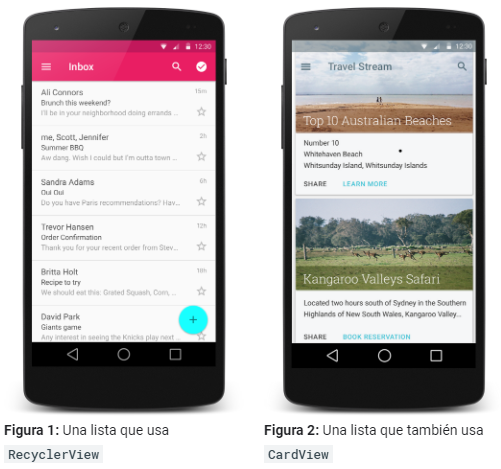
**RECYCLERVIEW**

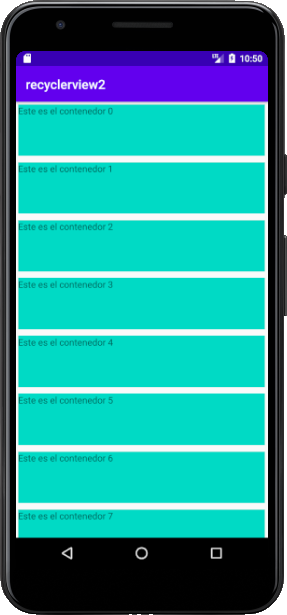
Se trata de un componente para mostrar grandes conjuntos de datos que se pueden desplazar de manera muy eficiente al mantener una cantidad limitada de vistas. Aquí vemos dos ejemplos.



Para conseguir esto necesitamos que varios componentes trabajen juntos.

* El contenedor general de la interfaz será el recyclerview. Suele ser un fragment que insertamos en nuestra actividad. Sus funciones son:
  + Crear solo la cantidad de objetos contenedores de vistas que son necesarios para mostrar la parte de la pantalla más algunos más (para visualizarlos al desplazarnos).
  + A medida que el usuario se desplaza por la vista, crea nuevos contenedores de vistas según sea necesario.
  + Guarda los contenedores de vistas que se desplazaron fuera de la pantalla para que puedan volver a usarse. Si el usuario cambia la dirección de desplazamiento, se pueden recuperar los contenedores de vistas que se desplazaron fuera de la pantalla.
  + Vincula con datos nuevos los contenedores de vistas que salieron de la pantalla mucho antes. La app simplemente actualiza el contenido de la vista para que coincida con el elemento nuevo con el que está vinculada.
* Los objetos contenedores de vistas serian cada uno de los elementos donde se guardan los datos. Son instancias de una clase que extiende de RecyclerView.ViewHolder. Tendremos que crear un layout y además una clase dentro de la clase adaptador.
* El adaptador administra los objetos contenedores, es decir, creo los objetos contenedores necesarios y los vincula a sus datos. Un adaptador es una instancia de una clase que extiende de RecyclerView.Adapter y para realizar estas funciones es necesario que implemente tres métodos.
  + onCreateViewHolder: Infla o coloca la vista en el contenedor de vistas, es decir, hace que el layout se muestre en el contenedor de vista.
  + onBindViewHolder: Enlaza los datos al contenedor de vistas. Para ello usa la posición del contenedor de vistas.
  + onItemCount: Devuelve el número de datos, es decir, cuantos contenedores de vistas necesitamos.

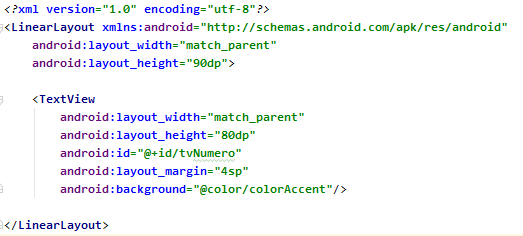
Aquí están los pasos a seguir para poder crear el RecyclerView que vemos en esta imagen



1. Dentro del Gradle, en el del módulo, debes implementar la librería del RecyclerView para poder trabajar con ellos.

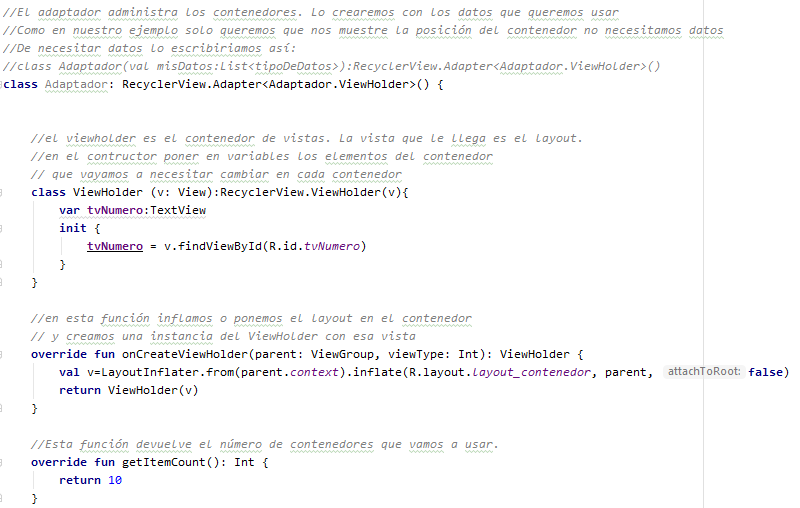


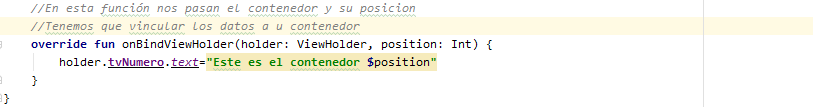
1. Vamos a crear un layout de los contenedores de vistas. Para ello clickamos con el botón derecho en la carpeta layout, new>XML>Layout XML file. Yo le voy a llamar layout\_contenedor y va a tener un único textview donde aparecerá el número de cada contenedor.



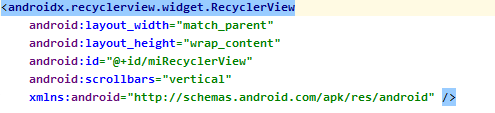
He usado un LinearLayout porque es el que aparece por defecto, pero podríais usar el constraintlayout.

1. Una vez creado el layout del contenedor de vistas, vamos a crear el adaptador que va a ser el encargado de administrador los objetos contenedores. Como hemos dicho el adaptar va a ser una clase que extienda de RecyclerView.Adapter. Para hacerla clickamos con el botón derecho en la carpeta donde se encuentra nuestra MainActivity. New> Kotlin File/class. Yo le voy a poner por nombre Adaptador.

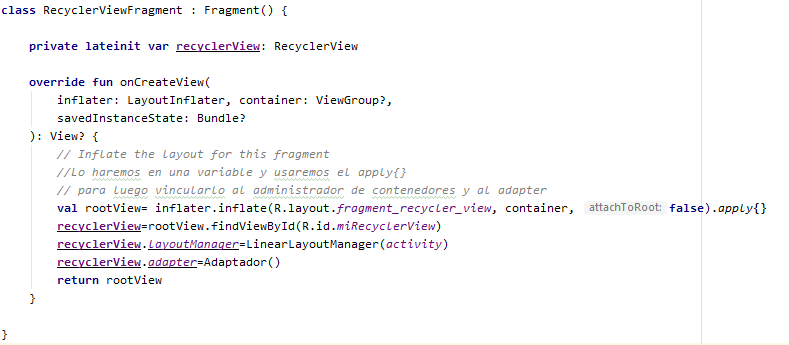




1. Lo siguiente será crear el RecyclerView. Para crearlo tenemos que crear un fragment. Yo lo voy a llamar RecyclerViewFragment.
   1. En el xml de este fragment tendremos que insertar el elemento recyclerView.



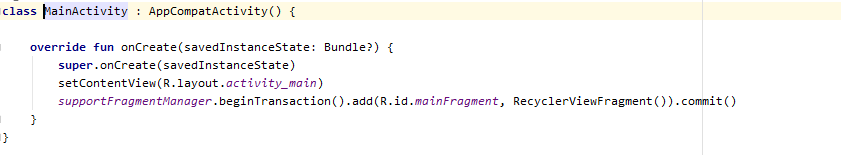
* 1. En el código además de inflar el fragment tendremos que inicializar los datos que queremos que usen los contenedores de vistas, en nuestro caso esto no tenemos que hacerlo, ya que como datos vamos a usar la posición del contenedor. Además, tendremos que vincular al RecyclerView un administrador de diseño (LinerlayoutManager, GridLayoutManager…) y el adapter.



1. Por último, en el la interfaz gráfica de la actividad principal crearemos un elemento FrameLayout y desde el código insertaremos el fragment con el recyclerView en el.
   1. El xml



* 1. El código o .kt



**CARDVIEW**

Con frecuencia, las apps deben mostrar datos en contenedores con un estilo similar. Para ello tenemos las CardView que muestran la información dentro de una tarjeta que tienen un aspecto coherente en la plataforma. Estas tarjetas tienen una elevación predeterminada por encima del grupo de vistas que las contiene, de modo que el sistema dibuja sombras por debajo de ellas.

Para usar el CardView lo primero de todo será implementar la librería en el gradle de nuestro modulo.



Ahora podremos insertar CardView como elementos de nuestro entorno gráfico.

En este ejemplo queremos crear una tarjeta con la imagen de un delfín, el nombre, y la descripción. Se vera así.



Como veis el CardView crea una sombra, y podemos redondearle las esquinas con el atributo cardCornerRadius.

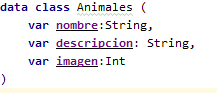
Para ello en el xml pondremos el elemento CardView, y dentro del resto de elementos(o vistas) que vayamos a usar. En nuestro caso una imagen (ImageView), el nombre del animal (TextView) y la descripción (TextView).



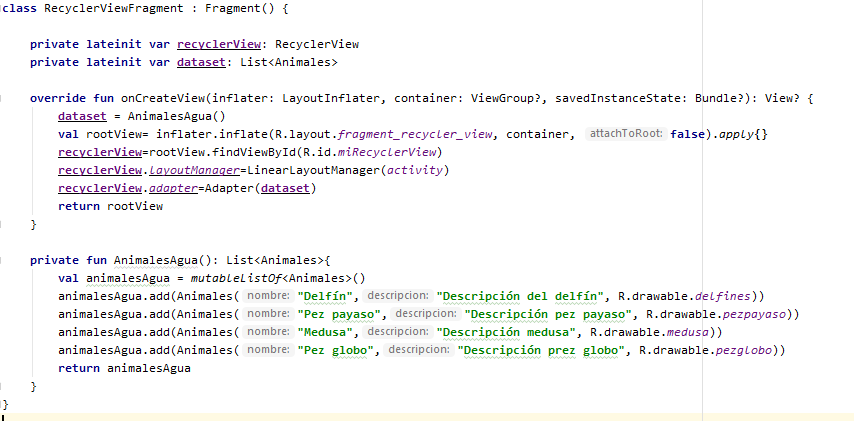


Si por ejemplo quisiéramos que nuestro RecyclerView usara CardView para mostrar la información de diferentes animales acuáticos, tan solo tendríamos que seguir estos pasos.

1. Cambiar el layout creado en el punto 2 por este.
2. Voy a crear una data class Animales y cada animal tendrá un nombre, una descripción y una foto. La foto será un integer, ya que estarán subidas a nuestra aplicación. Para ello creáis una clase de Kotlin. Yo le he llamado Animales.



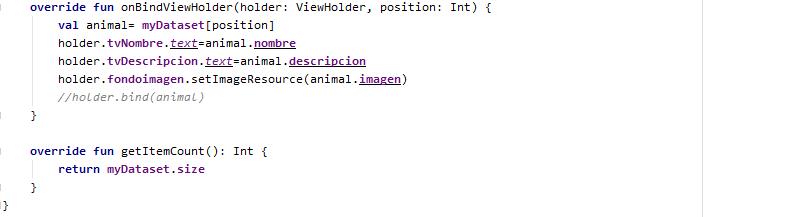
1. Como ahora cada uno de los contenedores de vistas va a tener datos, tendremos que crear los datos en nuestro RecyclerView, creado en el paso 4.



Como veis al inicio del onCreateView he llamado a la función AnimalesAgua que lo que hace es devolver una lista con los animales que voy a usar. Y al crear el adaptador le voy a pasar esta lista. **(En este ejemplo al adaptador le he llamado Adapter en vez de Adaptador, pero por lo demás es igual)**

1. Por último en la clase adaptador que creamos en el punto 3, no le pasábamos ningún dato, ya que usábamos tan solo la posición, en este caso el dato que le pasamos es la lista de animales, la cual tendremos que usar luego en el método onBindViewHolder() para vincular los datos a su contenedor. Además, en este caso en el método getItemCount() en vez de devolver un número de contenedores fijos, devuelve el tamaño de la lista, para que cree tantos contenedores como datos le paso.





El resultado sería el siguiente. **(Además de lo anterior también le he insertado una toolbar)**



En este enlace encontrareis otro ejemplo de RecyclerView. Usan una versión obsoleta de recyclerView, pero tan solo tendríais que cambiar la implementación y el layout del RecyclerView. En este ejemplo no usan un CardView, pero usan la librería de Picasso, de este modo la aplicación no ocupa tanto, ya que las imágenes en vez de estar en la aplicación, están en Picasso, en la nube. Por ello, además de implementar la librería, también tendremos que insertar en el manifest una petición de permiso para usar internet.

Os dejo el enlace.

<https://cursokotlin.com/capitulo-15-recyclerview-kotlin/>

**VIEWPAGER2**

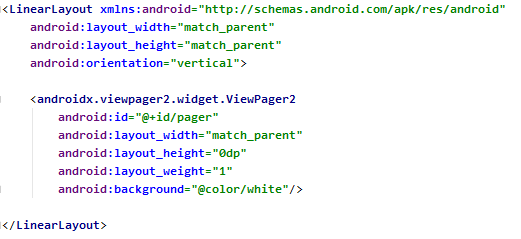
El ViewPager2 sirve para hacer vistas deslizantes. Las vistas deslizantes permiten navegar entre pantallas relacionadas con un gesto horizontal del dedo. Cada vista será un fragment que gestionara el ViewPager. Para ello necesitara un adaptador.

Para poder usar un ViewPager tendremos que crear un fragment donde estará el elemento ViewPager, al que se le asignara un adaptador. En ese adaptador vincularemos cada vista del ViewPager a un fragmento. En nuestro caso usaremos el mismo fragmento para todas las vistas, pero cambiaremos el texto y el color de fondo. Para ello tendremos que dar los siguientes pasos,

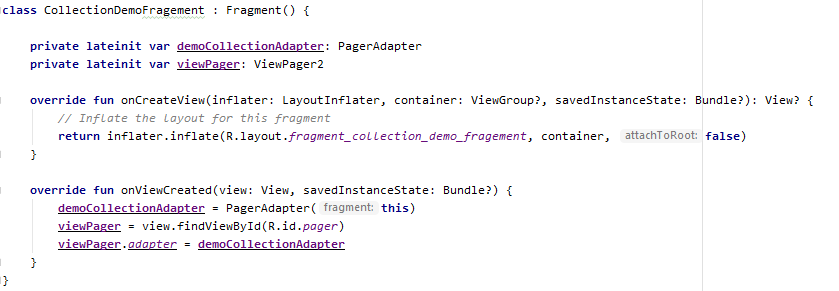
1. Para poder usar un ViewPager2 lo primero que tendréis que hacer será implementar una librería para ello en el gradle del módulo,



1. Además, tendréis que crear un fragment en la actividad principal. En este fragment, en el .xml tendréis que poner el elemento ViewPager2.



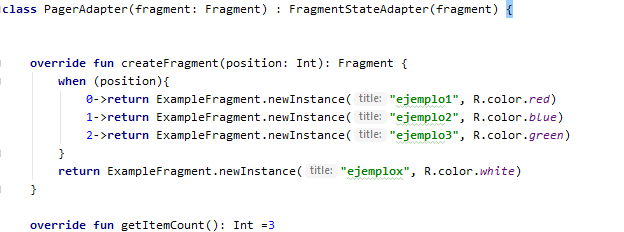
1. En el código del ViewPager2, es decir, en el .kt tendréis que asignarle un adaptador al ViewPager. En nuestro caso el adaptador será una instancia de la clase PageAdapter que crearemos en el siguiente paso. A esta clase siempre debemos pasarle el fragment donde se encuentra el ViewHolder.



1. Ahora crearemos una clase para el adaptador, a la que yo llamara PageAdapter. Esta clase hereda de FragmentStateAdapter. En esta clase tenemos que sobrescribir dos métodos, el createFragment, que lo que hace es crear el fragmento que se va a ver en cada vista, para ello usa la posición.

En este ejemplo para crear los fragmentos hemos creado el método newInstance que está en el fragment que vamos a cargar. Este método tiene como parámetros el titulo y el color de fondo.

También sobrescribiremos el método getItemCount que al igual que en el RecyclerView nos dice cuántas vistas crear.



1. Por ultimo tendremos que crear los fragments que queremos que se carguen en nuestro ViewPager2.

En nuestro ejemplo en todas las opciones crearemos una nueva instancia del fragmento ExampleFragment. Al crear la instancia del fragment, pondremos el título y el color de fondo en un bundle, para que después se puedan usar en nuestra vista.

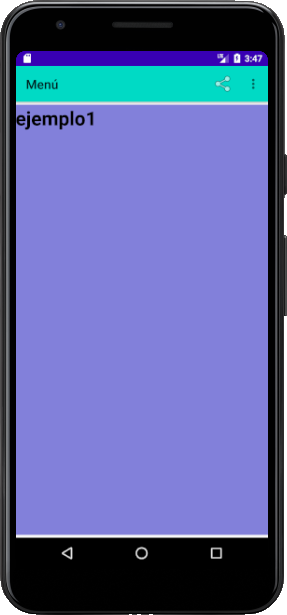
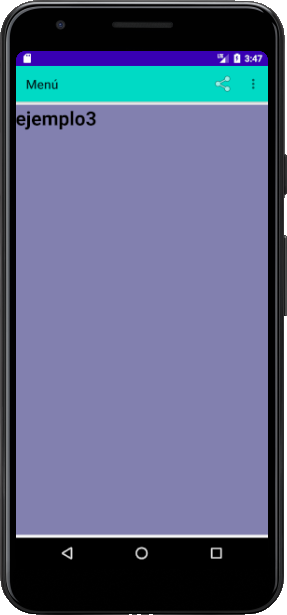


En cuanto al .xml de este fragment tan solo sera un textView que es donde cargaremos el título.



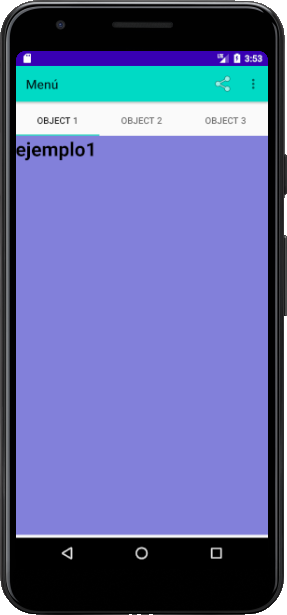
1. El MainActivity lo dejaremos tal cual, y en el .xml tendremos que poner el fragment con el ViewPager, es decir, el primero que creamos.



En este ejemplo aparece un include porque yo también he insertado una toolbar. Este sería el resultado final(de imagen a imagen hemos deslizado la pantalla con el dedo):

**TABLAYOUT**

Un objeto TabLayout va unido a un ViewPager2 y proporciona una forma de mostrar pestañas horizontalmente. Usarlo es muy fácil. Siguiendo con el ejemplo anterior tan solo tendríamos que dar tres sencillos pasos para conseguir esto:



1. Lo primero sería implementar una librería para poder usar el TabLayout, para eso en el gradle del módulo tendréis que implementar lo siguiente,



1. Lo siguiente será incluir un elemento TabLayout encima del ViewPager. Para ello en el .xml del fragment pondremos el elemento <com.google.android.material.tabs.TabLayout…/>



1. Por último, en el código del fragment donde está el ViewPager2 tendremos que crear un TabLayoutMediator para sincronizar la posición del ViewPager con cada una de las pestañas. Además, podremos darle un título a cada pestaña. Para ello usaremos el siguiente código,

