SESIÓN 6

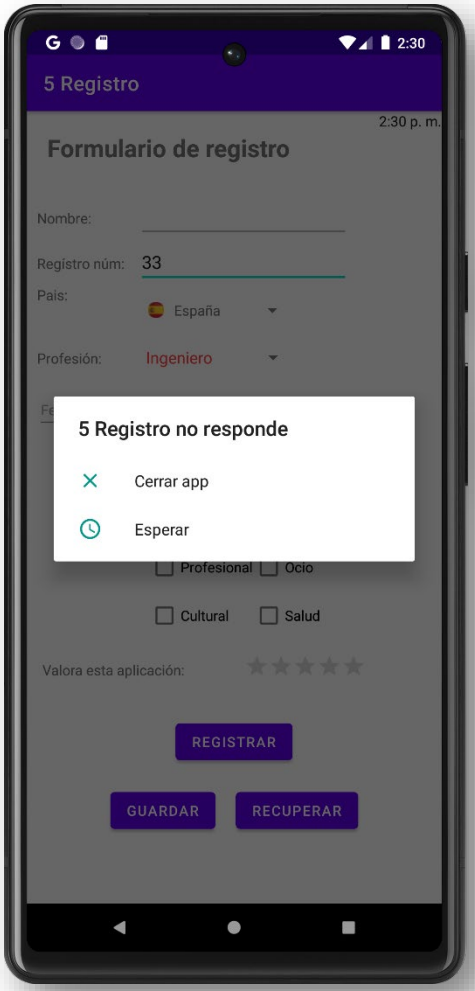
Tareas asíncronas

Conexiones HTTP

Clase ViewModel

ANR: Application Not Responding

- Al inicio de una APP solamente hay un hilo en ejecución, el *hilo de la GUI*.
- Si el *hilo de la GUI* es bloqueado durante un periodo de tiempo prolongado (5 segundos) el sistema muestra el cuadro de dialogo ANR: *Application Not Responding*.
- Como evitarlo:
  - Los manejadores de eventos de la GUI deben ser ligeros y rápidos, retornando lo antes posible.
  - Para operaciones con alta carga computacional o potencialmente lentas, como una operación de descarga de un recurso de internet, debe lanzarse un *hilo de trabajo* que se encargue de la operación.
- El objetivo es que la interfaz de usuario no se quede nunca bloqueada.
- El acceso a los elementos de la interfaz, por ejemplo modificar un `TextView`, debe hacerlo e*l hilo de la GUI*, no el *hilo de trabajo*.



Tareas asíncronas (1)

- Para realizar tareas que lleven tiempo se deben usar hilos diferentes al de la actividad de la GUI.
- La clase Activity dispone del método runOnUiThread() que permite ejecutar código en el hilo principal (UI Thread)
- Sintaxis:

```
runOnUiThread(new Runnable() {
        @Override
        public void run() {
                // Código que se ejecutará en la GUI
        }
    });
```

Tareas asíncronas (2)

- Android dispone de la clase AsyncTask (*deprecated* desde la versión 11) que facilita el trabajo de ejecutar operaciones en un *hilo de trabajo* distinto al de la GUI, y realizar las modificaciones de la interfaz en el *hilo de la GUI*.
- La clase AsyncTask tiene el método doInBackground() que se ejecuta en un *hilo de trabajo* y otros métodos que se ejecutan en el *hilo de la GUI*.

Métodos principales	Métodos auxiliares
onPreExecute() **doInBackground()** **onProgressUpdate()** **onPostExecute()**	**publishProgress()**

- La clase AsyncTask ha sido desaconsejada por Google por ser menos eficiente que alternativas como Thread o java.util.concurrent, tener fugas de memoria si no se usa adecuadamente, incapacidad de realizar múltiples tareas en segundo plano, …

https://developer.android.com/reference/android/os/AsyncTask.html

Tareas asíncronas (3)

```
// Ejemplo de una clase interna, de la clase de la actividad, para ejecutar
código pesado usando un esquema similar a AsyncTask.

private class TareaAsíncrona implements Runnable {
   private double parámetros[];

   public TareaAsíncrona(double parámetros[]) {
      this.parámetros=parámetros;
   }

   @Override
   public void run() {

            onPreExecute(); // Ejecución en la GUI

            int resultado = doInBackground(parámetros); // Código de pesado

            onPostExecute(resultado); // Ejecución en la GUI

}
```

Tareas asíncronas (4)

```
private void onPreExecute() {
  // Método de procesamiento inicial.
  runOnUiThread(new Runnable() {
     @Override
      public void run() {
            // Acceso a la GUI
      }
  });
}

// Se ejecuta en el hilo de trabajo, por lo que
// NO debe modificar elementos de la GUI
private int doInBackground(double parámetros[]) {
    // Trabajo del hilo

    publishProgress(progreso);

    // Trabajo del hilo

    publishProgress(progreso);

    return resultado;  // Valor devuelto a onPostExecute()
}
```

```
private void publishProgress(double progreso) {
   runOnUiThread(new Runnable() {
       @Override
       public void run() {
          onProgressUpdate(progreso);
       }
   });
}

private void onProgressUpdate(double progreso) {
    // Acceso a la GUI
}

private void onPostExecute(final int resultado) {
    // Método de procesamiento final.
    runOnUiThread(new Runnable() {
       @Override
        public void run() {
            // Método de procesamiento inicial.
      }
   });
} // Fin de la clase interna
```

Tareas asíncronas (5)

La tarea asíncrona puede lanzarse:

- Mediante *Thread*:

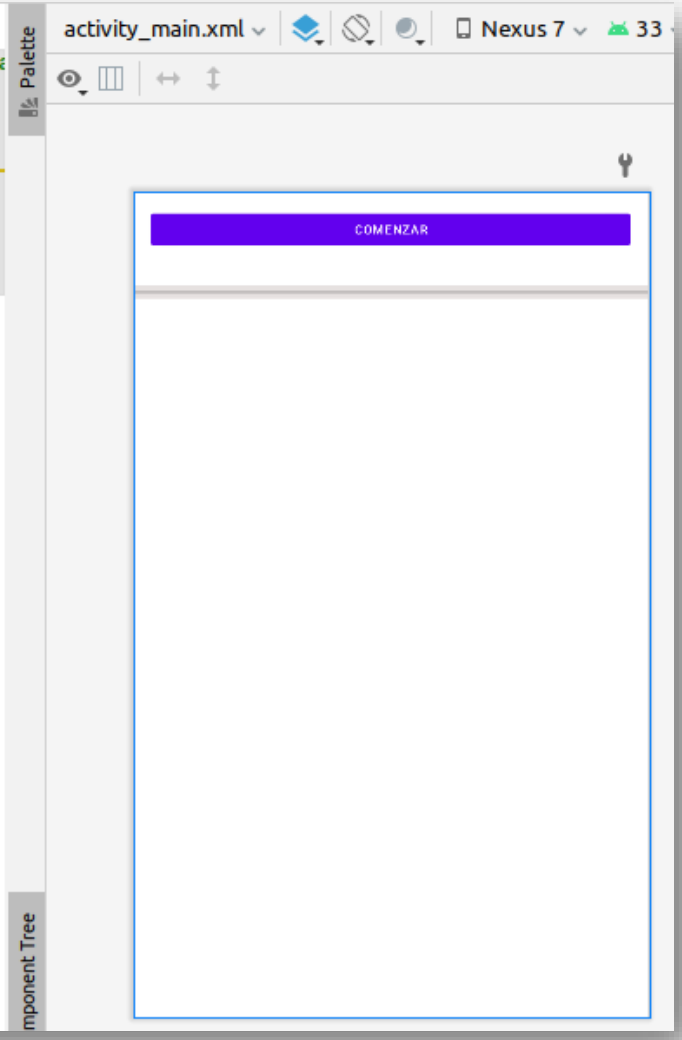
```
Thread tarea = new Thread (new TareaLenta(parámetros));
tarea.start();
```

- Mediante *ExecutorService*

```
ExecutorService executor = Executors.newSingleThreadExecutor();
executor.execute (new TareaLenta(parámetros));
```

Ejemplo de tarea asíncrona. Diseño actividad

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/a
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btComienzo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20dp"
        android:onClick="btComenzarOnClick"
        android:text="Comenzar" />

    <ProgressBar
        android:id="@+id/barraProgreso"
        style="@style/Widget.AppCompat.ProgressBar.Horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:background="#E6E1E0"
        android:max="100" />

</LinearLayout>
```



Ejemplo de tarea asíncrona interna. Código (1)


```
void btComenzarOnClick (View view ) {
   // Valor máximo e incremento
   double parámetros[] = {100.0, 10.0};
   (new Thread (new TareaProgreso (parámetros))).start();

}


// Declaración de la clase como interna a la de la actividad principal

private class TareaProgreso implements Runnable {
   private double parámetros[];
   public TareaProgreso (double parámetros[]) {
       this.parámetros=parámetros;
   }

   @Override
   public void run() {
           onPreExecute();
           doInBackground(parámetros);
           onPostExecute();
   }
```

Ejemplo de tarea asíncrona interna. Código (2)

```
protected  void onPreExecute() {

        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // Código a ejecutar en el hilo de la GUI
                barraProgreso.setProgress(0); // Inicializar la barra de progreso
            }
        });

 }

 protected void doInBackground(double parámetros) {  // Se ejecuta en el hilo de trabajo

     // parámetros[0] es el valor máximo (100)
     // parámetros[1] es el incremento (10)
     for ( int i=0; i<=parámetros[0]; i+=parámetros[1] ) {
            publishProgress(i);              // Invoca a onProgressUpdate()
            SystemClock.sleep(1000);     // Esperar 1 segundo
     }
}

private void  publishProgress (int progreso) {
   runOnUiThread(new Runnable() {
       @Override
       public void run() {
                onProgressUpdate(progreso);
        }
    });
}
```

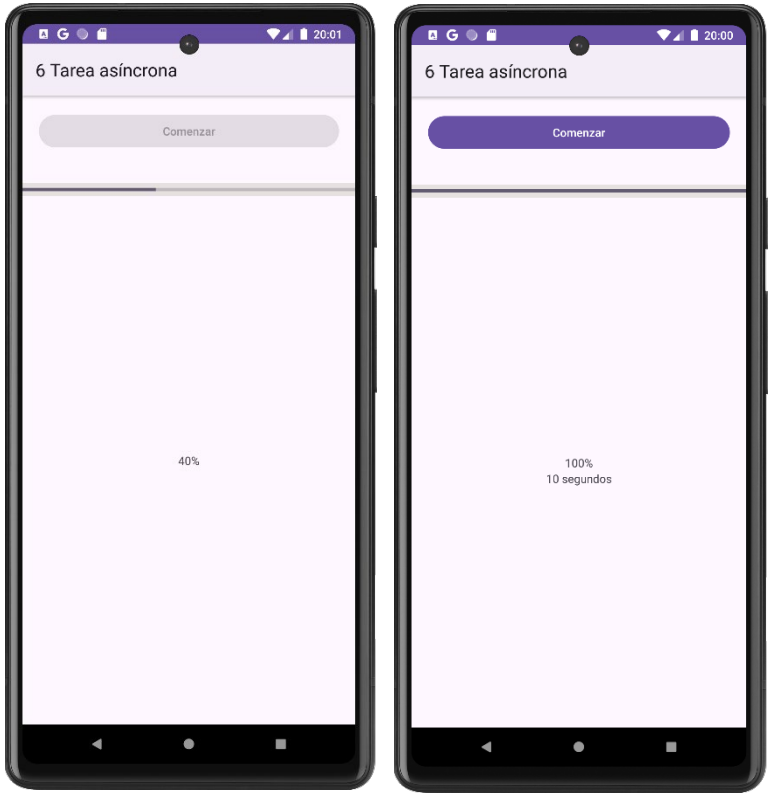
```
// Se ejecuta en el hilo de la GUI
protected void onProgressUpdate(int progreso) {

     barraProgreso.setProgress( progreso );

}
```

```
protected void onPostExecute() {

        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // Se ejecuta en el hilo de la GUI
            }
        });


  }
} // Fin de la clase TareaProgreso
```

Ejercicio 1

- Crea un proyecto nuevo, con las siguientes características:
  - Nombre: “6 Tarea asíncrona”
  - Paquete: dam. tareaasincrona
  - Directorio: 6- TareaAsincrona
- Implementa el ejemplo anterior añadiendo:
  - El % de progreso
  - Al finalizar que muestre el tiempo total que ha pasado
- Prueba a pulsar el botón otra vez cuando ya ha comenzado a ejecutarse la tarea. Para evitar el problema deshabilita el botón en `onPreExecute()` y vuelve a habilitarlo en `onPostExecute()`

```
btComienzo.setEnabled(false);// Deshabilita
btComienzo.setEnabled(true); // Habilita
```



https://developer.android.com/reference/android/widget/ProgressBar

Clase asíncrona externa (1)

- Si la clase asíncrona es externa, podría necesitar acceder a atributos y métodos de la clase de la actividad que la pone en marcha
- Puede solucionarse si la clase asíncrona dispone de un constructor con un parámetro en el que recibe una referencia a la clase que le invoca. En este caso los métodos o atributos de la clase principal a los que se accede desde la clase externa asíncrona tienen que estar declarados con el modificador de acceso **`protected`** (si está en el mismo paquete) o bien usar métodos públicos *setter* y *getter* para su acceso
- Un diseño mejor sería usando una interfaz

Clase asíncrona externa (2)

Definición de la interfaz

```
public  interfaz Comunicación {
    void inicializarDiseño ();
    void mostrarProgreso (final int progreso);
    void actualizarDiseño (final int resultado);
}
```

Clase asíncrona externa (3)

```
public  class TareaAsíncrona implemets Runnable {
   private Comunicación comunicación;
   private double parámetros[];

   // El constructor recibe como parámetro la referencia de la clase que implemente la interfaz
   // para usar los métodos públicos definidos en la interfaz
   public TareaAsíncrona (Comunicación comunicación, double parámetros[]) {
      this. comunicación = comunicación;
      this.parámetros=parámetros;
   }

   @Override
   public void run() {
         onPreExecute();    // Preparar entorno
         int resultado = doInBackground(parámetros);  // Código de pesado
         onPostExecute(resultado);  // Actualizar entorno al finalizar ejecución
   }
}
```

Clase asíncrona externa (4)

```
private void onPreExecute() {
    // Método de procesamiento inicial.
    comunicación.inicializarDiseño();
}

private int doInBackground(double parámetros[]) {
    // Trabajo del hilo
    publishProgress(progreso);
    // Trabajo del hilo
    publishProgress(progreso);
    return resultado;  // Valor devuelto a onPostExecute()
}
```

```
private void publishProgress(double progreso) {
    // Indicación de progreso
    comunicación.mostrarProgreso();
}

private void onPostExecute(final int resultado) {
    // Método de procesamiento final
    comunicación.actualizarDiseño();
}
```

Clase asíncrona externa (5)

```
// La actividad debe implementar la interfaz
public class MainActivity extends AppCompatActivity implements Comunicación {
    :::::::


    private void inicializarDiseño() {
        runOnUiThread(new Runnable() {
        @Override
                public void run() {
                    // Acceso a los de la GUI
                }
        });
    }

    private void mostrarProgreso(final int progreso) {
        runOnUiThread(new Runnable() {
        @Override
                public void run() {
                    // Acceso a los de la GUI
                }
        });
    }

    private void actualizarDiseño (final int resultado) {
        runOnUiThread(new Runnable() {
        @Override
                public void run() {
                    // Acceso a los de la GUI
                }
        });
    }

}
```

Clase asíncrona externa (6)

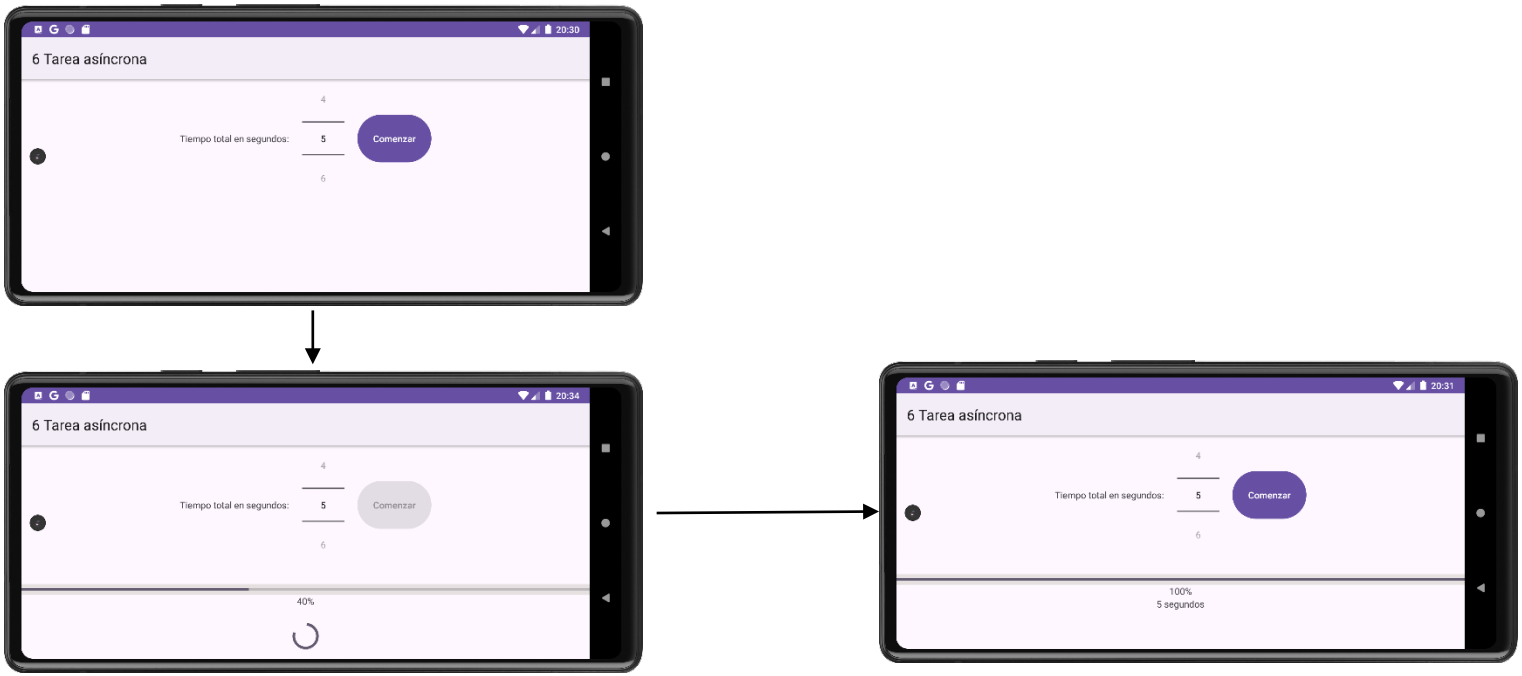
- Si la actividad principal se destruye, por ejemplo por un cambio de orientación, la tarea asíncrona, al ser una clase externa, sigue ejecutándose, y al tener una referencia a la actividad, el objeto no es eliminado por el recolector de basura y puede haber pérdida de memoria
- Solución: usar **WeakReference** para obtener la instancia de la actividad.
  - Crear atributo:
    ```
    final private WeakReference<Comunicación> comunicaciónRef;
    ```
  - Si comunicación es un parámetro recibido en el constructor de la clase:
    ```
    this.comunicaciónRef = new WeakReference<>(comunicación);
    ```
  - En todos los métodos de la clase asíncrona verificar que la instancia de la actividad existe:
    ```
    Comunicación comunicación = comunicaciónRef.get();
    if (comunicación == null)
        return;
    else
        comunicación. mostrarProgreso(); // Invocar a uno de los métodos de la interfaz
    ```

https://developer.android.com/reference/java/lang/ref/WeakReference

Ejercicio 2 (1)

Modificar el ejercicio 1 con un nuevo diseño para la orientación horizontal que recoja el tiempo desde un widget `NumberPicker`. Restricciones:

- Crear una nueva clase *asíncrona* externa para gestionar el progreso
- Antes de pulsar el botón *Comenzar* no debe mostrarse ninguna barra de progreso
- Tras pulsar el botón *Comenzar* debe recoger el tiempo en el que realizará la acción desde el `NumberPicker`. A partir de este tiempo calculará el incremento de la barra de progreso
- Mientras se ejecuta la tarea *asíncrona*, mostrará una barra de progreso horizontal, una circular infinita y el %
- Al finalizar la tarea *asíncrona* debe mostrar el tiempo realmente transcurrido y ocultar la barra de progreso circular



Ejercicio 2 (2)

Pistas:

- El widget NumberPicker no está en la *Palette* del editor de diseño de Android Studio, debes empezar a escribirlo en la vista XML y saldrá un asistente para completarlo
- Los métodos a usar de NumberPicker son:

```
NumberPicker numberPicker = findViewById(R.id.numpicker);
numberPicker.setMaxValue(99); // Valor máximo
numberPicker.setMinValue(1);  // Valor mínimo
numberPicker.setValue(5);     // Valor inicial
int valor=numberPicker.getValue();
```

Ejercicio 2 (3)

Pistas:

- La barra de progreso circular usa el elemento ProgressBar pero con otro valor en el atributo style

```
<ProgressBar
   android:id="@+id/barraProgresoHorizontal"
   style="?android:attr/progressBarStyleHorizontal"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"/>
```

```
<ProgressBar
   android:id="@+id/barraProgresoCircular"
   style="?android:attr/progressBarStyle"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"/>
```

Conexión HTTP

- Es preciso solicitar permiso al usuario para poder acceder a red. Esto se realiza en el fichero AndroidManifest.xml bajo el elemento raíz `<manifest>`.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

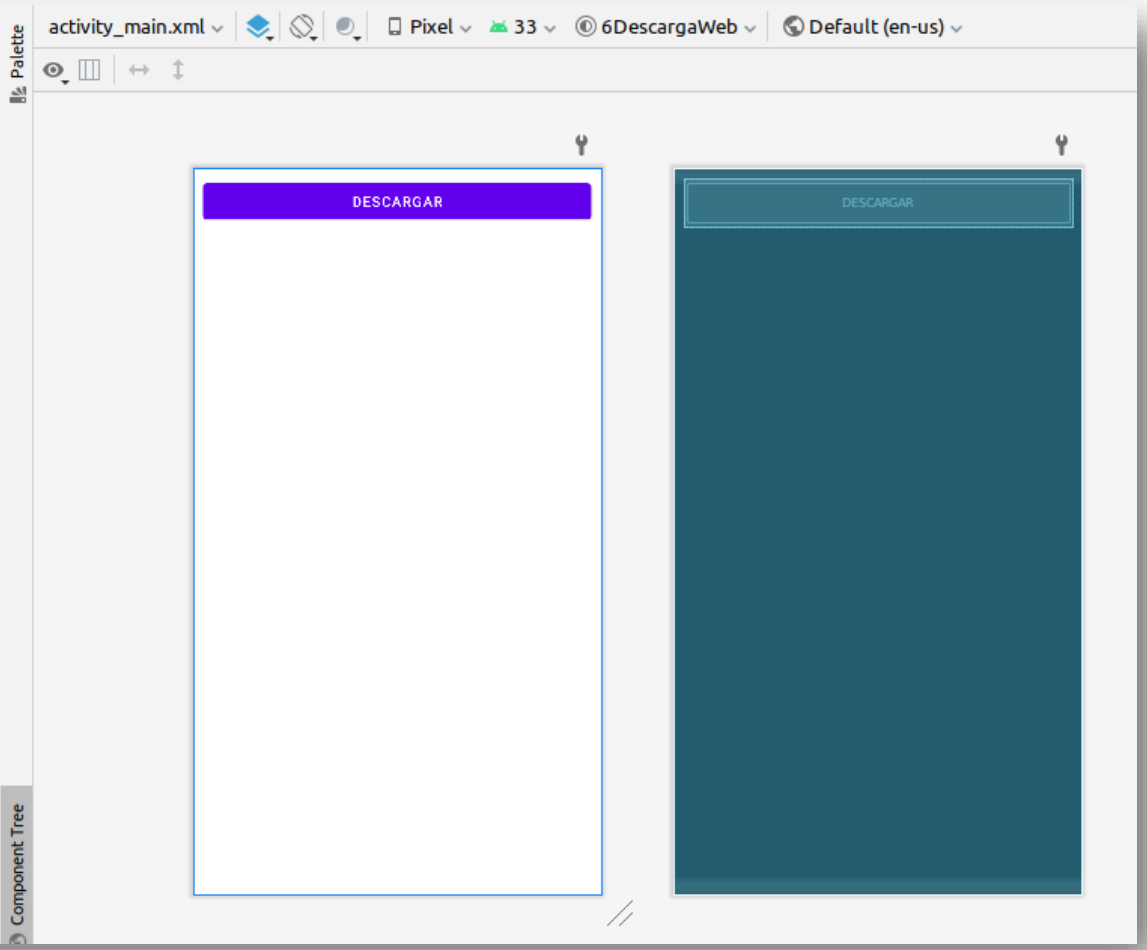
- Petición HTTP básica

```
try {
    URL url = new URL("https://www.upm.es/");
    HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
    if (urlConnection.getResponseCode() == HttpURLConnection.HTTP_OK) {
        String contentType = urlConnection.getContentType();
        InputStream is = urlConnection.getInputStream();
        // Leer los datos del flujo InputStream
    }
} catch (Exception e) { respuesta = e.toString(); }
```

- Más información
  - https://developer.android.com/reference/java/net/HttpURLConnection
  - Tipos MIME: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

Ejemplo DescargaWeb: Cámaras tráfico Madrid (1)

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.andr
    xmlns:tools="http://schemas.android.com/tools
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/botonDescargar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:onClick="btComenzarOnClick"
        android:text="Descargar" />

    <ScrollView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/datosLeidos"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"

    </ScrollView>

</LinearLayout>
```



Ejemplo DescargaWeb: Cámaras tráfico Madrid (2)

```
private String doInBackground(String url) {
 String respuesta;

 try {
   URL url = new URL(url);
   HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
   if (urlConnection.getResponseCode() == HttpURLConnection.HTTP_OK) {
     String contentType = urlConnection.getContentType();

     // EL tipo MIME debe ser "application/vnd.google-earth.kml+xml"
     if (contentType.equals("application/vnd.google-earth.kml+xml")) {
        InputStream is = urlConnection.getInputStream();
        InputStreamReader reader = new InputStreamReader(is);
        BufferedReader bufferedReader = new BufferedReader(reader);

        respuesta = "";
        final StringBuilder contenido=new StringBuilder();
        final String línea;
        línea= bufferedReader.readLine();
        while (linea != null) {
           contenido.append(linea).append("\n");
           linea = bufferedReader.readLine();
        }
        respuesta=contenido.toString();
      } else {
        respuesta = "La respuesta es de tipo "+contentType+" y no será procesada";
     }
     urlConnection.disconnect();
   } else {
     respuesta = "Ha fallado la conexión a "+ url;
   }
 } catch (Exception e) {
     respuesta = "Se ha producido esta excepción: "+ e.toString();
 }
 return respuesta;
}
```

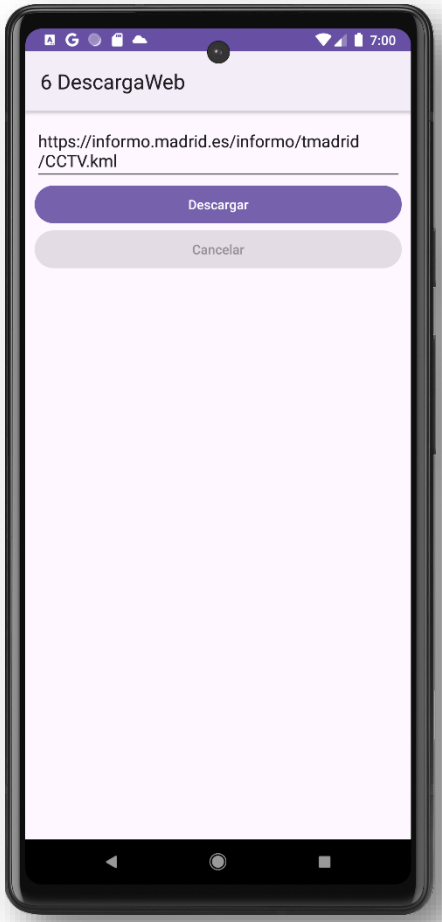
https://informo.madrid.es/informo/tmadrid/CCTV.kml

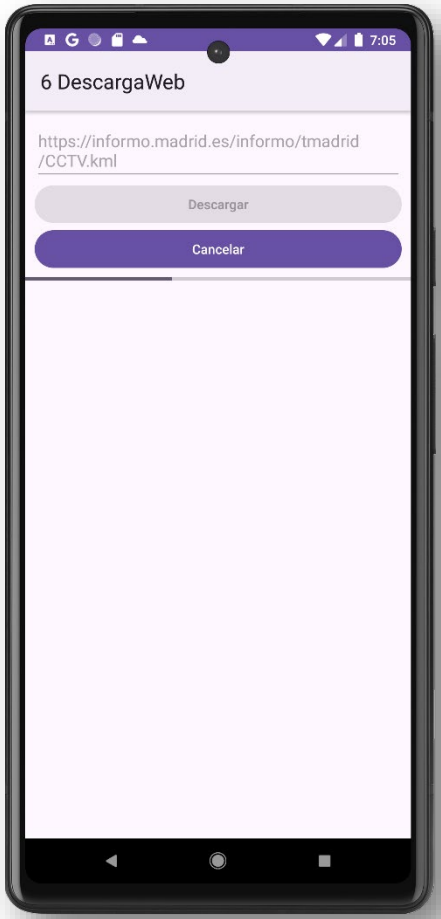
Ejercicio 3 (1)

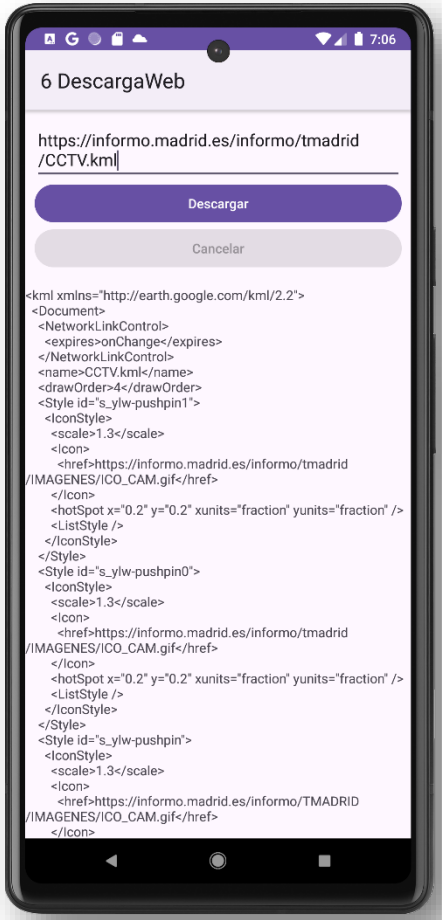
- Crea un proyecto nuevo, con las siguientes características:
  - Nombre: “6 DescargaWeb”
  - Paquete: dam.descargaweb
  - Directorio: 6-DescargaWeb
- Implementa el ejemplo anterior con las siguientes modificaciones:
  - Añade un botón para cancelar y un `EditText` para que el usuario pueda especificar una URL
  - Si la URL es un recurso html o klm:
    - Muestra una barra de progreso con el estilo:
      - Horizontal, si ha podido obtener el tamaño del recurso
      - Circular, si no lo ha podido obtener
    - Si el recurso es html, muestra el contenido en un `WebView`
    - Verifica el funcionamiento con los siguientes urls:
      - https://www.upm.es
      - https://informo.madrid.es/informo/tmadrid/CCTV.kml
  - Si la URL es un recurso imagen:
    - Muestra una barra de progreso circular.
    - Verifica el funcionamiento con el siguiente url: https://www.etsist.upm.es/uploaded/80/biblioteca.jpg
- La clase que descarga la imagen debe ser interna y la que descarga textos externa
- Las barras de progreso y el botón cancelar solo deben estar visibles durante la descarga.
- El botón Descargar y el *EditText* del url deben estar inhabilitados durante la descarga
- Puesto que la descarga puede ser muy rápida, introduce retardos.
- Si hay un error en la descarga, muéstralo en un `TextView` o con un `toast`
- Pon un icono a la aplicación

Ejercicio 3 (2)

Descarga de klm

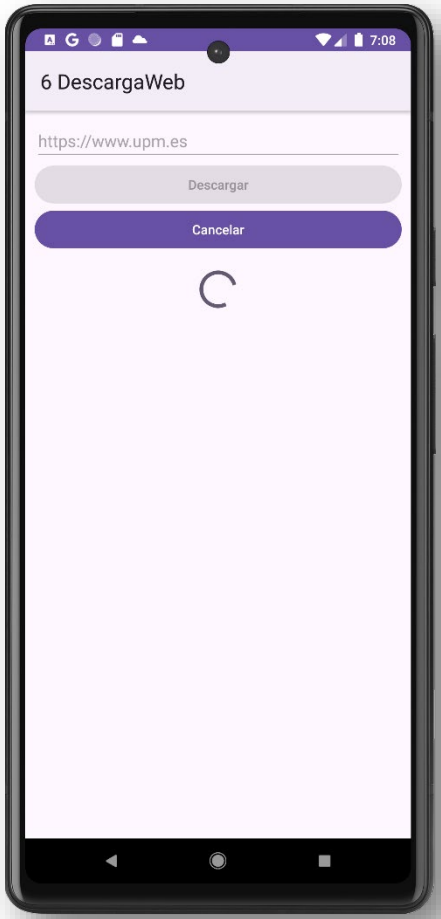






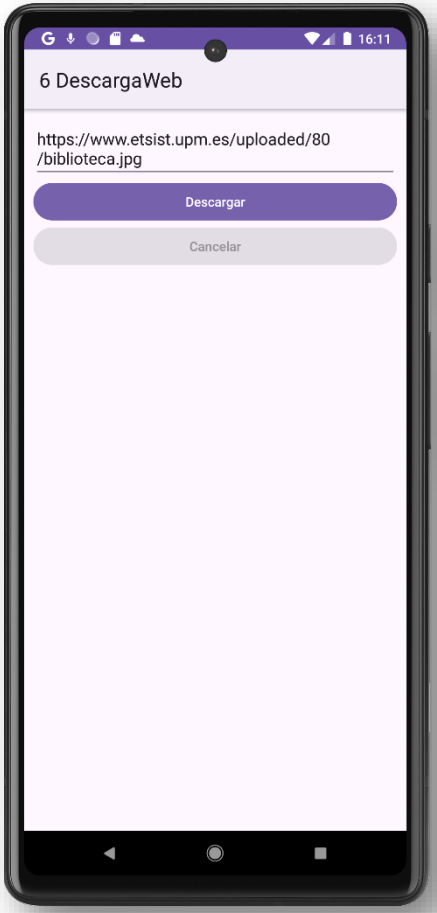
Ejercicio 3 (3)

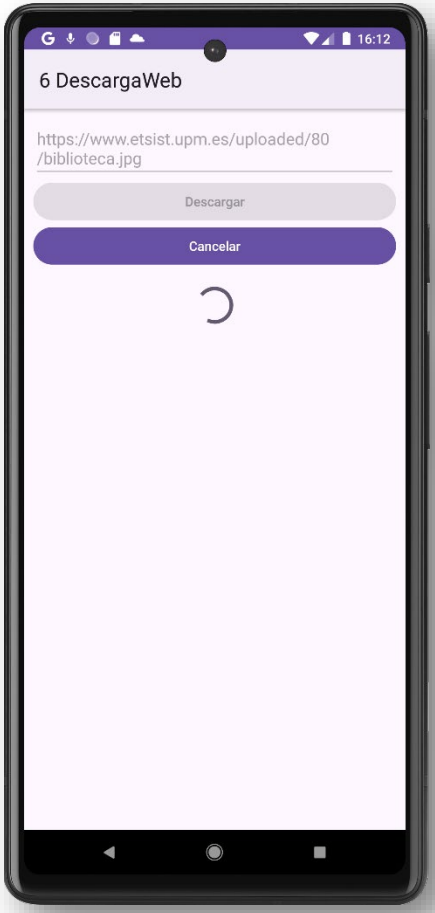
Descarga de html

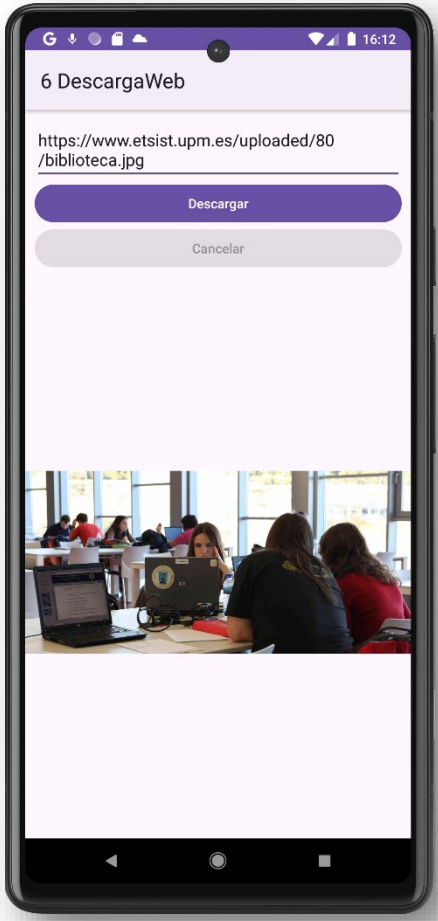


Ejercicio 3 (4)

Descarga de una imagen







Ejercicio 3 (5)

Pistas:

- En un recurso de texto se puede obtener el nº de líneas de antemano, para implementar una barra de progreso horizontal, mediante el método `getContentLength()` de `HttpURLConnection`. Si devuelve -1 es que esa información no está en la cabecera http. Antes hay que configurar la conexión para que no comprima el recurso con `setRequestProperty()`. Ejemplo:

```
HttpURLConnection urlConnection = (HttpURLConnection) new URL(url).openConnection();
urlConnection.setRequestProperty("Accept-Encoding", "identity");
int tamañoRecurso=urlConnection.getContentLength();
```

- Para mostrar un recurso html en un `WebView` usa el método loadData de la clase WebView, tal y como lo hace este ejemplo.

- BitmapFactory permite leer una imagen de un InputStream

```
Bitmap bitmap = BitmapFactory.decodeStream (inputStream);
```

- Una imagen almacenada en un Bitmap se asigna a un ImageView mediante el método setImageBitmap.

```
imagen.setImageBitmap (bitmap);
```

Ejercicio 3 (6)

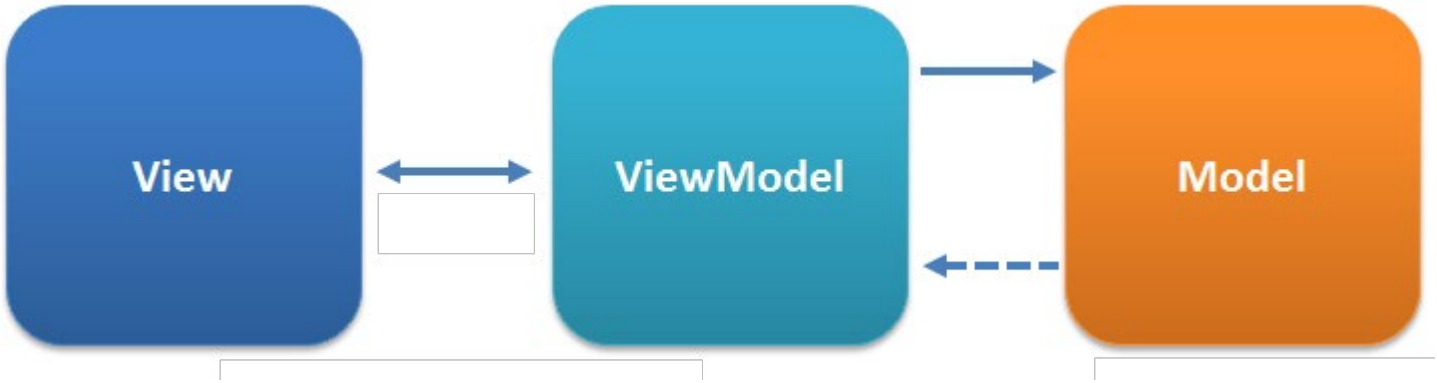
- Cancelar una tarea asíncrona:
  - En la clase de la GUI ejecutar el método **`interrupt()`** de la instancia del hilo
  - En la clase asíncrona (el hilo):
    - Crear un método, por ejemplo **`boolean esCancelado(){ }`**
    - En este método consultar si se ha solicitado cancelar el hilo mediante **`Thread.currentThread().isInterrupted()`**
    - Invocar ese método en los bucles para salir de ellos
    - No usar dentro de **`runOnUiThread()`**, pues este sería otro hilo

- Ocultar el teclado virtual:

```
InputMethodManager inputManager = (InputMethodManager)
        getSystemService(Context.INPUT_METHOD_SERVICE);
if (inputManager != null ) {
    inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
        InputMethodManager.HIDE_NOT_ALWAYS);
}
```

ViewModel (I)

- En el patrón de diseño MVVM (Model-View-ViewModel) el ViewModel actúa como intermediario entre modelo y vista:
  - View: Muestra datos y recoge interacción del usuario
  - Model: Lógica de negocio y acceso a datos
  - ViewModel: Intermediario entre Vista y Modelo. Mantiene el estado de la UI



https://developer.android.com/topic/libraries/architecture/viewmodel.html

ViewModel (2)

- Características de una clase `ViewModel`:
  - Se instancia una vez en el método `OnCreate()` de la actividad
  - Permanece en memoria, aunque la actividad se destruya por un cambio de orientación
  - Desaparece del sistema solo cuando la actividad finaliza*
  - No debe almacenar referencias a la actividad que lo crea, pues la instancia de la actividad puede cambiar (por ejemplo al rotar la pantalla)
- Puesto que en un `Bundle` no pueden almacenarse objetos grandes (como una imagen) que se puedan guardar/recuperar con `onSaveInstanceState()` / `onRestoreInstanceState()`, es la clase apropiada para guardar la información que pueda perderse en un cambio de orientación

*Si el S.O. finaliza la actividad en segundo plano por falta de memoria, es necesario guardar la información del ViewModel, en el método onDestroy(), en un lugar persistente

ViewModel (3)

```
public class MainActivity extends AppCompatActivity {
  protected ImageView imagen;

  // Atributo para guardar un objeto ViewModel que permite salvar el valor de objetos grandes
  // al destruirse la actividad como consecuencia de un cambio de orientación:
  private GuardarEstado guardarEstado;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    imagen= findViewById(R.id.imagen);

    // Obtener una referencia a la clase ViewModel.
    // Si ya existía el objeto, no se crea una nueva instancia
    guardarEstado = new ViewModelProvider(this).get(GuardarEstado.class);
    // Recuperar los datos almacenados en el ViewModel que queramos usar en la actividad
    if (guardarEstado.getImagen()!= null) // Podría interesar verificar si es la primera vez
      imagen.setImageBitmap (guardarEstado.getImagen());
  }
}
```

```
public class GuardarEstado extends ViewModel {
  private Bitmap imagen;
  public Bitmap getImagen() {
    return imagen;
  }
  public void setImagen(Bitmap imagen) {
    this.imagen = imagen;
  }
}
```

Ejercicio 4

Modificar el ejercicio 3 para que la imagen o el texto descargado no desaparezca en un cambio de orientación. No es necesario crear un diseño específico para la orientación horizontal

Cuando se pulse cancelar o se inicie la descarga, borrar los datos de ViewModel para que no salgan datos antiguos en un cambio de orientación