TEMA 3.

Contenido

1 Introduccion

En este tema trataremos **para empezar con RecyclerViews**. Posteriormente seguiremos con intenciones y fotografías.

La RecyclerView es una versión mejorada de la clásica ListView que te permite crear **una lista en la que cada elemento contenga vistas variadas con botones, texto imágenes, enlaces**, etc.

2 RecyclerView

Dentro **del API de Android encontramos las vistas ListView y GridView** nos ofrece una alternativa a **RecyclerView**. Esta última **no ha sido añadida a ningún API** si no que se añade en **una librería de compatibilidad**. A pesar de que **resulta algo más compleja** de manejar, recomendamos el uso de RecyclerView, en lugar de ListView o GridView, al **ser más eficiente y flexible**.

Documentación official de Android:

https://developer.android.com/guide/topics/ui/layout/recyclerview?gclid=CjwKCAiAv_KMBhAzEiwAs-rX1BkAlmsOmVF-LvbBwnF4xSHmWHS4GtexrCPOMWFlYXAYZnPYul3KshoCaqkQAvD_BwE&gclsrc=aw.ds

Aporta las siguientes ventajas:

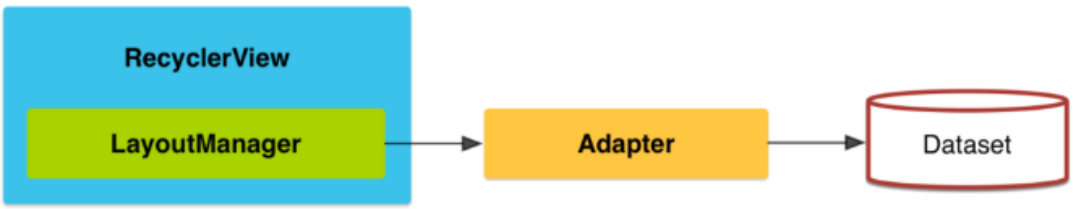
- **Reciclado de vistas** (RecyclerView.ViewHolder)
- **Distribución de vistas configurable** (LayoutManager)

El **RecyclerView es una versión más avanzada del tradicional ListVie**w y lo que hace es mostrar datos cuyos elementos se van reciclando cuando ya no son visibles por el scroll de la lista, lo que **mejora el rendimiento en gran medida**.

¿Como se usa un RecyclerView?

Para usar RecyclerView tienes que definir un adaptador con un LayoutManager.

El adaptador acercará el modelo de datos para ser mostrados y el LayoutManager será el responsable de posicionar cada ítem dentro del RecyclerView y de decidir cuándo reciclar las vistas de items que ya no son visibles.



2.1 Introduciéndola en nuestra aplicación MisLugares

Agregar a vuestro **graddle la siguiente librería de compatibilidad**. Es la versión para AndroidX, mas reciente de Android. Ya no es necesario a partir de version 29

implementation 'androidx.recyclerview:recyclerview:1.0.0'

Creando Adaptador

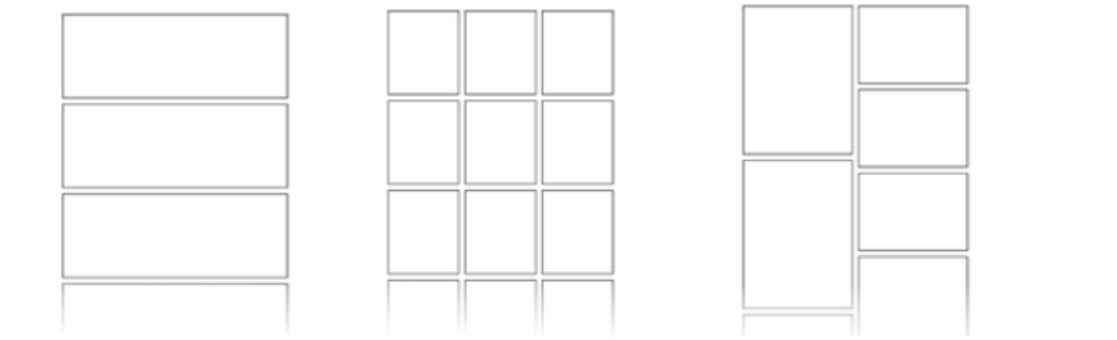
Para personalizar los elementos a **mostrar en un RecyclerView hemos de usar un adaptador**. La creación de **adaptadores puede ser delica**da, en algunos

casos podemos tener problemas de eficiencia. Para evitar estos problemas, Google ha cambiado la forma de trabajar con RecyclerView. Ya no se puede utilizar la interfaz Adapter, como en un ListView, si no que se ha **de utilizar la clase RecyclerView.Adapter.**

Distribuyendo los elementos

A diferencia **ListView o GridView**, que muestran los elementos usando una determinada configuración, **RecyclerView puede configurar esta distribución** por medio de la clase LayoutManager. **El sistema nos proporciona tres descendientes de LayoutManager**, que son mostrados en la siguiente figura. También podemos crear nuestro descendiente de LayoutManager.

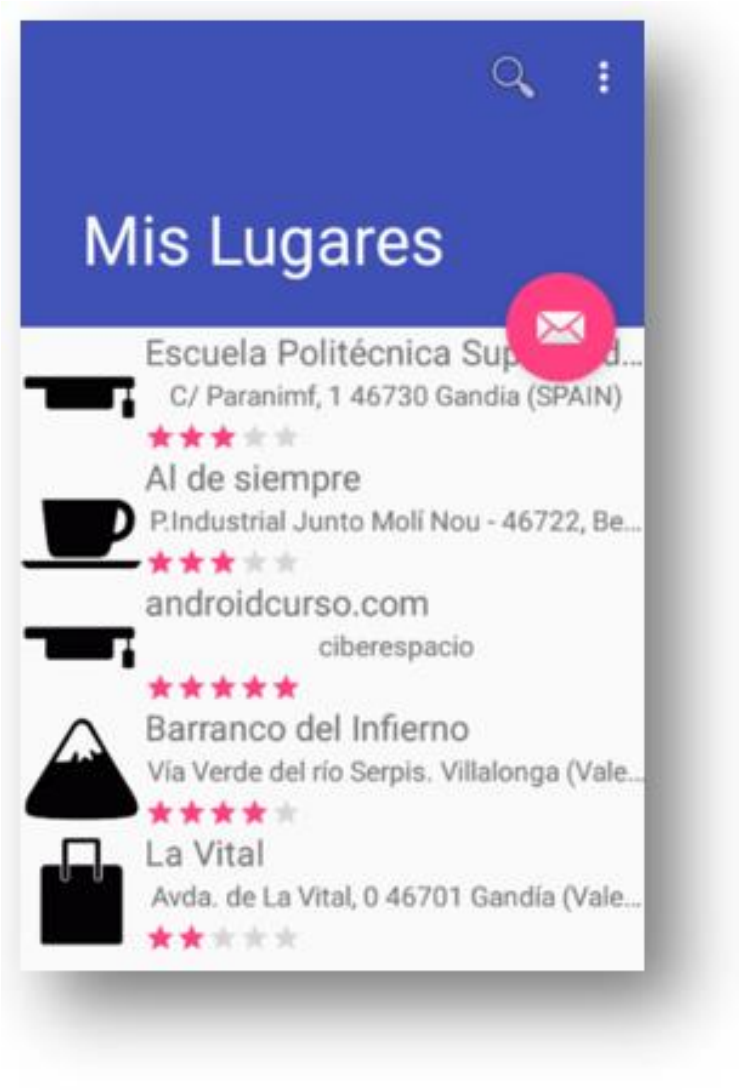
En los siguientes ejercicios usaremos un RecyclerView en Mis Lugares. La **actividad inicial de la aplicación** nos permite **escoger entre cuatro botones**. En una aplicación como la desarrollada, **sería mucho más interesante** que en esta actividad se visualizara directamente una lista con los **lugares almacenados**.



Vais a crear el paquete adaptador AdaptadorLugares a vuestra Aplicación. Y creaís la clase **AdaptadorLugares**.

Resultado

El nuevo aspecto de mainActivity, será el siguiente, mostrando una lista de lugares, al estilo de aplicaciones como whatsup.



2.2 Modificando Layouts código

1. **Reemplaza en el *layout content_main.xml* por el siguiente código:**
   Este Layout se crea para lintroducir la RecyclerView en nuestra aplicación.

<androidx.core.widget.NestedScrollView

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
tools:context="com.example.mislugares2019.presentacion.MainActivity"
    android:theme="@style/Theme.AppCompat.NoActionBar"
    android:padding="24dp"
    android:background="@color/colorPrimary">
    <androidx.recyclerview.widget.RecyclerView
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior" />
</androidx.core.widget.NestedScrollView>
```

2. **Creamos un nuevo layout llamado elementolista.xml**

   Este layout es el aspecto de cada elemento de la lista del RecyclerView, **el adaptador se encargará de rellenarlo**

correctamente. Añadirá uno de estos por cada elemento de la lista que recibe, como veremos más adelante.

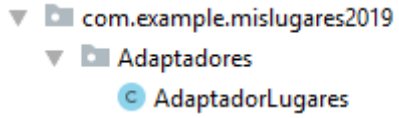
```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="4dp">
    <ImageView android:id="@+id/foto"
        android:layout_width="?android:attr/listPreferredItemHeight"
        android:layout_height="?android:attr/listPreferredItemHeight"
        android:contentDescription="fotografía"
        android:src="@drawable/bar"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"/>
    <TextView android:id="@+id/nombre"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Nombres del lugar"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textStyle="bold"
        android:maxLines="1"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toEndOf="@+id/foto"
        app:layout_constraintEnd_toEndOf="parent"/>
    <TextView android:id="@+id/direccion"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:maxLines="1"
        android:text="dirección del lugar"
        app:layout_constraintTop_toBottomOf="@id/nombre"
        app:layout_constraintStart_toEndOf="@+id/foto"
        app:layout_constraintEnd_toEndOf="parent"/>
    <RatingBar android:id="@+id/valoracion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        style="?android:attr/ratingBarStyleSmall"
        android:isIndicator="true"
        android:rating="3"
        app:layout_constraintTop_toBottomOf="@id/direccion"
        app:layout_constraintLeft_toRightOf="@+id/foto"
        app:layout_constraintBottom_toBottomOf="parent"/>

    <TextView android:id="@+id/distancia"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"

        app:layout_constraintTop_toBottomOf="@id/direccion"
        app:layout_constraintEnd_toEndOf="parent"
        android:gravity="right"
        android:text="... Km" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

3. **En la práctica "Recursos alternativos en Mis Lugares"** se crea un recurso alternativo para este **layout en res/layout-land/content_main.xml. Elimina este recurso alternativo.**

NOTA: Al borrarlo has de desactiva la opción Safe delete.

2.3 Añadiendo el adaptador

Crear **un paquete adaptadores**. Crear una **clase AdaptadorLugares**. Añadir el siguiente código:



Ejercicio. Hemos añadido la nueva clase adapter, debéis modificar el código para usarla al ahora de añadir el drawable y quitar el switch correspondiente.

```
package com.example.mislugares22.adaptadores;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.RatingBar;
import android.widget.TextView;

import androidx.recyclerview.widget.RecyclerView;

import com.example.mislugares22.R;
import com.example.mislugares22.modelo.Lugar;
import com.example.mislugares22.modelo.RepositorioLugares;


public class AdaptadorLugares extends
        RecyclerView.Adapter<AdaptadorLugares.ViewHolder> {
    protected RepositorioLugares lugares; // Lista de lugares a
mostrar
    protected LayoutInflater inflador; // Crea Layouts a partir del
XML
    protected Context contexto; // Lo necesitamos para el inflador
    protected View.OnClickListener onClickListener;


    public AdaptadorLugares(Context contexto, RepositorioLugares
lugares) {
        this.contexto = contexto;
        this.lugares = lugares;
```

```
        inflador = (LayoutInflater) contexto
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }

    public void setOnItemClickListener(View.OnClickListener
onClickListener) {
        this.onClickListener = onClickListener;
    }

    //Creamos nuestro ViewHolder, con los tipos de elementos a
modificar
    public static class ViewHolder extends RecyclerView.ViewHolder {
        public TextView nombre, direccion;
        public ImageView foto;
        public RatingBar valoracion;
        public TextView distancia;
        public ViewHolder(View itemView) {
            super(itemView);
            nombre = (TextView) itemView.findViewById(R.id.nombre);

            direccion = (TextView)
itemView.findViewById(R.id.direccion);
            foto = (ImageView) itemView.findViewById(R.id.foto);
            valoracion=(RatingBar)
itemView.findViewById(R.id.valoracion);
            distancia = (TextView)
itemView.findViewById(R.id.distancia);
        }
    }
    // Creamos el ViewHolder con la vista de un elemento sin
personalizar
    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
// Inflamos la vista desde el xml
        View v = inflador.inflate(R.layout.elemento_lista, parent,
false);
        v.setOnClickListener(onClickListener);
        return new ViewHolder(v);
    }
    // Usando como base el ViewHolder y lo personalizamos
    @Override
    public void onBindViewHolder(ViewHolder holder, int posicion) {
        Lugar lugar = lugares.getElemento(posicion);
        personalizaVista(holder, lugar);
    }
    // Personalizamos un ViewHolder a partir de un lugar
    public void personalizaVista(ViewHolder holder, Lugar lugar) {
        holder.nombre.setText(lugar.getNombre());
        holder.direccion.setText(lugar.getDireccion());
        int id =
AdaptadorTipoLugarDrawable.getRecursoDrawable(lugar.getTipo());

        holder.foto.setImageResource(id);
        holder.foto.setScaleType(ImageView.ScaleType.FIT_END);
        holder.valoracion.setRating(lugar.getValoracion());

        if (RepositorioLugares.posicionActual != null &&
lugar.getPosicion() != null) {
            int d=(int)
RepositorioLugares.posicionActual.distancia(lugar.getPosicion());
```

```
            if (d < 2000) {
                holder.distancia.setText(d + " m");
            } else {
                holder.distancia.setText(d / 1000 + " Km");
            }
        }
    }
    // Indicamos el número de elementos de la lista
    @Override public int getItemCount() {
        return lugares.tamanyo();
    }

}
```

Un adaptador es **un mecanismo estándar en Android que nos permite crear** una serie de **vistas** que han de ser mostradas dentro de un contenedor. Con las **vistas ListView, GridView, Spiner o Gallery has de crear el adaptador utilizando la interfaz Adapter**. Pero con RecyclerView has de utilizar **la clase RecyclerView.Adapter**.

En el **constructor se inicializa el conjunto de datos a mostrar** (en el ejemplo lugares) y otras **variables globales a la clase**. El objeto inflador **nos va a permitir crear una vista a partir de su XML**.

Luego se **crea la clase ViewHolder**, que **contendrá las vistas que queremos modificar de un elemento** (en concreto: dos **TextView** con el nombre y la dirección, un **ImageView** con la imagen del tipo de lugar y un **RatingBar**). Esta clase es utilizada para evitar tener que crear las vistas de cada elemento desde cero. Lo va a hacer **es utilizar un ViewHolder** que contendrá las **cuatro vistas ya creadas**, pero sin personalizar. De forma que, **gastará el mismo ViewHolder para todos los elementos y simplemente** lo personalizaremos **según la posición**. Es decir, **reciclamos el ViewHolder**. Esta forma de proceder mejora el rendimiento del **ReciclerView**, haciendo que funcione más rápido.

El método **onCreateViewHolder()** devuelve una vista de un elemento sin personalizar. Podríamos definir diferentes vistas para **diferentes tipos de elementos** utilizando el parámetro **viewType**. Usamos **el método inflate() para crear una vista a partir del layout XML** definido en **elementolista.xml**. En este método se indica como segundo parámetro el **layout padre que contendrá a la vista que se va a crear**. En este caso, resulta imprescindible indicarlo, ya que queremos que **la vista hijo ha de adaptarse al tamaño del padre** (en **elementolista** se ha **indicado layout_width="match_parent")**. El tercer parámetro del **método permite indicar si queremos que la vista sea insertada en el padre. Indicamos false**, dado que esta operación la va **a hacer el ReciclerView**.

El método **onBindViewHolder() personaliza un elemento de tipo ViewHolder** según **su posicion.** A partir del **ViewHolder que personalizamos ya es el sistema quien se encarga de crear la vista definitiva** que será insertada en el **ReciclerView.** Finalmente, el **método getItemCount() se utiliza para indicar el número de elementos a visualizar.**

2.3.1 Diseccionando el código.

Recogemos la lista de lugares que pasamos como parámetro

```
    protected RepositorioLugares lugares;         // Lista de lugares
a mostrar
    public AdaptadorLugares(RepositorioLugares lugares) {
        this.lugares = lugares;
    }
```

Creamos el ViewHolder. Esta clase le va a dar **el aspecto a nuestra lista.** Internamente **Android crea un ViewHolde**r para cada elemento de la lista es decir recorre la lista que hemos unido al adaptador. Accede a elementolista.xml e **lo infla para mostrar por pantalla cada elemento** de la lista.

```
    public static class ViewHolder extends RecyclerView.ViewHolder {
        public TextView nombre, direccion;
        public ImageView foto;
        public RatingBar valoracion;
        public TextView distancia;
        public ViewHolder(View itemView) {
            super(itemView);
            nombre = itemView.findViewById(R.id.nombre);
            direccion = itemView.findViewById(R.id.direccion);
            foto = itemView.findViewById(R.id.foto);
            valoracion= itemView.findViewById(R.id.valoracion);
            distancia=itemView.findViewById(R.id.distancia);
        }
```

onCreateViewHolder es un método de ViewHolder que tenemos que sobreescribir. **Se ejecuta una vez por cada elemento de la lista** que hemos unido a nuestro adaptador.Podéis ver **que es un ViewHolder por cada elemento de la lista aquí.** El ViewHolder **infla cada fila del RecyclerView** con R.layout.elemento_lista. Falta personalizarlo, ser realizará en el onBind.

Usa la clase **LayoutInflater para inflar con el from R.Layout.elementolista**. Retorna un nuevo **ViewHolder**. Se ejecutará una vez por elemento de la lista. Devuelve un elemento vacío

En este método se indica como segundo parámetro el layout padre que contendrá a la vista que se va a crear. En este caso, resulta imprescindible indicarlo, ya que queremos que la vista hijo ha de adaptarse al tamaño del padre (en elemento_lista se ha indicado layout_width="match_parent"). Ese parent es el LinearLayoutManager, que creamos al inicializar el RecyclerView (ver el código de InicializarRecyclerView de MainActivity). El tercer parámetro del método permite indicar si queremos que la vista sea insertada en el padre. Indicamos false, dado que esta operación la va a hacer el ReciclerView.

```
    // Creamos el ViewHolder con la vista de un elemento sin
personalizar
    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
        // Inflamos la vista desde el xml
        View v = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.elemento_lista, parent, false);
        return new ViewHolder(v);
    }
```

Una vez que los elementos han sido creados, **se realiza el bind con onBindViewHolder**. Este **método de ViewHolder lo sobreescibimos**. Es automático funciona como un evento. **Por cada elemento de la lista se ejecuta una vez. Nos va a permitir personalizar cada elemento de la lista**. Llamaremos **al método personaliza** que hemos creado para personalizarlo.

```
// Usando como base el ViewHolder y lo personalizamos
@Override
public void onBindViewHolder(ViewHolder holder, int posicion) {
    Lugar lugar = lugares.elemento(posicion);
    holder.personaliza(lugar);
}
```

El método personaliza rellena cada elemento de la lista con los valores del lugar para el que se ha creado el ViewHolder. Recordar quitar el switch y usar la clase AdaptadorTipoLugarDrawable

```
public void personalizaVista(ViewHolder holder, Lugar lugar) {
    holder.nombre.setText(lugar.getNombre());
    holder.direccion.setText(lugar.getDireccion());
    int id =
AdaptadorTipoLugarDrawable.getRecursoDrawable(lugar.getTipo());

    holder.foto.setImageResource(id);
    holder.foto.setScaleType(ImageView.ScaleType.FIT_END);
    holder.valoracion.setRating(lugar.getValoracion());

}
```

Cambiamos la foto con setImageResource. Le pasamos como veis un r.drawable, imágenes que hemos introducido en la carpeta drawable anterioremente.

```
foto.setImageResource(id);
```

setScaleType hace que la imagen se ajuste al tamaño que le hemos dado con el parámetro **ImageView.ScaleType.FIT_END**.

```
foto.setScaleType(ImageView.ScaleType.FIT_END);
```

setRating nos permite cambiar la valoración es un float entre 0 y el número máximo de estrella definido.

```
valoracion.setRating(lugar.getValoracion());
```

2.4 Añadiendo el RecyclerView a nuestra Aplicación

Primero en la clase Aplicación vamos a definir nuestro nuevo adaptador.

```
adaptador= new AdaptadorLugares(lugares);
```

Vamos a crear otro método para poder recuperar nuestro adaptador desde cualquier parte de la aplicación.

```
public AdaptadorLugares getAdaptador() {return adaptador;}
```

Vamos a inicializar el RecyclerView en MainActivity. Para eso podies crear un método inicializarReciclerView si queréis que llamaremos desde el onCreate de MainActivity.

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
```

```
lugares = ((Aplicacion) getApplication()).getLugares();
adaptador = ((Aplicacion) getApplication()).getAdaptador();
inicializarReciclerView();


public void inicializarReciclerView() {
    recyclerView = findViewById(R.id.recycler_view);
    recyclerView.setHasFixedSize(true);
    recyclerView.setLayoutManager(new LinearLayoutManager(this));
    recyclerView.setAdapter(adaptador);
```

Ajustamos el tamaño a fijo

recyclerView.setHasFixedSize(true);

Ponemos de LayoutManager un Linear

recyclerView.setLayoutManager(new LinearLayoutManager(this));

Y cargamos el adaptador que vamos a definir.

recyclerView.setAdapter(adaptador);

2.4.1 Añadiendo un OnItemTouchListener al ReciclerView.

El Listener que tenéis en el tutorial no va a funcionar. Vamos a **usar una implementación más compleja basándonos en la pantalla táctil**. Vais a añadir este código a continuación en Inicializar RecyclerView del MainActivity. Añadimmos un OnItemTouchListener que comprueba cuando tocamos un ítem de la lista en la pantalla. **Es un Listener propio de la RecyclerView, pero funciona de manera muy parecida a los listener de pantalla táctil**. Para más información, mirar el tema 5 del tutorial de Android, pantalla táctil.

http://www.androidcurso.com/index.php/recursos/36-unidad-5-entradas-en-android-teclado-pantalla-tactil-y-sensores

Un **OnItemTouchListener permite que la aplicación intercepte eventos táctiles** en progreso en el **nivel de la jerarquía de vistas de RecyclerView** antes de que esos eventos táctiles sean **considerados para el propio comportamiento de desplazamiento de RecyclerView**.

Esto puede **ser útil para aplicaciones que desean implementar diversas formas de manipulación gestual** de vistas de elementos dentro de RecyclerView. OnItemTouchListeners puede **interceptar una interacción táctil que ya está en progreso**, incluso si RecyclerView ya está **manejando ese flujo de gestos para el desplazamiento**. Lo que quiere decir que en el

desplazamiento o deslizamiento podremos capturar ese evento también. No lo haremos en nuestro caso.

http://www.androidcurso.com/index.php/recursos/36-unidad-5-entradas-en-android-teclado-pantalla-tactil-y-sensores/152-la-pantalla-tactil

```
final GestureDetector mGestureDetector =
        new GestureDetector(MainActivity.this, new
GestureDetector.SimpleOnGestureListener() {
      @Override public boolean onSingleTapUp(MotionEvent e) {
          return true;
      }
  });
  recyclerView.addOnItemTouchListener(new
RecyclerView.OnItemTouchListener() {
      @Override
      public void onRequestDisallowInterceptTouchEvent(boolean b) {

      }

      @Override
      public boolean onInterceptTouchEvent(RecyclerView recyclerView,
MotionEvent motionEvent) {
          try {
              View child =
recyclerView.findChildViewUnder(motionEvent.getX(),
motionEvent.getY());

              if (child != null &&
mGestureDetector.onTouchEvent(motionEvent)) {

                  int pos =
recyclerView.getChildAdapterPosition(child);
                  usoLugar.mostrar(id);
                  Toast.makeText(MainActivity.this,"The Item Clicked
is: "+ pos ,Toast.LENGTH_SHORT).show();

                  return true;
              }
          }catch (Exception e){
              e.printStackTrace();
          }

          return false;
      }

      @Override
      public void onTouchEvent(RecyclerView recyclerView, MotionEvent
motionEvent) {

      }
  });
```

2.4.1.1 Diseccionando el Código

```
final GestureDetector mGestureDetector =
        new GestureDetector(MainActivity.this, new
GestureDetector.SimpleOnGestureListener() {
     @Override public boolean onSingleTapUp(MotionEvent e) {
        return true;
     }
 });
```

Detecta varios gestos y eventos utilizando los MotionEvents suministrados. La **devolución de llamada** OnGestureListener notificará a los usuarios cuando se haya producido **un evento de movimiento en particular**. Esta clase solo **se debe usar con MotionEvent**, objeto que proporcional informes de eventos táctiles (no usar para eventos de trackball).

https://developer.android.com/reference/android/view/GestureDetector

Creamos un GestureDetector. Esta clase **nos permite identificar que gesto hemos realizado en la pantalla táctil**. Sobre el vamos a **sobreescribir el método onSingleTapUp** y a devolver **true**. **Lo que conseguimos con esto es que el Listener sólo reaccione, cuando hagamos un tapup. Hacemos click en la pantalla y levantamos el dedo**. Cualquier **otro gesto como deslizar, será ignorado por el Listener**. Pues **el único gesto que hemos hecho que devuelva true es el onSingleTapUp**. Con este listener **podrías detectar también clicks fuera de los ítems por si queréis desplegar un diálogo**.

Con las siguientes lineas añadimos el OnItemTouchListener

```
 recyclerView.addOnItemTouchListener(new
RecyclerView.OnItemTouchListener()                                    {
```

Hay dos métodos del Listener que son obligatorios de sobreeescribir. El primero **onRequestDisallowInterceptTouchEvent lo** dejamos vacío porque no lo usaremos. Porque sirve para **deshabilitar los gestos táctiles** sobre el RecyclerView**. Útil para deshabilitar el control y no permitir eventos en e**l.

```
@Override
    public void onRequestDisallowInterceptTouchEvent(boolean b) {

    }
```

El segundo onInterceptTouchEvent lo usaremos para controlar que hacemos click, **Tapup sobre la lista.** No nos interesa **ni un tapDown ni desplazamientos**, por eso dejamos **el Tapup simulando un click.**

View child = recyclerView.findChildViewUnder(motionEvent.getX(), motionEvent.getY());

Obtenemos en child el item que queremos **capturar a partir de la posición x e y del motionEvent. Clase utilizada para informar eventos de movimiento** (mouse, bolígrafo, dedo, trackball). Los **eventos de movimiento pueden contener movimientos absolutos o re**lativos y **otros datos, según el tipo de dispositivo.**

https://developer.android.com/reference/android/view/MotionEvent

View **findChildViewUnder(float x, float y) Encuentra la primera vista, ítem de la lista justo debajo del punto dado**. Recordar que **cada Item de la lista se muestra como un ViewHolder**, es una vista en sí (elementolista.xml).

Obtenemos **la posición en la lista con**:

```
int pos = recyclerView.getChildAdapterPosition(child);
```

De esta manera **ya sabemos el lugar que hemos seleccionado solo nos falta ir a VistaLugar**. Para eso **llamar a mostrar**, que **nos llevara a VistaLugarActivity** con la posición del lugar que queremos visualizar. El **Toast lo añado para que veaís** que **el ítem recogido es el correcto.**

```
usoLugar.mostrar(id);
Toast.makeText(MainActivity.this,"The Item Clicked is: "+ pos
,Toast.LENGTH_SHORT).show();
```

2.5 Intenciones

Para las intenciones veremos estos enlaces del tutorial, tal cual.

http://www.androidcurso.com/index.php/130

http://www.androidcurso.com/index.php/483

2.6 Fotografías y Permisos

2.6.1 Añadiendo permisos al manifest

Añadiremos los siguientes permisos al comienzo del manifest.xml. Son los que vamos a usar en la **aplicación para añadir fotos de la galería y almacenar fotos** tomadas en **nuestro espacio de memoria externa de la aplicación**. Es **obligatorio si la versión mínima de API** es **anterior a 24**, vamos a tener que pedir permiso para leer ficheros de la memoria externa. En *AndroidManifest.xml* añade dentro de la etiqueta <manifest …> </manifest> el siguiente código. **Hacerlo en cualquier caso**.

Como veis usamos la etiqueta `<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE".` En el name usamos escribimos el tipo de permiso, en este caso READ_EXTERNAL_STORAGE y WRITE_EXTERNAL_STORAGE.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.mislugares2019">
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"
tools:node="replace"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"
tools:node="replace"/>
```

2.6.2 CasoDeUsoPermisos

Este apartado es más complejo porque **debemos asegurarnos de que tenemos permisos** para guardar las fotografías en Android. Para ello creamos un nuevo caso de uso que se va a encargar de los permisos de almacenamiento.

Ver los enlaces:

http://www.androidcurso.com/index.php/282

http://www.androidcurso.com/index.php/626

Crearemos la **clase CasoDeUsoAlmacenamiento** desde el que **controlaremos la petición de permisos** de almacenamiento.

Como veis **solicitamos el permiso de almacenamiento**:

```
solicitarPermiso(Manifest.permission.READ_EXTERNAL_STORAGE,
        "Necesita permisos de almacenamiento para añadir
fotografías",1 );
solicitarPermiso(Manifest.permission.WRITE_EXTERNAL_STORAGE,
        "Necesita permisos de almacenamiento para añadir
fotografías",1 );
```

READ_EXTERNAL_STORAGE: este **permiso lo solicitamos para leer de almacenamiento externo, memoria**, la galería.

WRITE_EXTERNAL_STORAGE: este permiso **lo solicitamos para escribir en almacenamiento**, hacer una foto y almacenarla en un content provider.

Con **el método de activityCompact *requestPermissions*** solicitamos los permisos **que necesitamos**.

```
ActivityCompat.requestPermissions(actividad,
                        new String[]{permiso}, requestCode);
```

Para comprobar si tenemos los permisos de almacenamiento implementamos el método:

```
public static boolean hayPermisoAlmacenamiento(Activity actividad) {
    //La función checkselfpermission pregunta si el permiso se ha
concedido previamente
    return (ActivityCompat.checkSelfPermission(
            actividad, Manifest.permission.READ_EXTERNAL_STORAGE)
            == PackageManager.PERMISSION_GRANTED)

&&(ActivityCompat.checkSelfPermission(actividad,Manifest.permission.WR
ITE_EXTERNAL_STORAGE)
            == PackageManager.PERMISSION_GRANTED);
}
```

Para solicitar permisos de almacenamiento usaremos el siguiente método:

```
public static void solicitarPermisoAlmacenamiento(Activity actividad)
{

    CasoDeUsoPermisos.solicitarPermiso(actividad,new
```

```
String[]{Manifest.permission.READ_EXTERNAL_STORAGE,
                    Manifest.permission.WRITE_EXTERNAL_STORAGE},
            "Necesita permisos de almacenamiento para añadir
fotografías", 1);

}
```

Podemos solicitar el permiso de almacenamiento desde el onCreate de VistaLugarActivity.

VistaLugarActivity en el onCreate.

```
id = extras.getInt("id", -1);
lugar = lugares.elemento(id);
CasoDeUsoPermisos.solicitarPermisoAlmacenamiento(this);
```

Crear la clase CasoUsoPermisos

```
import android.Manifest;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.pm.PackageManager;

import androidx.core.app.ActivityCompat;


public class CasoDeUsoPermisos {

    public static void solicitarPermiso(Activity actividad, final
String permiso[], String justificacion,
                                        final int requestCode) {
        if
(ActivityCompat.shouldShowRequestPermissionRationale(actividad,
permiso[0])||

ActivityCompat.shouldShowRequestPermissionRationale(actividad,
permiso[1])){
            new AlertDialog.Builder(actividad)
                    .setTitle("Solicitud de permiso")
                    .setMessage(justificacion)
                    .setPositiveButton("Ok", new
DialogInterface.OnClickListener() {

                        public void onClick(DialogInterface dialog,
int whichButton) {

ActivityCompat.requestPermissions(actividad,
                                    permiso, requestCode);
```

```
                        }})
                    .show();
        } else {
            ActivityCompat.requestPermissions(actividad,
                    permiso, requestCode);
        }
    }


    public static boolean hayPermisoLocalizacion(Activity actividad) {
        //La función checkselfpermission pregunta si el permiso se ha
concedido previamente
        return (ActivityCompat.checkSelfPermission(
                actividad, Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED);
    }


    public static boolean hayPermisoAlmacenamiento(Activity actividad)
{
        //La función checkselfpermission pregunta si el permiso se ha
concedido previamente
        return (ActivityCompat.checkSelfPermission(
                actividad, Manifest.permission.READ_EXTERNAL_STORAGE)
                == PackageManager.PERMISSION_GRANTED)

&&(ActivityCompat.checkSelfPermission(actividad,Manifest.permission.WR
ITE_EXTERNAL_STORAGE)
                == PackageManager.PERMISSION_GRANTED);
    }

    public static boolean hayPermisoAlmacenamientoEscritura(Activity
actividad) {
        //La función checkselfpermission pregunta si el permiso se ha
concedido previamente
        return (ActivityCompat.checkSelfPermission(
                actividad, Manifest.permission.READ_EXTERNAL_STORAGE)
                == PackageManager.PERMISSION_GRANTED);
    }

    public static void solicitarPermisoAlmacenamiento(Activity
actividad) {

        CasoDeUsoPermisos.solicitarPermiso(actividad,new
String[]{Manifest.permission.READ_EXTERNAL_STORAGE,
                        Manifest.permission.WRITE_EXTERNAL_STORAGE},
                "Necesita permisos de almacenamiento para añadir
fotografías", 1);

    }

    public static void solicitarPermisosLocalizacion(Activity
actividad) {

        CasoDeUsoPermisos.solicitarPermiso(actividad,new
String[]{Manifest.permission.ACCESS_FINE_LOCATION,
                        Manifest.permission.ACCESS_COARSE_LOCATION},
                "Necesita permisos de Localización", 1);
```

```
    }
}
```

En VistaLugar activity debemos introducer este código

```
   @Override public void onRequestPermissionsResult(int requestCode,
                                        String[] permissions, int[] grantResults) {
      if (requestCode == codigoPermiso
            && grantResults.length == 1
            && grantResults[0] == PackageManager.PERMISSION_GRANTED)
         Toast.makeText(this.actividad, "Permiso de almacenamiento
concedido",Toast.LENGTH_LONG).show();
   }
}
```

2.6.2.1 Diseccionando el Código

La parte de **permisos en Android es bastante complicada**. Intentaré ser breve y explicar le funcionamiento en detalle sobre el código. Vamos a empezar con la petición de permisos.

Tenemos tres opciones:

1. El permiso ya se ha concedido, y no se vuelve a preguntar
2. El permiso se ha denegado, con la opción no se vuelve a preguntar. En este caso se debería indicar al usuario que conceda el permiso manualmente.
3. El permiso se ha concedido o denegado una vez, pero se vuelve a preguntar siempre.

Como veis **en casodeusoalmacenamiento** solicitamos los permisos llamando al **método solicitar permisos.** Eso quiere decir que cuando la clase casoUsoAlmacenamiento se instancia lo primero que hará es solicitar los permisos para lectura y escritura en almacenamiento externo. Os muestro el método solicitarPermiso.

```
public static void solicitarPermiso(Activity actividad, final String
permiso[], String justificacion,
                                    final int requestCode) {
    if (ActivityCompat.shouldShowRequestPermissionRationale(actividad,
permiso[0])||

ActivityCompat.shouldShowRequestPermissionRationale(actividad,
permiso[1])){
        new AlertDialog.Builder(actividad)
```

```
                .setTitle("Solicitud de permiso")
                .setMessage(justificacion)
                .setPositiveButton("Ok", new
DialogInterface.OnClickListener() {

                    public void onClick(DialogInterface dialog, int
whichButton) {
                        ActivityCompat.requestPermissions(actividad,
                                permiso, requestCode);
                    }})
                .show();
    } else {
        ActivityCompat.requestPermissions(actividad,
                permiso, requestCode);
    }
}
```

El método **shouldShowRequestPermissionRationale** va a devolver verdadero si al usuario nunca le ha solicitado el permiso la aplicación, o se le ha solicitado pero ha indicado que se pregunte cada vez acerca del permiso. Devolvera falso si ya se ha solicitado y el usuario ha contestado que no se vuelva a preguntar. En este caso, se le debería indicar al usuario que debe dar los permisos manualmente. En nuestro caso y para que no os falle la aplicación volveremos a solicitar el permiso.

En caso de que **shouldShowRequestPermissionRationale**, nos devuelva verdadero creamos un dialogo con la clase AlertDialog y el método builder. En este se indica el nombre del permiso, el título, la justificación de porque debemos pedir el permiso y un botón de ok. A ese botón ok le añadimos un listener con la petición de permisos en caso de

```
 new AlertDialog.Builder(fragment.getActivity())
                    .setTitle("Solicitud de permiso")
                    .setMessage(justificacion)
                    .setPositiveButton("Ok", new
DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {
       fragment.requestPermissions(
      new String[]{permiso}, requestCode);
                  }})
             .show();
      } else {
         fragment.requestPermissions(new String[]{permiso},
requestCode);
      }
```

El método hayPermisoAlmacenamiento(), nos va a comprobar si el permiso ha sido concedido o no. Lo vamos a usar en las actividades cuando vayamos a meter fotografías. Si no lo hacemos e intentamos almacenar fotografías en nuestra aplicación, el programa producirá una excepción y fallará.

```
public static boolean hayPermisoAlmacenamiento(Activity actividad) {
    //La función checkselfpermission pregunta si el permiso se ha
concedido previamente
    return (ActivityCompat.checkSelfPermission(
            actividad, Manifest.permission.READ_EXTERNAL_STORAGE)
            == PackageManager.PERMISSION_GRANTED)

&&(ActivityCompat.checkSelfPermission(actividad,Manifest.permission.WR
ITE_EXTERNAL_STORAGE)
            == PackageManager.PERMISSION_GRANTED);
}
```

Implementamos el interfaz **OnRequestPermissionsResultCallback**, para poder entre otras cosas recibir el resultado de la petición de permisos lanzada. Funciona muy parecido a **startActivityForResult** en el sentido de que mandamos un **requestcode**, y en la respuesta podemos comprobar si los permisos han sido efectivamente concedidos. Este método se debe añadir en VistaLugarActivity

```
//devuelve el resultado de la edición
@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent
        data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == CasosUsoLugar.RESULTADO_EDITAR) {
        actualizarVistas();
        findViewById(R.id.scrollViewVista).invalidate();
    } else if (requestCode == CasosUsoLugar.RESULTADO_GALERIA) {
        if (resultCode == Activity.RESULT_OK) {
            lugar.setFoto(data.getDataString());
            CasosUsoLugar.ponerFoto(imageView, lugar.getFoto());
            CasosUsoLugar.guardar(this, lugares, lugar.getId(),
lugar);
        } else {
            Toast.makeText(this, "Foto no cargada",
Toast.LENGTH_LONG).show();
        }
    } else if (requestCode == CasosUsoLugar.RESULTADO_FOTO) {
        if (resultCode == Activity.RESULT_OK && uriUltimaFoto != null)
{
            uriUltimaFoto= CasosUsoLugar.getUltimaFoto();
            lugar.setFoto(uriUltimaFoto.toString());
            CasosUsoLugar.ponerFoto(imageView,
uriUltimaFoto.toString());
            CasosUsoLugar.guardar(this, lugares, lugar.getId(),
lugar);
            findViewById(R.id.scrollViewVista).invalidate();
```

```
        } else {
            Toast.makeText(this, "Error en captura",
Toast.LENGTH_LONG).show();
        }

    }

}
```

2.6.3 Permisos en VistaLugarActivity

Tanto al tomar la fotografía como al poner una fotografía de la galería, necesitaremos comprobar si tenemos permisos de almacenamiento externo, para poder guardar la fotografía en la carpeta reservada en nuestra aplicación por el contentprovider que veremos más adelante. Esto lo realizaremos en el listener asociado a los dos iconos de fotografías y galería, que veremos en el siguiente punto. Pero no olvidéis en esos listener preguntar si tenemos permisos de almacenamiento externo.

```
galeria.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        if (usoAlmacenamiento.hayPermisoAlmacenamiento()) {
            usoLugar.galeria(v, RESULTADO_GALERIA);
        } else {
            Toast.makeText(getBaseContext(), "No hay permisos de
almacenamiento, no se puede tomar la foto", Toast.LENGTH_LONG).show();
        }
    }
});

camara.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {


       if (usoAlmacenamiento.hayPermisoAlmacenamiento()) {

           uriUltimaFoto=usoLugar.tomarFoto(RESULTADO_FOTO);

        }else {

           Toast.makeText(getBaseContext(), "No hay permisos de
almacenamiento, no se puede tomar la foto", Toast.LENGTH_LONG).show();


        }

    }
});
```

`usoAlmacenamiento.hayPermisoAlmacenamiento()` nos devuelve si tenemos permisos para almacenar en la memoria externa.

2.6.4 Fotografías

Para las fotografías seguir el libro tal cual

http://www.androidcurso.com/index.php/484