

{desafío}
latam_

Kotlin y Android _

Parte I

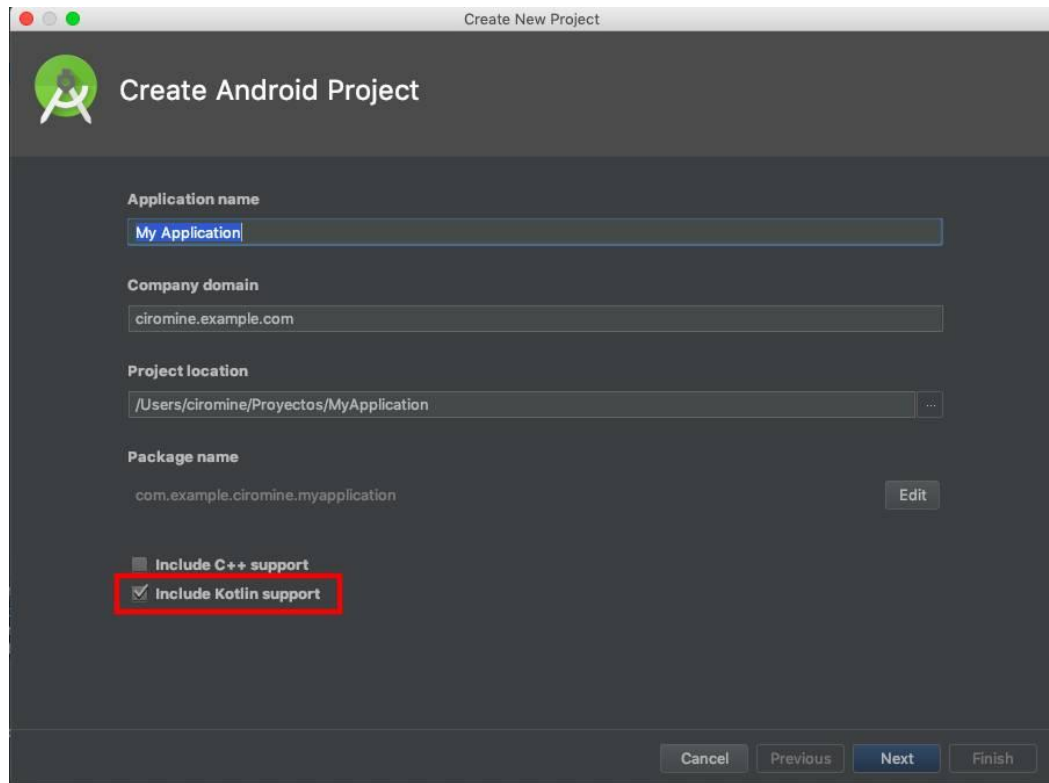


Iniciando con Kotlin

¿Qué aprenderemos?

- Integrar Kotlin en Android Studio
- Aprender a crear clases Kotlin
- Convertir Clases Java a Kotlin

Iniciando un proyecto Android con Kotlin



Create New Project

Create Android Project

Application name
My Application

Company domain
ciromine.example.com

Project location
/Users/ciromine/Proyectos/MyApplication

Package name
com.example.ciromine.myapplication Edit


☐ Include C++ support

☒ Include Kotlin support

Cancel Previous Next Finish

Iniciando un proyecto Android con Kotlin

Create New Project

 Target Android Devices

Select the form factors and minimum SDK
Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

☒ **Phone and Tablet**
API 21: Android 5.0 (Lollipop) ▼

By targeting **API 21 and later**, your app will run on approximately **85.0%** of devices. [Help me choose](#)

☐ Include Android Instant App support

☐ **Wear**
API 21: Android 5.0 (Lollipop) ▼

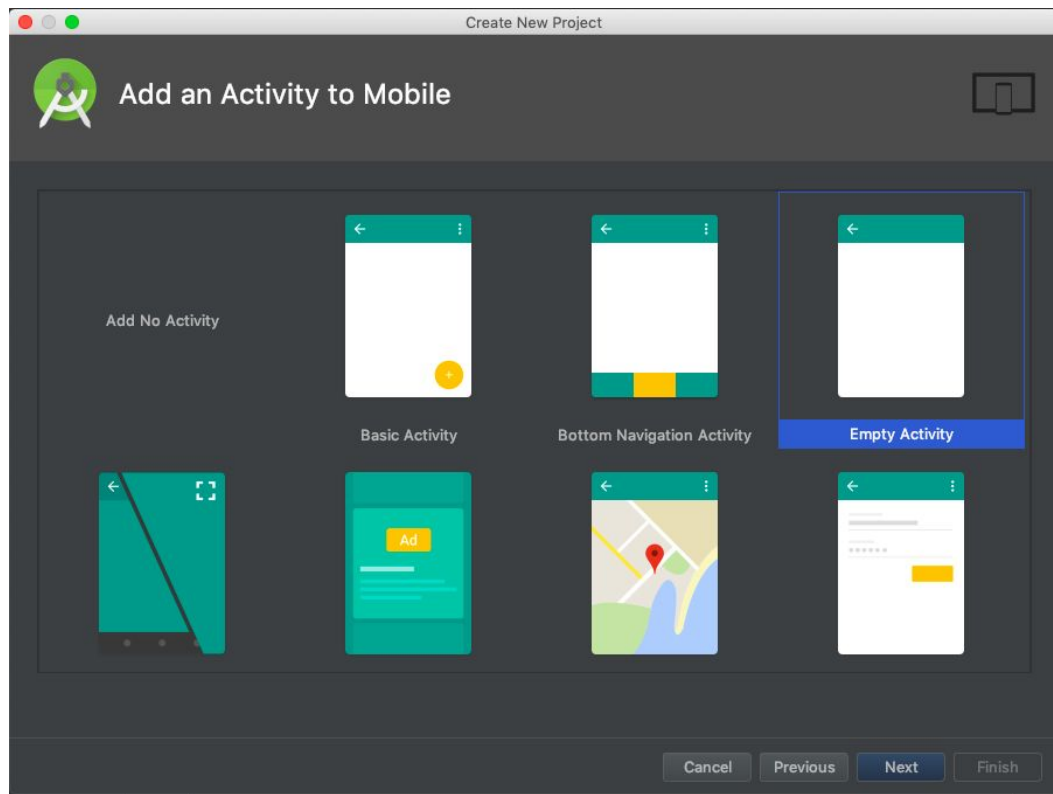
☐ **TV**
API 21: Android 5.0 (Lollipop) ▼

☐ **Android Auto**

☐ **Android Things**
API 24: Android 7.0 (Nougat) ▼

Cancel Previous **Next** Finish

Iniciando un proyecto Android con Kotlin

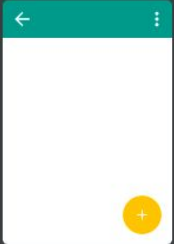


Iniciando un proyecto Android con Kotlin

Create New Project

Configure Activity

Creates a new basic activity with an app bar.



Activity Name
MainActivity

Layout Name
activity_main

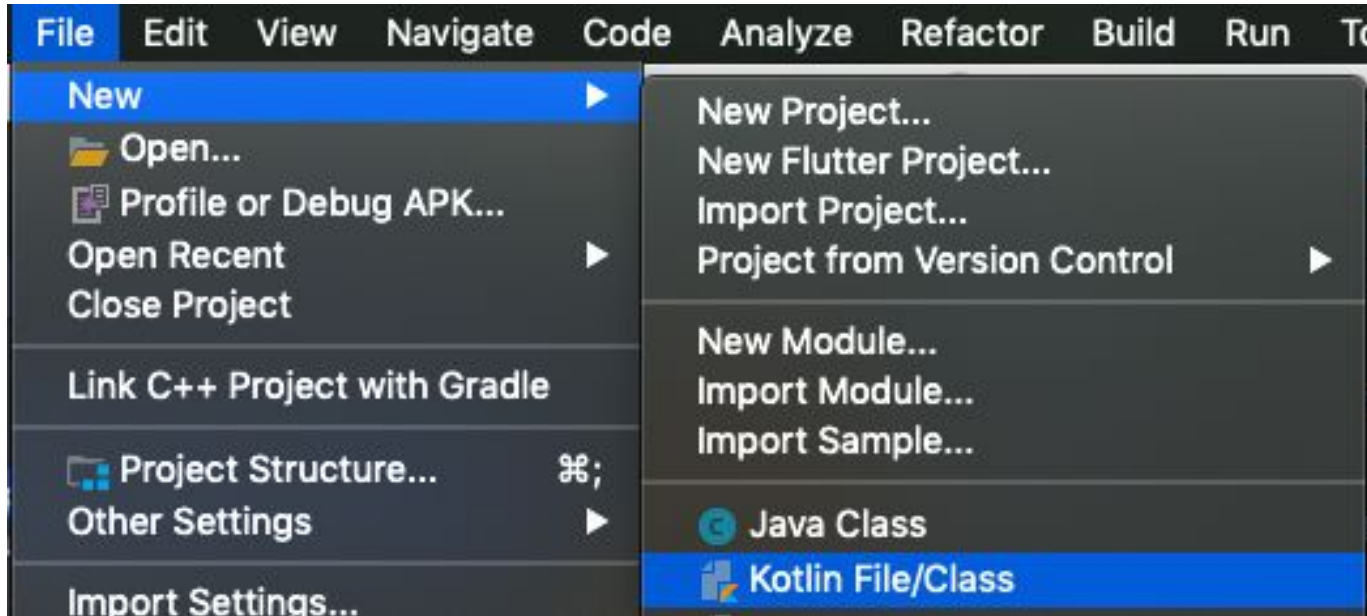
Title
MainActivity

☐ Use a Fragment

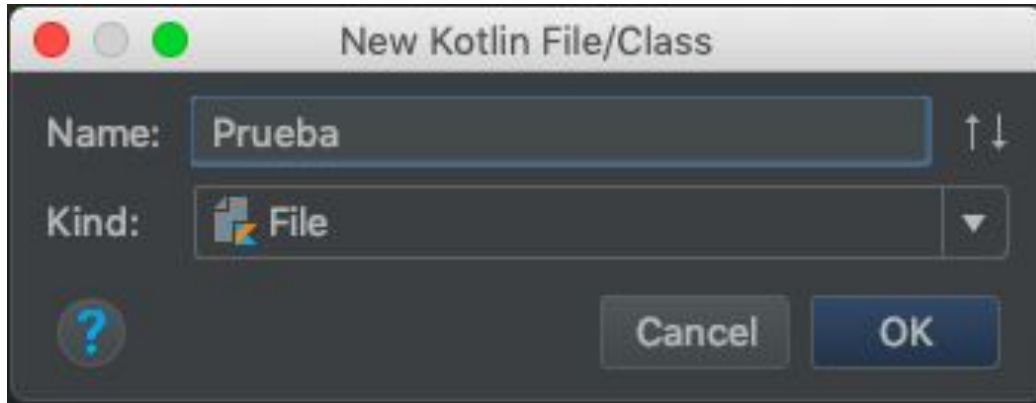
Hierarchical Parent

Cancel Previous Next Finish

Creando clases Kotlin



Creando clases Kotlin

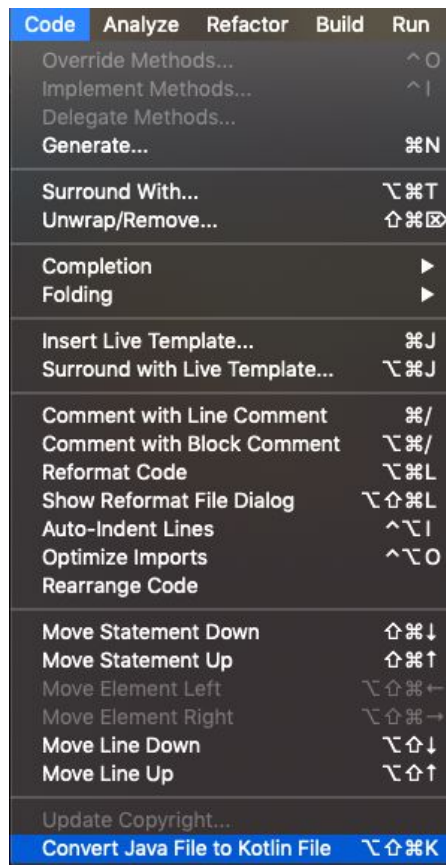


Convirtiendo una clase Java en Kotlin

Veamos la siguiente clase java.

```
public class Calculadora {  
  
    public Calculadora() {}  
  
    public int suma(int a, int b) {  
        return a + b;  
    }  
}
```

Creando clases Kotlin



Convirtiendo una clase Java en Kotlin

Veamos la siguiente clase java.

```
public class Calculadora {  
  
    public Calculadora() {}  
  
    public int suma(int a, int b) {  
        return a + b;  
    }  
}
```

Convirtiendo una clase Java en Kotlin

Se ha convertido en esta clase Kotlin de
manera automática.

```
class Calculadora {
```

```
    fun suma(a: Int, b: Int): Int {
```

```
        return a + b
```

```
    }
```

```
}
```

Kotlin y Java

¿Qué aprenderemos?

- Interoperabilidad Android y Kotlin

¿Qué es la interoperabilidad?

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) define interoperabilidad como la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.

Interoperabilidad Java Kotlin

Usando la misma clase java que
teníamos antes.

```
public class Calculadora {
```

```
    public Calculadora() {}
```

```
    public int suma(int a, int b) {
```

```
        return a + b;
```

```
    }
```

```
}
```

Interoperabilidad Java Kotlin

Podemos llamarla desde una clase kotlin y utilizarla sin problemas.

También podemos llamar clases Kotlin desde Java.

```
val textview =  
findViewById<TextView>(R.id.textview) as TextView
```

```
val calculadora = Calculadora()
```

```
textview.text = "Tu resultado es: " +  
calculadora.suma(1,1)
```

```
}
```

Ventajas de usar Kotlin

¿Qué aprenderemos?

- Utilizar sentencia When
- Usar Null Safety
- Manejar operadores

Sentencia When

Esta es la estructura básica de la sentencia when.

```
when (valor){
```

```
    //serie de condiciones
```

```
}
```

Sentencia When

Supongamos tenemos el siguiente caso donde tenemos que usar muchos if y else, seguimos para evaluar distintos casos.

```
if (numero == 0) {  
    println("numero invalido")  
}  
else if (numero == 1 || numero == 2) {  
    println("numero muy bajo")  
}  
else if (numero == 3) {  
    println("numero correcto")  
}  
else if (numero == 4) {  
    println("numero alto, pero aceptable")  
}  
else {  
    println("numero muy alto")  
}  
}
```

Sentencia When

Con la sentencia when, podríamos hacer lo mismo que el código anterior, pero de manera más simple y legible.

```
when(numero) {
```

```
    0 -> println("numero invalido")
```

```
    1, 2 -> println("numero muy bajo")
```

```
    3 -> println("numero correcto")
```

```
    4 -> println("numero alto, pero aceptable")
```

```
    else -> println("numero muy alto")
```

```
}
```

Sentencia When

También podemos asignar el resultado a una variable y utilizarla después.

```
var result = when(number) {  
  
    0 -> "numero invalido"  
  
    1, 2 -> "numero muy bajo"  
  
    3 -> "numero correcto"  
  
    4 -> "numero alto, pero aceptable"  
  
    else -> "numero muy alto"  
  
}  
  
// it prints when returned "Number too low"  
  
println("Retorno \"$result\"")
```


Sentencia When

Podemos evaluar distinto tipos de datos, como por ejemplo booleanos.

```
val check = true
```

```
val result = when(check) {  
    true -> println("it's true")  
    false -> println("it's false")  
}
```

Sentencia When

Podemos usar también rangos, como los que usamos en la sentencia for, para ayudarnos en nuestras condiciones.

```
var result = when(number) {  
  
    0 -> "numero invalido"  
  
    1, 2 -> "numero muy bajo"  
  
    3 -> "numero correcto"  
  
    in 4..10 -> "numero alto, pero aceptable"  
  
    else -> "numero muy alto"  
  
}
```

Sentencia When

Podemos evaluar el tipo de variable que es, como en este ejemplo, tenemos distintos casos si es Int, String o IntArray.

```
when (x) {
```

```
    is Int -> print("es un Entero")
```

```
    is String -> print("Es un String")
```

```
    is IntArray -> print("Es un arreglo de  
Enteros")
```

```
}
```

Null Safety

¿Que es un un error de null?

- Null o nulo (en español) es literalmente un valor vacío, cuando iniciamos una variable en Java como por ejemplo un String y hacemos **String a;**
- pero no le asignamos un valor, por defecto esta variable **a** se le asigna un valor nulo
- que sería equivalente a un valor vacío
- Por eso cuando una variable es nula y tratamos de hacer alguna operación sobre ella, podemos obtener un `NullPointerException`.

Null Safety

- Kotlin es un lenguaje de programación que evita el uso de **null**.
- Es por esto que kotlin, tiene una manera segura para trabajar con nulls, comenzaremos a explicar ahora en qué consiste.

Null Safety

En estos 2 ejemplos, tendríamos un error de sintaxis, ya que no podemos asignar a una variable un valor null en kotlin.

```
var palabra: String = "abc"
```

```
palabra = null
```

```
var palabra: String = null
```

Null Safety

Para poder asociar variables null, debemos usar el operador **?**, como vemos en los 2 ejemplos.

Esto nos permitirá declarar null y sin que el IDE nos reclame un error de sintaxis.

```
var palabra: String? = "abc"
```

```
palabra = null
```

```
var palabra: String? = null
```

Null Safety

Podemos usar el operador `?.` para evitar nulos, en este caso, la variable `palabra` es `null`, no generará error al preguntar por `length` siempre que usemos el operador.

```
var palabra: String? = null
```

```
palabra?.length
```


Null Safety

Otro ejemplo un poco más complejo que el anterior, pero viendo el mismo uso del operador ?

```
class Perro {  
    var raza: Raza? = null  
}
```

```
class Raza {  
    var nombre: String? = null  
}
```

```
var perro = Perro()  
perro?.raza?.nombre
```

Null Safety

También existe otro operador !! que nos sirve para indicarle al IDE que la variable **NO ES** null.

```
class Perro {  
  
    var raza: Raza? = null  
  
}
```

```
class Raza {  
  
    var nombre: String? = null  
  
}
```

```
perro.raza!!.nombre
```

Null Safety

También podemos usar el operador `?`, para indicar que una lista puede contener nulos.

```
val listaDeRazas: List<String?> =  
listOf("bulldog", "labrador", null, "pitbull")
```

{desafío}
latam_

*Academia de
talentos digitales*

www.desafiolatam.com