19. El ciclo de vida de una actividad

Las actividades atraviesan una serie de estados desde el momento en que se crean hasta su destrucción. A lo largo de este recorrido se invocan automáticamente un conjunto de métodos estándar. La clase base `Activity` proporciona versiones por omisión de estos métodos que, opcionalmente, el programador puede sobrecargar. De hecho, ya has sobrecargado el método `onCreate()` para inflar la interfaz desde el fichero de diseño. Enseguida veremos cómo y para qué sobrecargar el resto de los métodos del ciclo de vida.

Una actividad puede estar:

- **Activa**: la actividad está en primer plano de la pantalla y tiene el foco.
- **En pausa**: la actividad sigue visible pero sin foco como, por ejemplo, cuando un diálogo la tapa parcialmente. Toda la información de estado se mantiene pero el sistema puede decidir matar la actividad en ciertas situaciones extremas.
- **Parada**: la actividad está cubierta completamente por otra actividad. Se mantiene la información de estado pero es probable que el sistema la mate si necesita memoria.

Una actividad en pausa o parada puede volver a estar activa. Todas estas transiciones vienen acompañadas por llamadas a un conjunto estándar de métodos como se puede ver en la Figura 1:

- **`onCreate()`**: llamado cuando la actividad se crea por primera vez. Este método se utiliza habitualmente para crear la interfaz de usuario, capturar referencias a las vistas de la interfaz y asignar escuchadores. Cuenta con un argumento de tipo `Bundle` (conjunto de pares clave-valor) con información del estado anterior de la actividad, que se utiliza para recrearla tras una rotación, por ejemplo. Siempre le sigue el método `onStart()`.
- **`onStart()`**: llamado cuando la actividad se vuelve visible para el usuario. En general le sigue el método `onResume()` u `onRestoreInstaceState()` cuando la actividad se recrea por ejemplo tras una rotación del dispositivo.
- **`onRestoreInstanceState()`**: llamado cuando la actividad se recrea a partir de un estado guardado en el argumento de tipo `Bundle`.
- **`onResume()`**: llamado cuando la actividad empieza a interactuar con el usuario. Sobrecarga este método para arrancar código que necesite ejecutarse cuando la actividad está en primer plano. La actividad pasa a estar *Activa*.
- **`onPause()`**: llamado cuando el sistema está a punto de reanudar una actividad previa o lanzar una nueva. También se llama cuando la actividad pasa a estar parcialmente visible, por ejemplo al abrir un diálogo. En estos casos la actividad pasa al estado *En pausa*. A este método le puede seguir `onResume()` u `onStop()`, dependiendo de lo que ocurra. Puedes sobrecargar `onPause()` para guardar información y detener código que debe ejecutarse tan solo cuando la actividad está en primer plano (animaciones, música,

etc). En casos extremos, la aplicación puede ser destruida sin que se realice la llamada a `onDestroy()`, como se puede ver en la Figura 1. Por lo tanto, el método `onPause()` es el adecuado para liberar recursos pues puede ser el último método del ciclo de vida que llegue a ejecutarse. Hay que tener en cuenta que, como la siguiente actividad no será reanudada hasta que `onPause()` termine, conviene que la ejecución del método sea lo más rápida posible.

- **`onSaveInstanceState()`**: llamado en ocasiones después de `onPause()`, como por ejemplo cuando se gira el dispositivo, y antes de que la actividad se mate de tal forma que se pueda recuperar su estado en `onCreate()` u `onRestoreInstanceState()`. El objeto de tipo `Bundle` creado se pasa a ambos métodos.
- **`onStop()`**: llamado cuando la actividad deja de ser visible completamente al usuario porque otra actividad ha sido reanudada y la está cubriendo.
- **`onDestroy()`**: llamado cuando la actividad se destruye. El método `isFinishing()` permite averiguar si la destrucción se debe a que se invocó el método `finish()` o bien fue el sistema quien mató la actividad.
- **`onRestart()`**: llamado cuando la actividad ha sido parada y se reanuda posteriormente.

Cuando se sobrecargan estos métodos hay que llamar a la implementación en la superclase, de lo contrario se lanzará una excepción.

19.1 Dos ciclos de ejecución habituales

En la Figura 1 se pueden apreciar tres ciclos que ocurren durante la ejecución de una actividad. Dos de ellos son los más frecuentes: El ciclo `onPause()` -> `onResume()` y el ciclo `onPause()` -> `onStop()` -> `onRestart()` -> `onStart()` -> `onResume()`. Veamos qué se suele hacer en cada uno de estos ciclos:

- **Pausando y reanudando una actividad:** El método `onPause()` se invoca cuando la aplicación pasa a segundo plano. En este método se deben parar acciones que consuman CPU como videos, animaciones, etc. y liberar recursos que consuman batería (como accesos a GPS, cámara, etc.). Además se deben guardar los cambios que el usuario espera que sean permanentes. El método `onResume()` es la contrapartida de `onPause()` y se deben inicializar elementos que se liberen en `onPause()`. Hay que tener en cuenta que `onResume()` se ejecuta también al crear una actividad y no siempre tras un `onPause()`.
- **Parando y reiniciando una actividad:** El método `onStop()` se invoca cuando la actividad está completamente oculta al usuario. Este método debe usarse para operaciones que requieran mayor uso de CPU como guardar información en la base de datos. El método `onStart()` se puede usar para hacer comprobaciones sobre las capacidades del sistema (como comprobar si el GPS está activado). Algunos casos habituales en los que la actividad se para y se reinicia son:
    - Cuando el usuario navega a otra aplicación y luego vuelve.

- Cuando la actividad abre otra actividad y luego el usuario pulsa el botón back.
- Cuando se recibe una llamada y luego se cuelga.

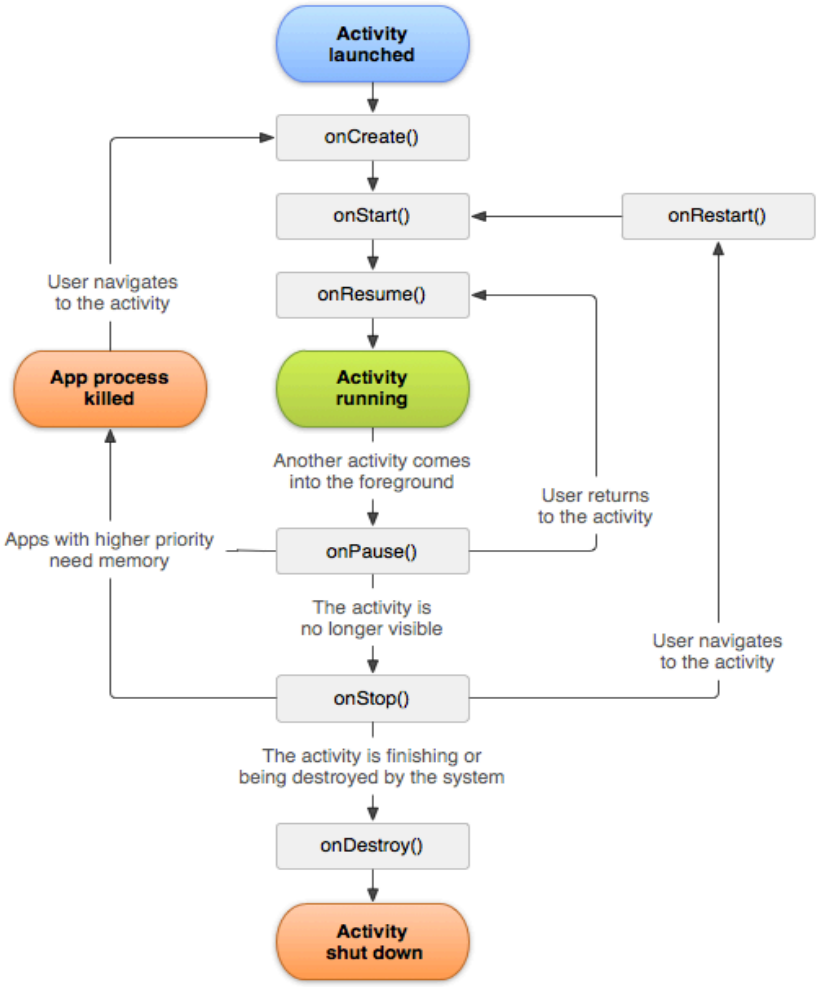


Figura 1. Ciclo de vida de una actividad[1].

19.2 El ciclo de vida en LogCat

El siguiente proyecto, `CCC19`, sirve para entender en qué situaciones se llama a cada uno de los métodos del ciclo de vida de una actividad. Puedes utilizar como fichero de diseño el generado automáticamente. El código Java es el siguiente:

[1] Figura perteneciente al proyecto Android Open Source y utilizada de acuerdo a los términos descritos en Creative Commons 2.5 Attribution License.

/src/MainActivity.java

```
package es.uam.eps.android.ccc19;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {

    private void log(String text) {
        Log.d("LifeCycleTest", text);
    }
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        log("created");
    }
    public void onStart() {
        super.onStart();
        log("started");
    }
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        log("onRestoreInstanceState() called");
    }
    protected void onResume() {
        super.onResume();
        log("resumed");
    }
    protected void onPause() {
        super.onPause();
        log("paused");
    }
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        log("onSaveInstanteState() called");
    }
    protected void onStop() {
        super.onStop();
        log("stopped");
    }
    protected void onDestroy() {
        super.onDestroy();
        log("destroyed");
    }
    protected void onRestart() {
        super.onRestart();
        log("restarted");
    }
}
```

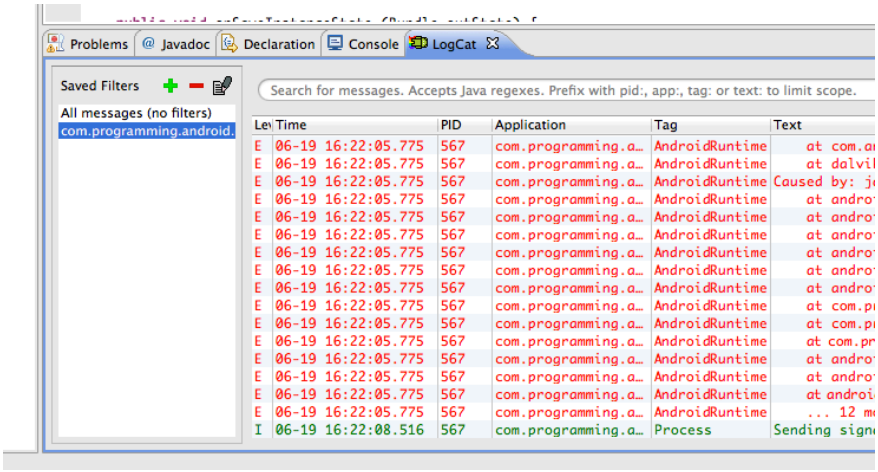
La primera instrucción de los métodos sobrecargados ha de ser una llamada al método implementado en la clase `Activity`:

```
super.onCreate(savedInstanceState);
```

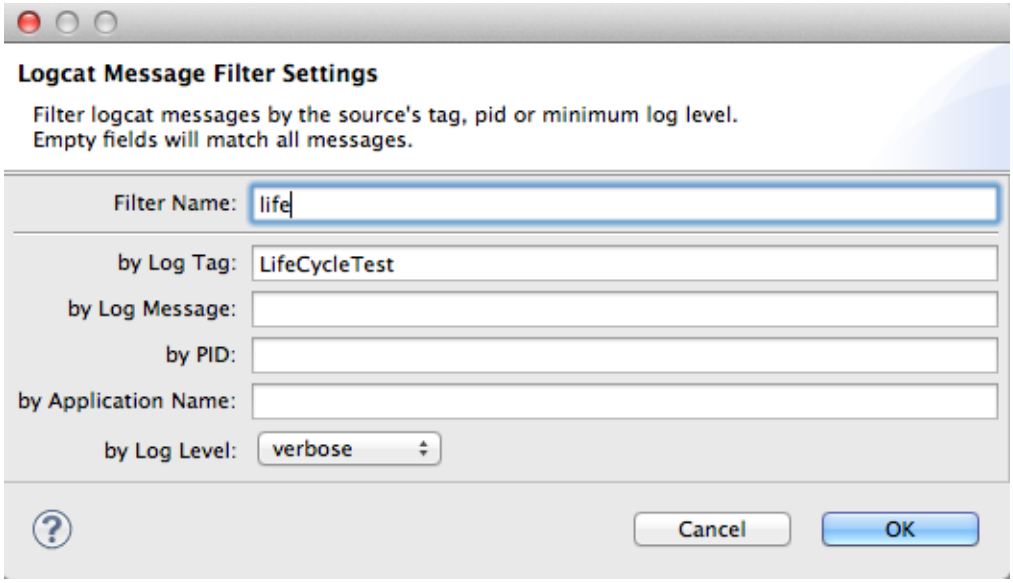
Por ejemplo, en `onCreate()`, el método de la superclase utiliza el estado almacenado de las vistas en el objeto de tipo `Bundle` para recrear la jerarquía de vistas de la actividad. El único método del código anterior no heredado de la clase base `Activity` es el método privado `log()`, el cual envía un mensaje marcado como `LifeCycleTest`:

```
Log.d ("LifeCycleTest", text);
```

Para ver estos mensajes, abre la vista `LogCat` de Eclipse pulsando `Window->Show View->LogCat`. Debería aparecer una nueva pestaña denominada `LogCat`:



Como puedes ver el número de mensajes es muy elevado. Para centrarnos en los que emite nuestro programa resulta cómodo añadir un filtro pulsando el signo + en verde. Tienes que elegir un nombre de filtro (*Filter Name*), una marca (*tag*) (`LifeCycleTest` en nuestro caso) y pulsar `OK`. Este filtro selecciona los mensajes marcados como `LifeCycleTest`:



Ejecuta la aplicación siguiendo los pasos 1 a 5 y responde a las cuestiones estudiando los mensajes de la ventana LogCat:

1. ¿Qué métodos se ejecutan cuando se llama a la aplicación por primera vez?
2. Pulsa el botón Hacia Atrás (*Back*). ¿Qué métodos se ejecutan? ¿Se destruye la actividad?
3. Ejecuta de nuevo la aplicación.
4. Baja la cortina de notificaciones y pulsa en cualquier notificación pendiente. ¿Qué métodos se ejecutan en este caso?
5. Vuelve a la actividad pulsando el botón Hacia Atrás. ¿Qué métodos se ejecutan en esta ocasión?
6. Gira el dispositivo y observa los métodos ejecutados.