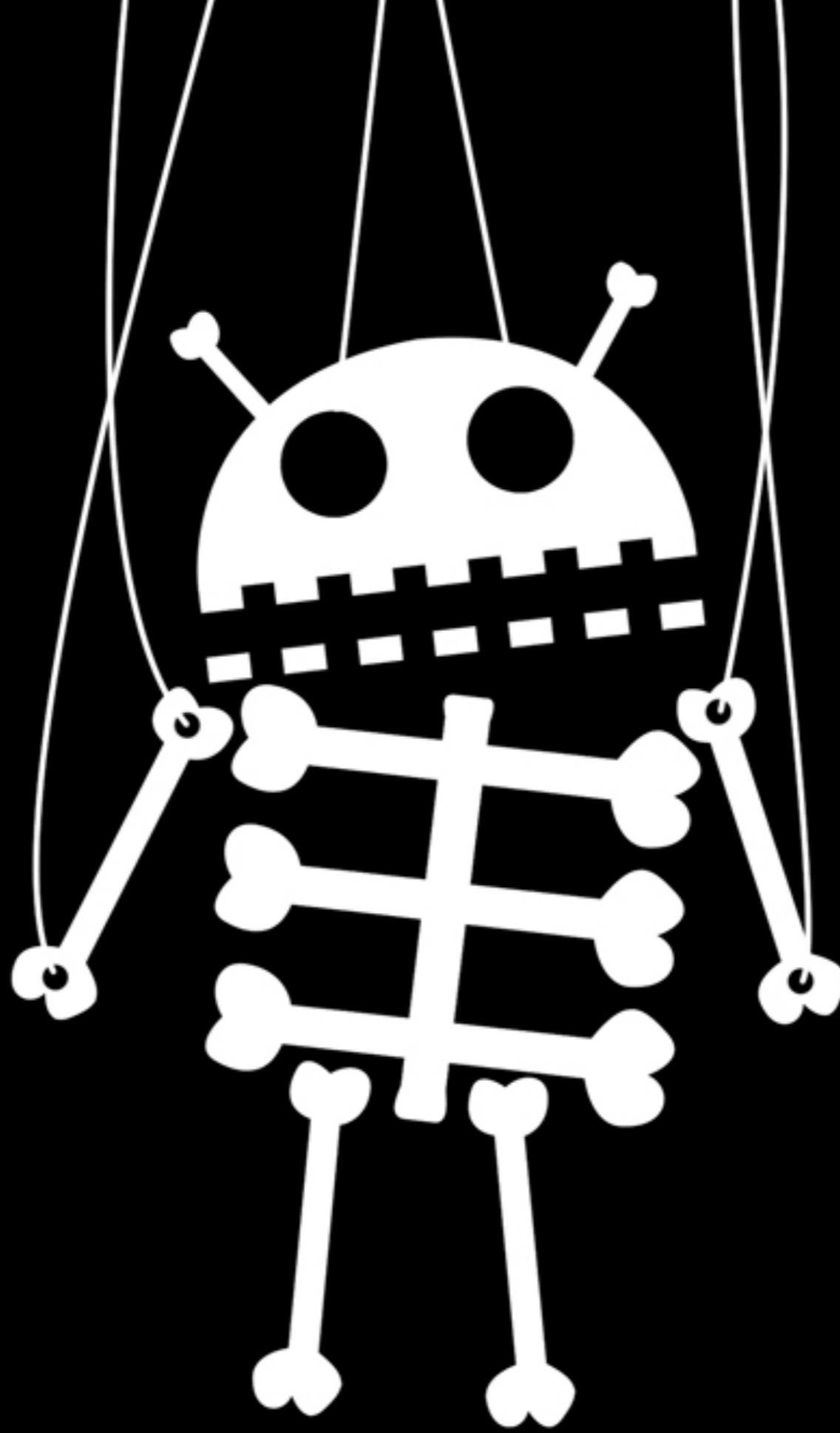




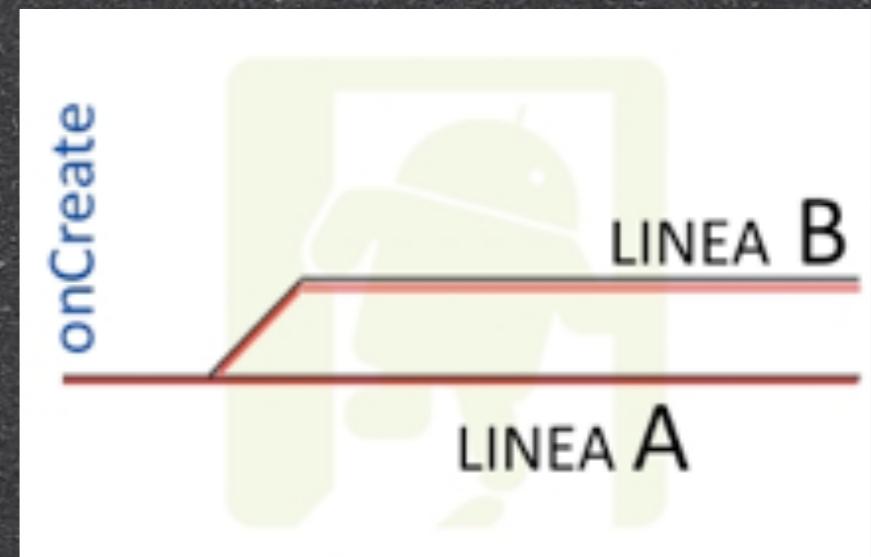
Introducción al Android

Hilos



¿Qué es un hilo?

- Es una porción de código que se ejecuta paralelamente.



¿Por qué utilizarlos?

- Uso de recursos que podrían bloquear hilos principal (UI).
- Mejora de la eficiencia de aplicaciones.

Al finalizar el hilo...

- El hilo puede simplemente terminarse (morir)
- El hilo desea volver al hilo principal (UI).
- Para esto se utilizan handlers

Handlers

- Nos permiten enviar y procesar Message y clases Runnable a su hilo asociado.
- El hilo asociado es el que realiza la creación del handler.

Dentro del Activity

Instanciamos el handler

```
final Handler handler = new Handler();
```

Creamos el Runnable

```
final Runnable ejecutarPosHilo = new Runnable(){

    @Override
    public void run() {
        Toast.makeText(MainActivity.this, "Se terminó hilo", Toast.LENGTH_LONG).show();
    }

};
```

Creamos y ejecutamos el hilo

```
private void crearHilo(){
    Thread hilo = new Thread(){
        public void run(){
            try{
                Thread.sleep(30000);
            }catch(Exception e){
                Log.e("Clase4", "Error en hilo:" + e.getMessage());
            }
            handler.post(ejecutarPosHilo);
        }
    };
    hilo.start();
}
```

¿Qué pasa si no estoy en mi hilo UI?

- Handler ejecuta el runnable en el hilo donde fue creado.
- Se utiliza el método runOnUiThread.
- Este método ejecuta el Runnable de todas formas en el hilo UI.

```
public class MainActivity extends Activity {
    //final Handler handler = new Handler();
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        crearHilo();
    }

    private void crearHilo(){
        Thread hilo = new Thread(){
            public void run(){
                try{
                    Thread.sleep(30000);
                }catch(Exception e){
                    Log.e("Clase4", "Error en hilo:" + e.getMessage());
                }
                runOnUiThread(ejecutarPosHilo);
            }
        };
        hilo.start();
    }

    final Runnable ejecutarPosHilo = new Runnable(){

        @Override
        public void run() {
            Toast.makeText(MainActivity.this, "Se terminó hilo", Toast.LENGTH_LONG).show();
        }
    };
}
```

AsyncTask

- Clase que permite implementar tareas que corren en segundo plano, para no bloquear la UI
- Para implementar una tarea, debemos extender a AsyncTask e implementar doInBackground
- Puede recibir parámetros y comunicar el progreso de la tarea

AsyncTask: ejemplo

```
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
     * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
     * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

AsyncTask: pasos de ejecución

Método	Hilo
onPreExecute	Principal
doInBackgroud	Secundario
onProgressUpdate	Principal
onPostExecute	Principal