ANDROID MEETS KOTLIN

**Android Studio y Kotlin**

### ¿Es necesario algún plugin?

No, Kotlin está totalmente soportado en Android Studio 3.0. Puedes usar todas sus herramientas, autocompletar, etc.  
Además, es posible usar un “conversor” de Java a Kotlin.

### ¿Cómo hacer que nuestro proyecto lo soporte?

Al crear un nuevo proyecto es necesario marcar el check que indica “Include Kotlin Support”.  
También es posible crear clases en Kotlin en los proyectos existentes.

### Conversor

Simplemente tenemos que copiar código Java en nuestro fichero Kotlin. Seguidamente saldrá un diálogo que nos preguntará si queremos que nos convierta nuestro código.  
Consejo: Siempre lo usaremos de guía pero no es recomendable fiarnos 100% ya que suele cometer errores.

### Crear recursos en Kotlin

Simplemente tendremos que hacer doble click y luego: New -> Kotlin file/class. Posteriormente simplemente seleccionaremos el tipo de archivo y su nombre.

**Bind de Vistas**

### Propiedades de extensión

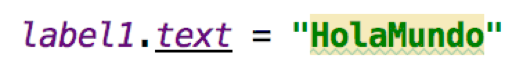
Las propiedades de extensión evitarán las llamadas repetitivas al método findViewById al crear las propiedades usando su id.

### ¿Cómo las usamos?

Accederemos a la vista a través de una propiedad de la clase nombrada con el mismo id que tenía en el layout.

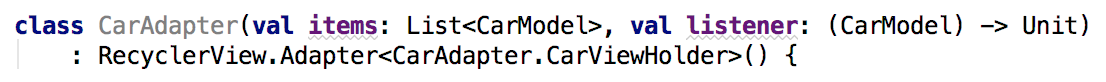
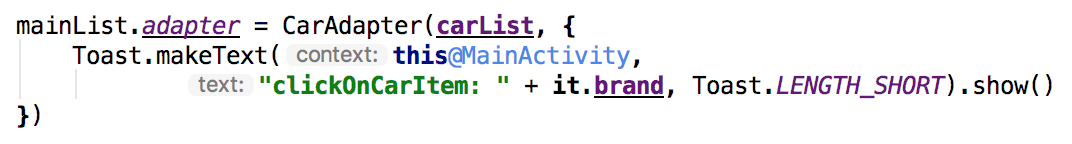
### 

### Ejemplo

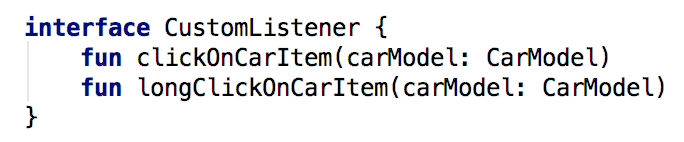
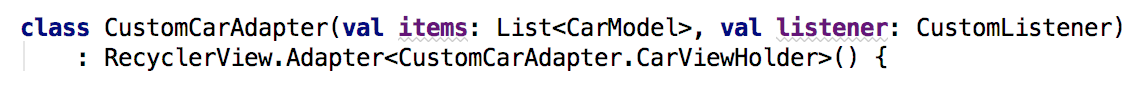
Tenemos que importar todas nuestras propiedades de extensión:  
  
Import de vistas  
  
Hacemos referencia al TextView a través de su id:  


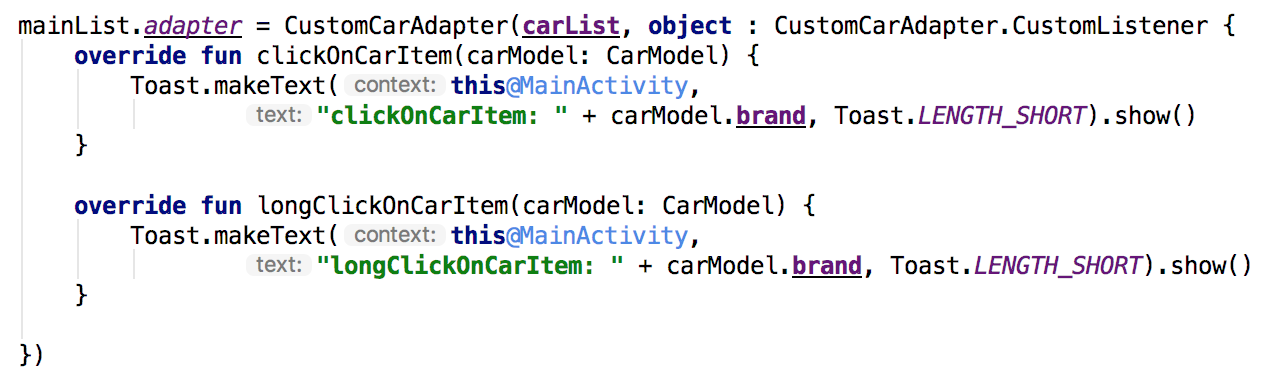
**Definición de Listener**

**Un solo método**

Si la interfaz sólo tiene un método es mejor usar una lambda  
  
  
  
Establecemos dicho adapter al RecyclerView, recibirá una lambda para la implementación del click.  
  


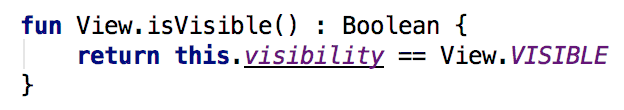
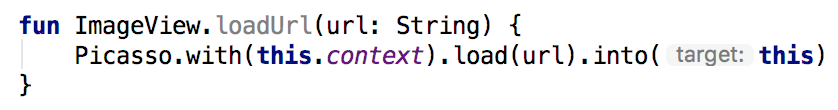
**Más de un método**

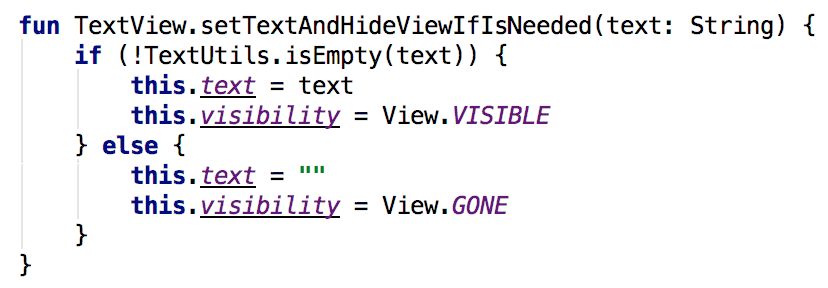
En este caso sería necesario crear un objeto de la interfaz.  
  
Creamos un adapter que reciba como objeto dicha interfaz.  
  


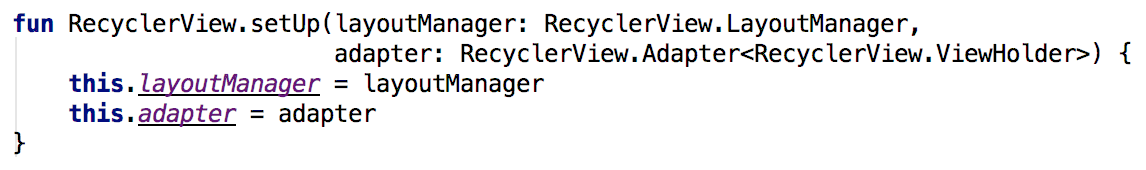
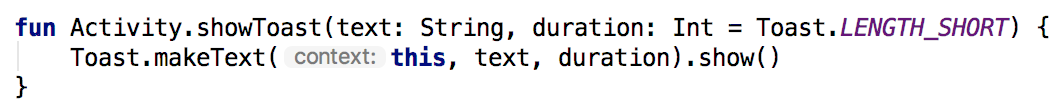
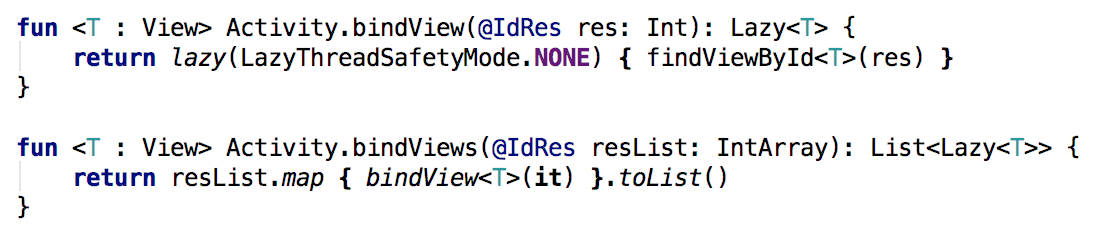
Establecer el CustomCarAdapter al RecyclerView e implementar sus dos métodos.  
  


**Funciones de extensión**

**Utilidades**

Crearemos funciones extensión que añadirán funcionalidades a clases ya existentes.  
  
**View**  
Comprobaremos si una vista es visible o no.  
  
**ImageView y Picasso**  
Conseguiremos cargar una imagen de un servicio con un sólo método.  
  
**TextView**  
Al establecer un texto a un TextView lo haremos visible o invisible en función de si es nulo/vacío o no.



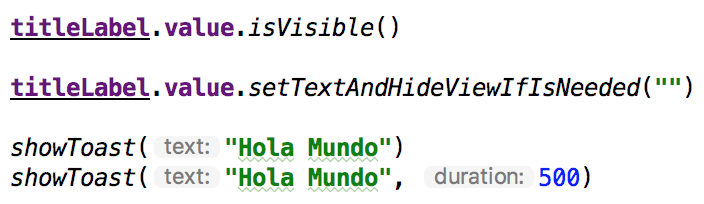
**RecyclerView**  
Configuración en un sólo método.  
  
  
 **Activity**  
Función para mostrar un toast.  
  
  
  
Bindearemos vistas para solventar los problemas con los nombres de los ids. Además podemos mejorar el rendimiento al usar funciones *lazy*.  
  


**¿Dónde las declaramos?**

Crearemos un archivo con la palabra Extension al final.  
Ej: ActivityExtension.kt  
ViewExtension.kt

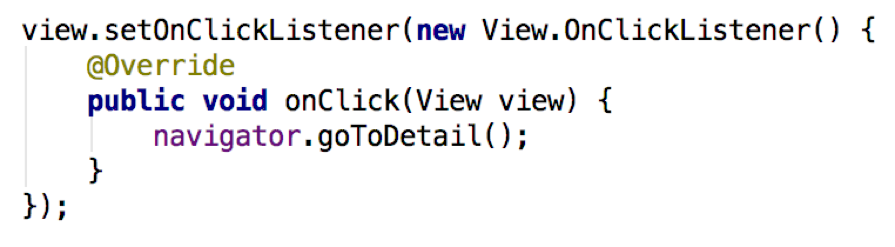
**¿Cómo las usamos?**

El *import* necesario  
import

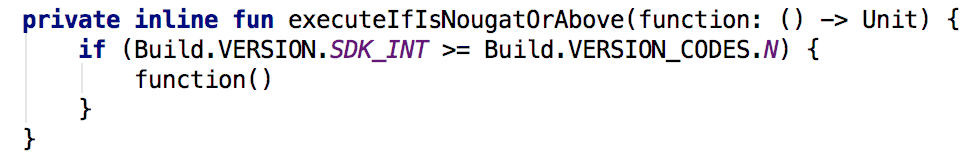
Ejemplo de uso:  
  
uso 1  


**Lambdas**

### setOnClickListener

**Java**  
Código repetitivo cada vez que queremos setear un click a una vista.  
  
**Kotlin**  
Iremos simplificando el click usando una lambda hasta que quede totalmente reducido.   
  
**Pasos**

1. Tiene un parámetro de tipo vista llamado view. En el cuerpo de la lambda escribiremos el métodonavigateToDetail() que irá separado por una flecha del listado de parámetros.   
   Paso 1
2. Puesto que el parámetro no se usa, podemos eliminarlo y al no tener más parámetros, podemos eliminar también la flecha.   
   Paso 2
3. Al ser la lambda el último parámetro, podemos sacarla de los paréntesis.   
   Paso 3
4. Puesto que no hay parámetros y la lambda ya la tenemos fuera, podemos eliminar los paréntesis.  
   Paso 4

Salvando la fragmentación Ejecutar función si es Nougat o superior  
  
  
Ejemplo de uso

### Ejecutar una función u otra

Si el número es par ejecutaremos la función executeIfIsPair, en caso contrario ejecutaremos executeIfNot.  
  
