una única app y exportarla en distintos entornos como Android, iOS, Desktop (Windows, Linux y Mac), Web e incluso en servidor

¿CÓMO FUNCIONA KOTLIN MULTIPLATFORM?

- Podemos preparar nuestros proyectos de distintas formas, todo depende de nosotros, ya que la idea de Kotlin Multiplatform o KMP es poder adaptarlo a cualquier tipo de situación por lo que podríamos dividirlo en tres formas distintas de configuración.
 - **Compartir partes de lógica:** la forma menos intrusiva o más suave. Por ejemplo en el caso de Android y iOS podríamos tener cada una de las app en su lenguaje (Swift para iOS y Kotlin para Android) pero tener parte de la lógica (escrita en Kotlin) separada y utilizarlas en ambas plataformas.
 - **Separar las UIs y compartir toda la lógica:** El segundo nivel sería tener TODA la lógica en Kotlin y compartida y simplemente tener las vistas de cada plataforma en nativo, por ejemplo Android con Jetpack Compose y iOS con SwiftUI.
 - **Reutilizar el 100% del código:** Esto se conoce como Kotlin Compose Multiplatform y la idea es que desarrollando una app podamos exportar el 100% del proyecto en cualquier plataforma.