



Ciclo de vida

Procesos



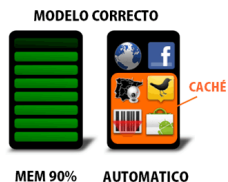
- Por seguridad, Android implementa el principio de privilegios mínimos con el siguiente mecanismo:
 - El sistema operativo Android es un sistema Linux multiusuario en el que, por defecto, **cada aplicación es un usuario diferente** (el UID es utilizado sólo por el sistema y es desconocido para la aplicación). El sistema establece los permisos para todos los archivos de una aplicación de forma que sólo el UID de usuario asignado a esa aplicación puede acceder a ellos.
 - Cada proceso tiene su propia máquina virtual (VM), por lo que el código de una aplicación se ejecuta en forma aislada de otras aplicaciones. De forma predeterminada, **cada aplicación se ejecuta en su propio proceso** de Linux. Android inicia el proceso cuando cualquiera de los componentes de la aplicación necesita ser ejecutado y cierra el proceso cuando ya no es necesario o cuando el sistema debe recuperar la memoria para otras aplicaciones.



Uso de RAM



UNUSED RAM IS WASTED

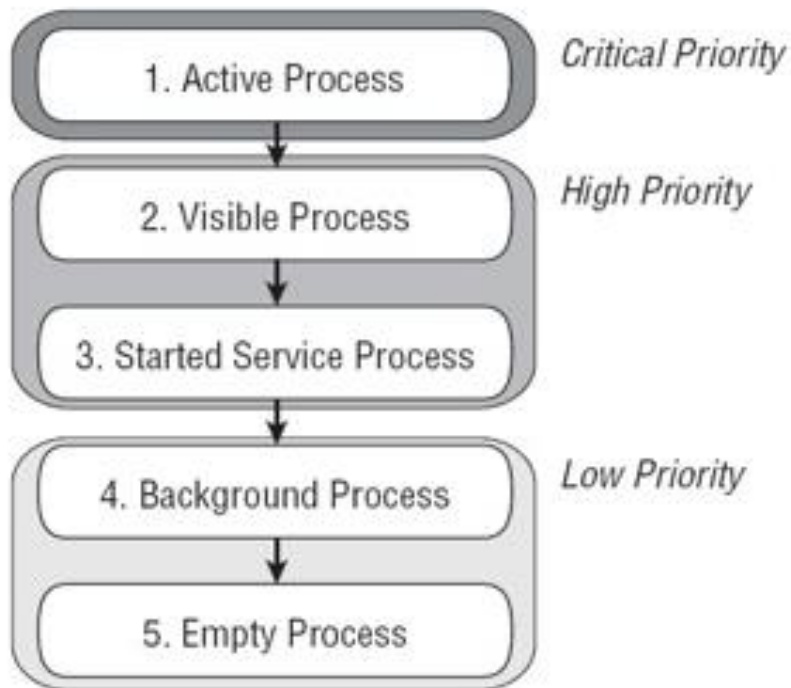


- Android utiliza casi el **100%** de la memoria RAM del dispositivo (*dejando apenas unos MB libres*). Se considera que **la memoria sin usar es memoria desperdiciada**.
- En un caso ideal todas las aplicaciones Android iniciadas por el usuario permanecen en la memoria, lo que hace que reiniciar las aplicaciones sea más rápido.
- Pero en realidad la memoria disponible en un dispositivo Android es limitada. Para **gestionar estos recursos limitados** se permite al sistema Android interrumpir los procesos en ejecución: si necesita memoria RAM para otras aplicaciones, simplemente empieza a **liberar aplicaciones** hasta disponer de la cantidad que necesita.
- Por tanto, una característica peculiar en Android es que el tiempo de vida de un proceso no es controlado directamente por la aplicación y/o usuario. **Es el sistema quien decide y determina el tiempo de vida** basándose en el uso y capacidades del sistema.

Prioridad de los procesos

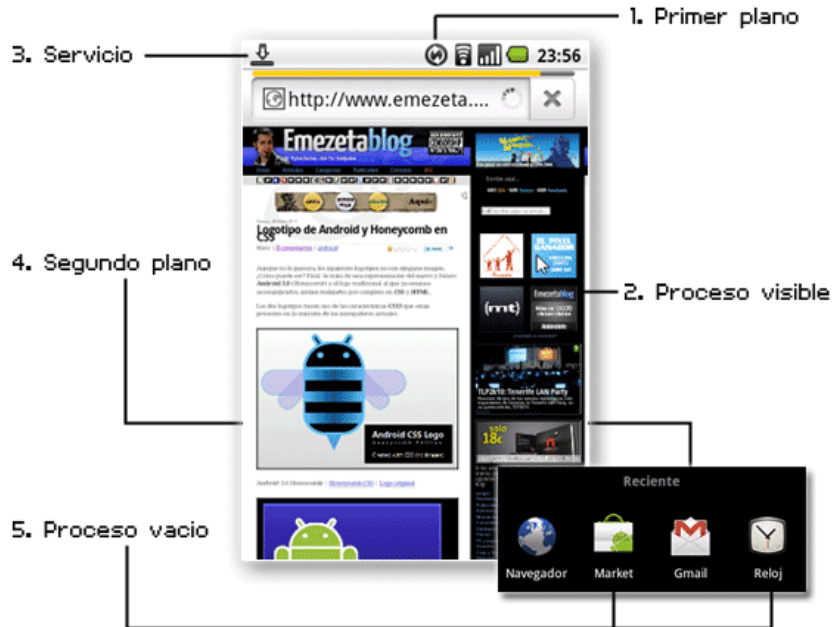


- Para determinar que procesos deberán ser eliminados ante una condición baja de memoria, Android prioriza los procesos bajo una jerarquía para asignar a cada proceso una importancia en el sistema:



- Un proceso **activo** o en primer plano: el de la aplicación en la que el usuario está interactuando con una actividad. En general, existen solo unos pocos procesos en primer plano en un momento dado.
- Un proceso **visible**: El usuario no está interactuando con la actividad, pero la actividad sigue siendo parcialmente visible.
- Los procesos de **servicio** no están directamente relacionados con lo que el usuario ve y por lo general realizan tareas que el usuario necesita (descarga de datos en red, copiar elementos al dispositivo desde el pc, etc.)
- Un proceso en **segundo plano**: se mantiene una actividad que no es visible actualmente para el usuario
- Un proceso **vacío**: es un proceso de una aplicación sin ningún tipo de componentes activo. La única razón para mantener éste tipo de procesos vivos es con fines de almacenamiento en caché, para mejorar el tiempo de inicio de la próxima vez que un componente necesita ejecutarse en él (es lo que hacen muchas apps al "morir").

Ejemplo

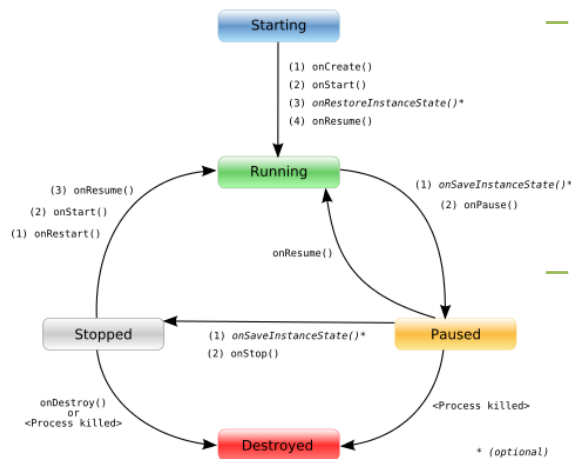


- **Primer plano:** La tarea se está realizando ahora mismo. (Ej: Ver una pág. Web en el navegador)
- **Procesos visibles:** Se visualiza la tarea en la pantalla pero no se puede interactuar con ella. (Ej: Al abrir el diálogo de una notificación "tapa" la pantalla)
- **Servicios:** Procesos "encargados" de alguna tarea. (Ej: Descargar un archivo...)
- **Segundo plano:** Una tarea interrumpida que puede ser reanudada o no (Ej: Cliente de Gmail)
- **Procesos vacíos:** Una tarea que se supone finalizada, pero se mantiene porque puede utilizarse en breve (Ej: Google Play, Reloj)

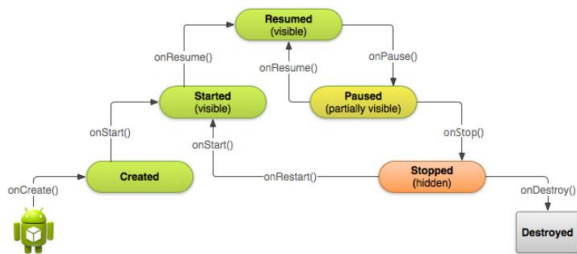
Estados de una actividad



- Una actividad, generalmente trabaja en uno de los siguientes cuatro estados en algún momento:
 - Activa:** Cuando la actividad ha sido iniciada por el usuario, está actualmente ejecutándose y se encuentra en primer plano.
 - Pausada:** Cuando la actividad ha sido iniciada por el usuario, está actualmente ejecutándose y está visible, pero una notificación o alguna otra cosa está superpuesta en alguna parte de la pantalla. Durante este estado, el usuario puede ver la actividad, pero no es posible interactuar con ella.
 - Parada:** Cuando la actividad ha sido iniciada por el usuario, sigue ejecutándose pero se encuentra oculta por otras actividades que se han lanzado. Cuando una actividad se encuentra en este estado, la actividad no es capaz de mostrar información significativa para el usuario de manera directa, pero puede hacerlo mediante el uso de notificaciones.
 - Terminada:** La actividad entra en este estado ya sea porque nunca se inició (escenario que se da después de que el usuario reinicia el teléfono) o porque fue terminada por el sistema por la falta de memoria disponible o por el usuario.



Métodos del ciclo de vida

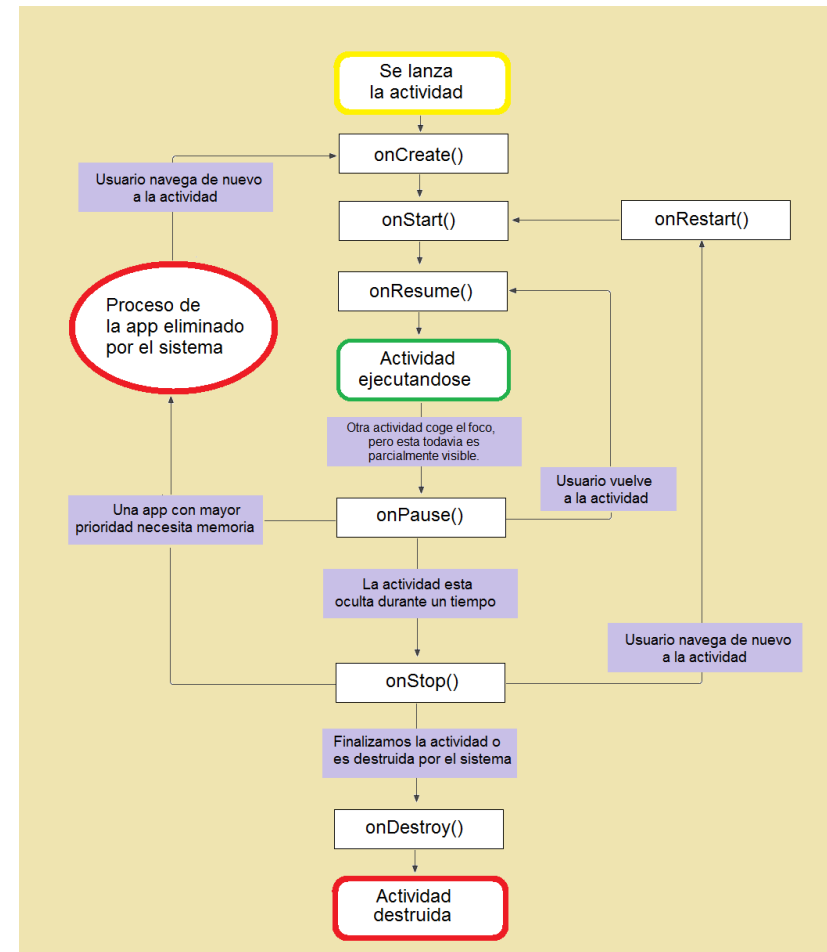


- Durante la vida de una actividad, el sistema llama a un conjunto básico de métodos en una secuencia similar a una pirámide escalonada.
- La parte superior de la pirámide es el punto en el que la actividad se ejecuta en primer plano y el usuario puede interactuar con ella.
- La implementación de los métodos de ciclo de vida de la actividad **garantiza el comportamiento adecuado de la aplicación**, incluyendo que:
 - No se bloquee si el usuario recibe una llamada telefónica o cambia a otra aplicación.
 - No consuma recursos valiosos del sistema cuando el usuario no está utilizándola activamente.
 - No se pierden datos si el usuario la abandona y vuelve a ella posteriormente.
 - No se bloquea, ni se pierden datos al cambiar la orientación del dispositivo.
- Dependiendo de la complejidad de la actividad, es probable que no se necesite implementar todos los métodos de ciclo de vida.
- El objetivo clave de la gestión del ciclo de vida de la actividad es **garantizar que el estado de la actividad se guarda y restaura en los momentos apropiados** (cuando se habla de estado nos referimos a los datos con los que se está trabajando dentro de la actividad y a la apariencia de la interfaz de usuario).

Ciclo de vida de una actividad



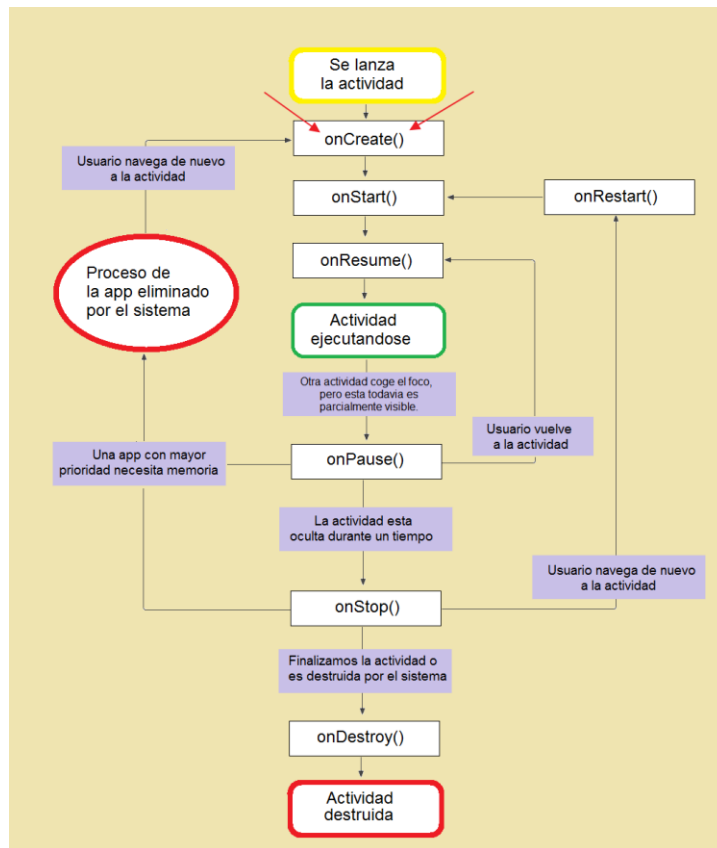
```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // La actividad está siendo creada.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // Invocado justo antes de hacerse visible
    }
    @Override
    protected void onResume() {
        super.onResume();
        // La actividad se ha vuelto visible (ahora está "Active").
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        // La actividad va a iniciarse otra vez, después de haber sido Stopped.
    }
    @Override
    protected void onPause() {
        super.onPause();
        // La actividad va a pasar al estado Paused.
    }
    @Override
    protected void onStop() {
        super.onStop();
        // La actividad no es visible (ahora está "Stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // La actividad va a ser eliminada
    }
}
```



onCreate()



- El sistema llama a este método cuando la Activity es llamada por primera vez.

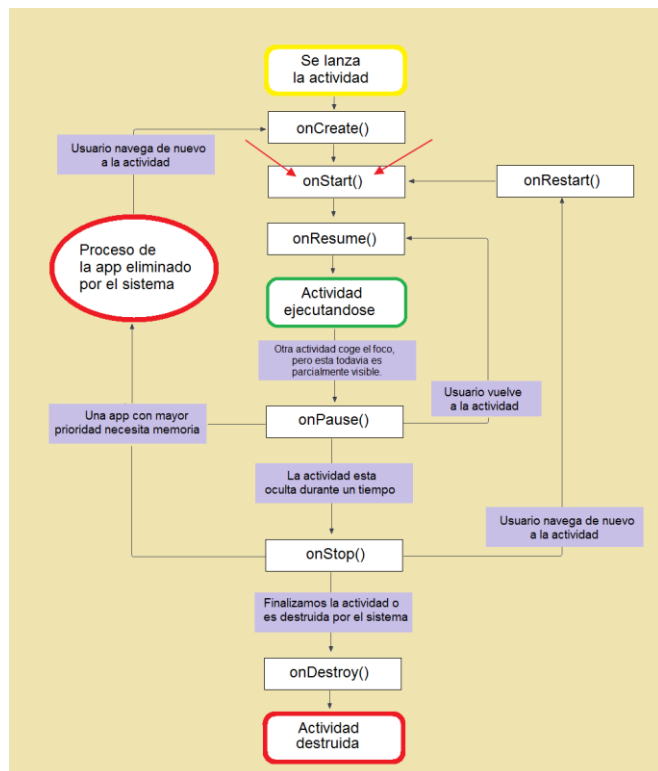


- Aquí es donde debe crearse la inicialización normal de la aplicación (crear vistas, cargar los datos a visualizar, etc.: llamaremos a los métodos **setContentView** y **findViewById**,...).
- Este método da acceso al estado de la aplicación cuando se cerró ya que recibe un objeto de tipo Bundle: **onCreate(Bundle savedInstanceState)**. Si no es la primera vez que se crea esta actividad en la ejecución actual de la aplicación, esto es, si la actividad ya ha existido antes pero fue destruida, el objeto **Bundle** puede contener lo necesario para restaurar su estado anterior de forma que el usuario se encuentre la actividad tal y como se la espera. Si no es así, el objeto será null.

onStart()



- Este método se ejecuta cuando el sistema tiene intención de hacer visible la actividad, aunque luego no llegue a ocurrir realmente.

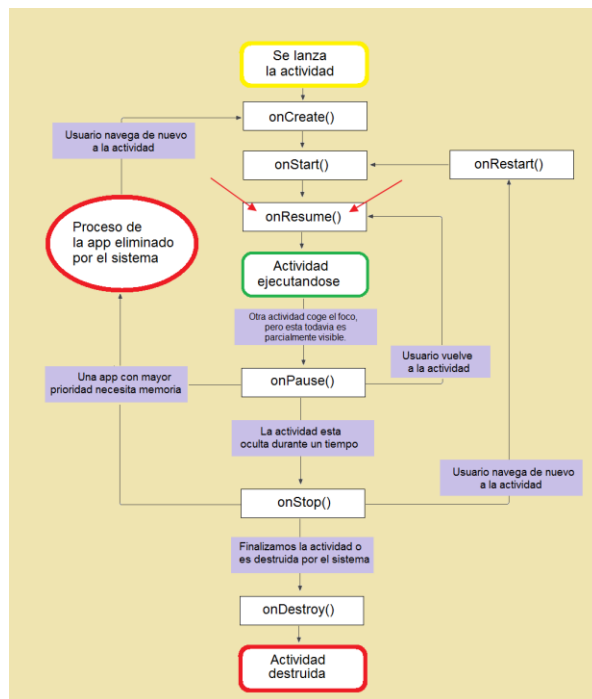


- Se suele utilizar para **actualizar la interfaz** de usuario: animaciones, temporizadores, localización GPS etc.
- Se hace en este método en lugar de en onCreate para no consumir recursos de forma innecesaria hasta que realmente hay intención de mostrar la actividad.

onResume()



- Este método se ejecuta justo antes de que la actividad sea completamente visible y el usuario pueda empezar a interactuar con ella.
- Siempre se va a ejecutar, tanto en el primer inicio, como después de pausarla o pararla.

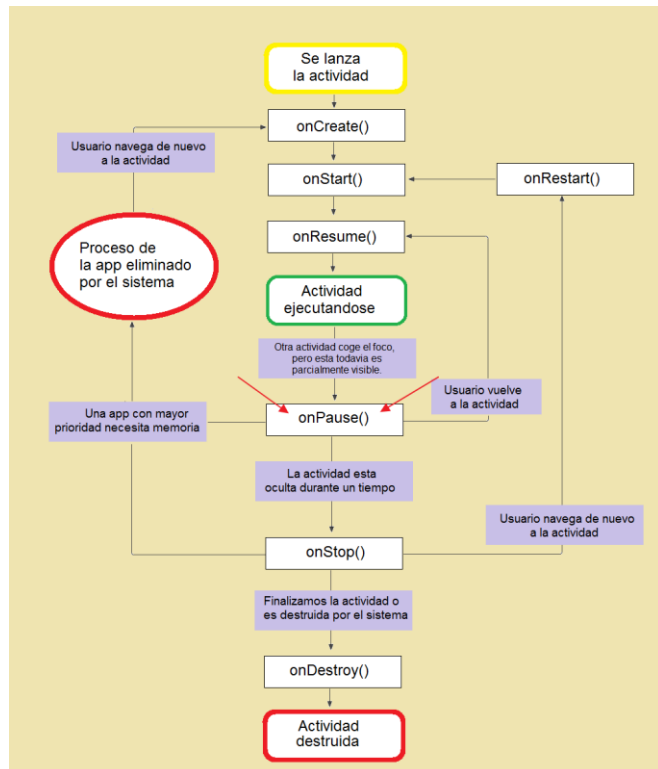


- **Es el mejor sitio donde recuperar datos guardados** (en ficheros, en preferencias, etc.), iniciar animaciones, acceder a recursos que se tienen que utilizar de forma exclusiva (como la cámara), etc. De esta forma no utilizamos los recursos más valiosos hasta que realmente van a ser necesarios. Hay que tener en cuenta que aunque se ejecute **onStart**, la actividad podría no llegar a visualizarse, por lo que no tendría sentido bloquear recursos o activar procesos que actualizan una interfaz de usuario que nadie va a ver.

onPause()



- El sistema llama a este método cuando recibe la indicación de que el usuario deja de poder interactuar con la actividad actual. Esto no significa que siempre la finalicemos, puede ser que se quede en segundo plano, abriendo una nueva actividad.

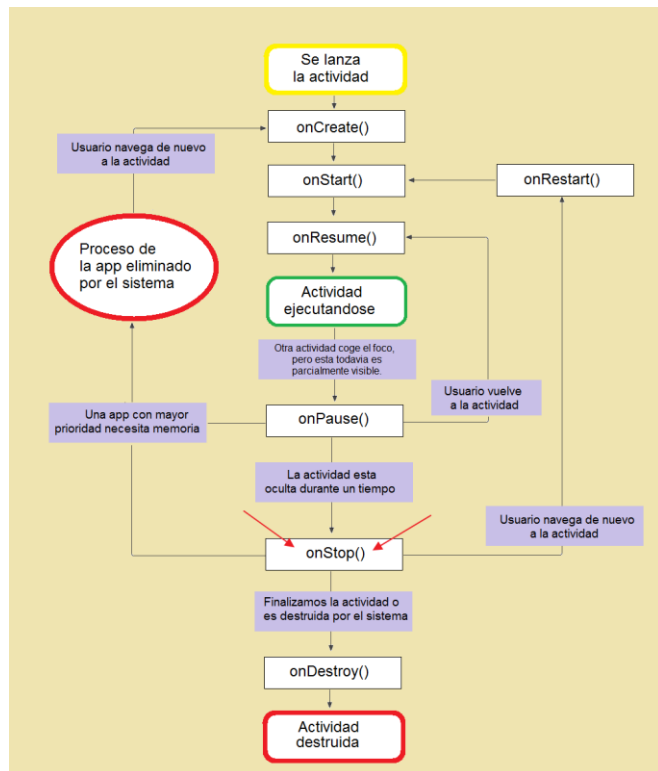


- Si nos fijamos en el dibujo, este método siempre es llamado, por tanto **aquí salvaremos los datos que no queramos que se pierdan porque puede ser que onStop y onDestroy nunca sean llamados y perderíamos los datos.**
- Es también el sitio ideal **para detener todo lo que se ha activado** en onResume y liberar los recursos de uso exclusivo (dado que podría necesitarlos la nueva actividad).
- En cualquier caso, la ejecución de este método debería ser lo más rápida posible, puesto que el usuario no podrá utilizar la nueva actividad hasta que finalice.

onStop()



- Se llama a este método cuando la actividad ha sido ocultada totalmente durante un tiempo por otra actividad que ya está interactuando con el usuario.

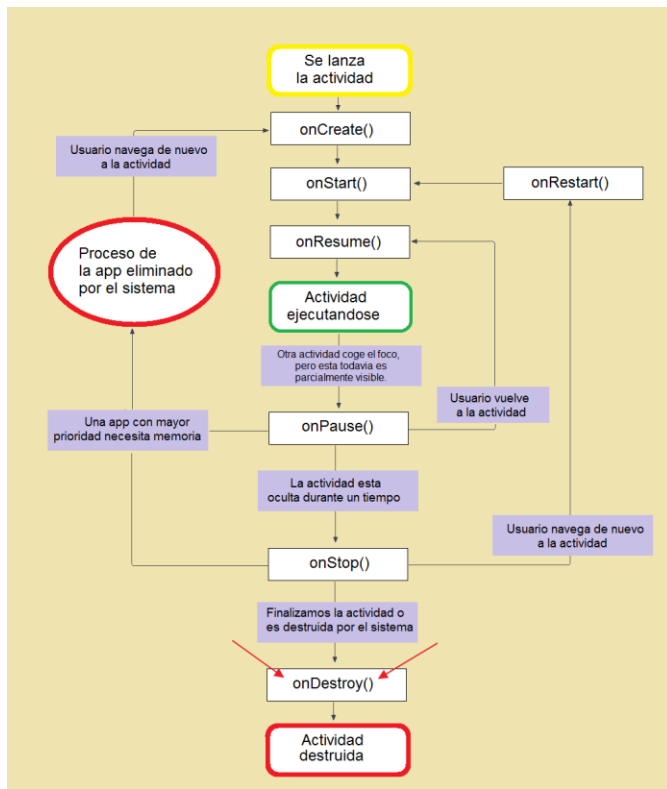



- Aquí se suelen **destruir** los procesos creados en el método onStart, para liberar recursos y permitir que la actividad que está interactuando actualmente con el usuario pueda ir lo más fluida posible.
- Si el sistema necesita memoria para una app con mayor prioridad puede eliminar procesos de actividades pausadas y este método no llega a ejecutarse.**

onDestroy()



- El sistema ejecuta este método cuando ya no tiene intención de reutilizar más la actividad.

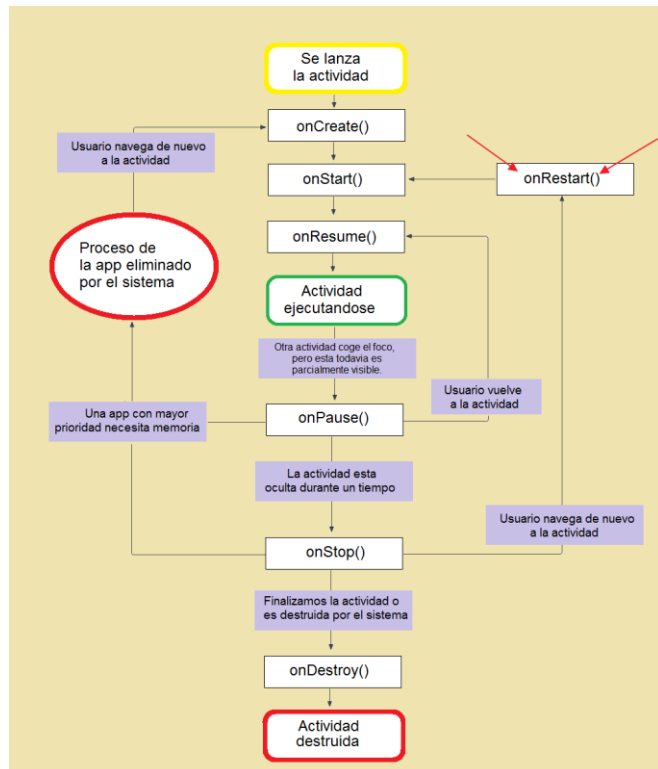


- Esto ocurre por los requerimientos de memoria que tenga el sistema o porque de manera explícita el usuario manda a llamar este método desde otro método del ciclo de vida haciendo uso del método finish() o bien porque pulsa el botón .
- Si se quiere volver a ejecutar la Activity arrancaría un nuevo ciclo de vida.
- Aquí ya no se puede hacer otra cosa que **destruir** los objetos, hilos y demás que hayamos creado en onCreate, **para no dejar basura**.

onRestart()

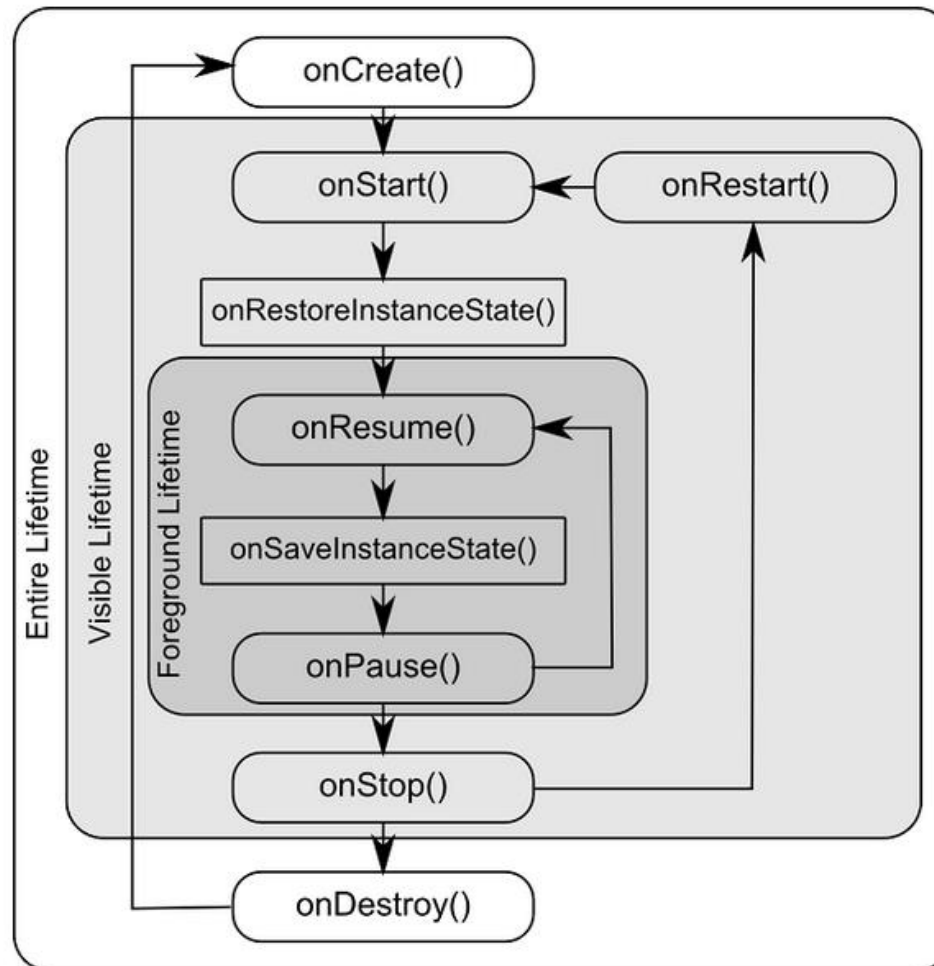


- Se ejecuta cuando la Activity ha sido parada y se quiere volver a utilizar.
- Después de este siempre se ejecutará onStart, por tanto, **este método no se suele sobrescribir.**



- Su uso no es tan habitual como el del resto de métodos, pero puede ser útil en algunos casos donde no compensa destruir ciertos elementos en onStop para luego tener que crearlos de nuevo en onStart, por cuestiones de eficiencia o porque eso provocaría el reinicio no deseado de alguna operación (consultas a una base de datos, por ejemplo). En esos casos lo que se hace es crear los procesos en onStart sólo la primera vez, detenerlos sin destruirlos en onStop y reanudarlos de nuevo en onRestart.

Ciclo de vida y estado de una actividad



Usando los métodos del ciclo de vida

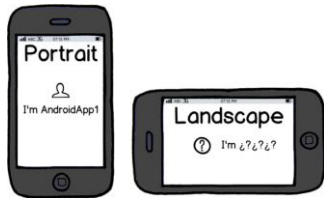


- En cualquier actividad, es conveniente programar los métodos del ciclo de vida y no solo el método onCreate().
- Por ejemplo, un audio que se está reproduciendo puede interesar (??) que se detenga al minimizar y que continúe al reanudarse la actividad.
- Cuando una actividad se reanuda lo habitual es que dicha actividad haya permanecido en memoria y continúe su ejecución sin alteraciones.
- Pero, en situaciones de escasez de memoria, es posible que el sistema haya eliminado el proceso que ejecutaba la actividad.
- En este caso, el proceso será ejecutado de nuevo, pero se habrá perdido su estado, es decir, se habrá perdido el valor de sus variables y el puntero de programa.
- Como consecuencia, si el usuario, por ejemplo, estaba reproduciendo un audio en un punto determinado perderá esta información.
- Es necesario guardar el estado de la actividad antes de que sea destruida para que pueda recuperarse en la nueva ejecución.
- También, cuando se ejecuta una actividad sensible a la inclinación/orientación del dispositivo (un juego es el ejemplo más claro) o cambiamos el idioma o abrimos el teclado, se presenta un problema similar al anterior. La actividad es destruida y vuelta a construir con las nuevas dimensiones de pantalla. Antes de que la actividad sea destruida también resulta fundamental guardar su estado.

Instancia del estado de la actividad



- Una actividad (en realidad, el proceso que la ejecuta) puede ser destruida por el sistema:
 - debido a **las limitaciones de memoria**
 - por **el cambio de orientación o de lenguaje del dispositivo** (se destruye y vuelve a crear la actividad debido a que la configuración ha cambiado y es posible que se tengan que cargar otros recursos).
- En ambos casos, es necesario que el sistema sea **capaz de recordar** que la actividad existía anteriormente y si el usuario reinicia de nuevo dicha actividad que no se pierdan los datos que ya se conocían.
- Si una actividad ejecuta sus ciclos de vida correctamente y guarda su estado actual la destrucción de su proceso no tendrá un efecto visible en la experiencia del usuario ya que cuando el usuario vuelva a esa actividad, la actividad restaurará la totalidad de su estado visible.
- El conjunto de datos guardados que el sistema utiliza para restaurar el estado anterior se denomina "**instancia del estado**" y es una colección de pares de valores-claves almacenada en un objeto de tipo Bundle.




Guardar y recuperar el estado de una actividad

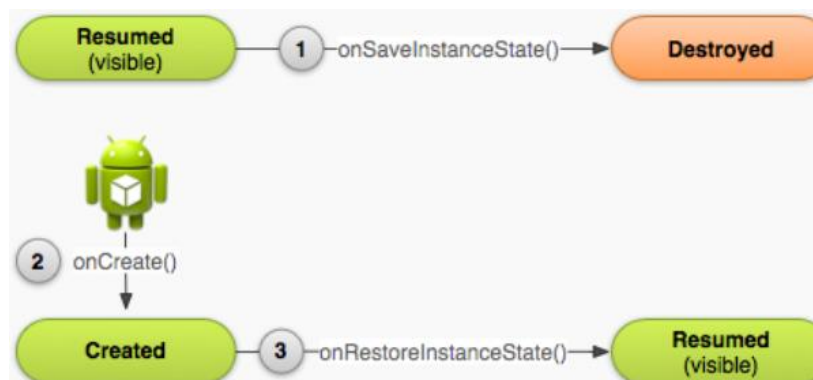


- Manera 1:
 - En caso de tratarse de información extremadamente sensible, se recomienda el uso del método **onPause()** para guardar el estado y **onResume()** para recuperarlos. Por supuesto, dicha información sensible ha de almacenarse en un medio permanente como un fichero, una base de datos.
- Manera 2:
 - También tenemos otra posibilidad que, aunque no tenga una fiabilidad tan grande, resulta mucho más sencilla. Consiste en utilizar los siguientes métodos:
 - **onSaveInstanceState(Bundle)**: Se invoca para permitir a la actividad guardar su estado. En Bundle guardaremos lo valores necesarios. Ojo, si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.
 - **onRestoreInstanceState(Bundle)**: Se invoca para recuperar el estado guardado por onSaveInstanceState().
 - **onCreate(Bundle)**: De forma alternativa podemos verificar si hay un estado guardado en la creación de la actividad.
 - La ventaja de usar estos métodos es que no tenemos que buscar un método de almacenamiento permanente, es el sistema quien hará este trabajo. El inconveniente es que el sistema no nos garantiza que sean llamados. En casos de extrema necesidad de memoria se podría destruir nuestro proceso sin llamarlos.

Guardar instancia del estado



- **onSaveInstanceState(Bundle datosAGuardar):** Se ejecuta antes de destruir la actividad. Este método tiene como parámetro un Bundle, donde **guardaremos** la información que deseemos. **Este método no se ejecuta cuando el usuario pulsa**  .
- **onRestoreInstanceState(Bundle savedInstanceState):** Este método solo será ejecutado, si se ha guardado información con onSaveInstanceState y siempre antes de que sea visible la actividad. Tiene como parámetro el Bundle con los datos guardados para que podamos **recuperarlos** y hacer lo que queramos con ellos. *Este segundo método no es necesario sobrescribirlo, ya que el Bundle con los datos a restaurar también se pasa como parámetro al onCreate y podemos recuperarlos allí. Hay que tener en cuenta que onCreate se ejecuta siempre, por tanto hay que comprobar que el Bundle que se nos pasa, tiene información antes de intentar recuperarla o puede producir una excepción.*



Ejemplo



@Override

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState); // Always call the superclass first  
    // Check whether we're recreating a previously destroyed instance  
    if (savedInstanceState != null) {  
        // Restore value from saved state  
        mCurrentScore = savedInstanceState.getInt("playerScore");  
        mCurrentLevel = savedInstanceState.getString("playerName");  
    } else {  
        // Probably initialize with default values for a new instance  
    }  
    ...  
}
```

@Override

```
public void onSaveInstanceState(Bundle savedInstanceState) {  
    super.onSaveInstanceState(savedInstanceState); // Always call the superclass first  
    // Save values  
    savedInstanceState.putInt("playerScore", mCurrentScore);  
    savedInstanceState.putString("playerName", mCurrentName);  
}
```

Fijando la orientación



- Se puede definir que en una actividad sólo se utilice una orientación específica de la pantalla a través del fichero AndroidManifest:

```
<activity
```

```
    android:name="com.pdm.ejercicios.ejer.MainActivity"
```

```
    android:label="@string/app_name"
```

```
    android:screenOrientation="landscape" >
```

```
    <intent-filter>
```

```
        <action android:name="android.intent.action.MAIN" />
```

```
        <category android:name="android.intent.category.LAUNCHER" />
```

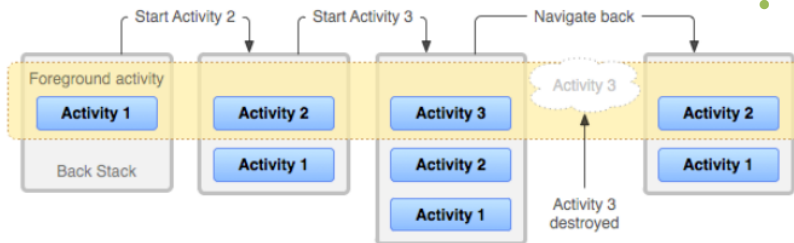
```
    </intent-filter>
```

```
</activity>
```


Tareas y pila de histórico

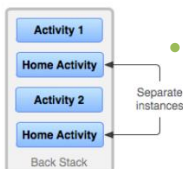
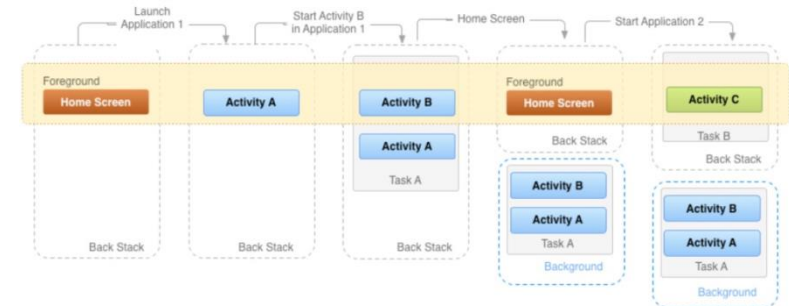


- Una tarea ("task") es una colección de actividades con las cuales interactúa el usuario cuando quiere realizar determinado trabajo. No tienen porque ser parte de la misma aplicación. La mayoría de las tareas empiezan en la pantalla principal de una aplicación.



• Cuando en una tarea se crea una nueva Activity, el sistema guarda su estado y la almacena en una pila de histórico ("back stack") tipo LIFO (el último que entra es el primero que sale). Así cuando se pulsa el botón de atrás se libera la actividad actual y muestra la anterior que había en la pila restaurando su estado. Para cada tarea existe una pila de histórico.

- Una tarea puede moverse a un "segundo plano" cuando el usuario comienza otra tarea. Mientras está en un segundo plano, todas las actividades en la tarea se paran, pero el histórico para la tarea ("back stack") se mantiene intacto.



• Dado que las actividades en el back stack nunca son reorganizadas, si la aplicación permite que los usuarios inicien una actividad concreta desde más de una actividad, se crea una nueva instancia de esa actividad y se mete en la pila (en vez de traer a la parte de arriba de la pila a una instancia anterior de la actividad).

Gestión de tareas



- La manera en la que por defecto Android gestiona las tareas y el "back stack", funciona muy bien para la mayoría de las aplicaciones.
- Sin embargo, a veces, se puede querer interrumpir el comportamiento normal.
- Quizás se necesite en la aplicación:
 - que una actividad inicie una nueva tarea cuando se inicie (en vez de colocarse dentro de la tarea en curso)
 - o que cuando se inicie una actividad, se traiga hacia delante una instancia ya existente (en vez de crear una nueva instancia en la parte superior del back stack)
 - o que cuando el usuario deja la tarea, se eliminen todas las actividades exceptuando la actividad raíz.
- Se puede conseguir con los atributos del elemento `<activity>` del archivo `AndroidManifest` o con los flags en el intent que se pasan al método `startActivity()`.

- Atributos

- *taskAffinity*
- *launchMode*
- *allowTaskReparenting*
- *clearTaskOnLaunch*
- *alwaysRetainTaskState*
- *finishOnTaskLaunch*

- Flags

- *FLAG_ACTIVITY_NEW_TASK*
- *FLAG_ACTIVITY_CLEAR_TOP*
- *FLAG_ACTIVITY_SINGLE_TOP*

Definir modo de inicio en AndroidManifest



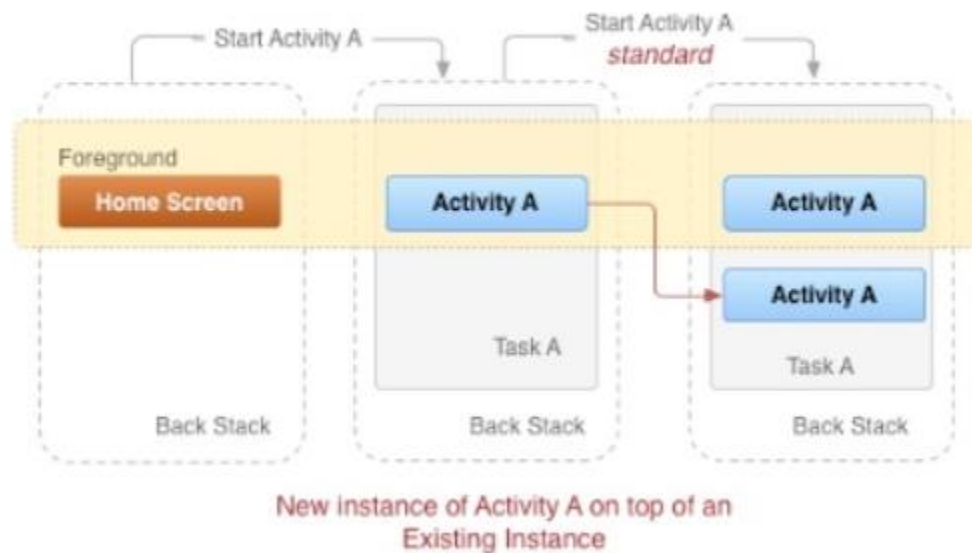
- Para modificar el comportamiento de la actividad en el manifiesto, se fija el atributo **launchMode**.
- Hay cuatro diferentes "modos de lanzamiento" para establecer el comportamiento de la actividad en la pila :
 - **standard**: El comportamiento predeterminado. Crea una nueva instancia de la actividad en la tarea.
 - **singleTop**: Reutilizar una instancia de actividad si ya está en la parte superior de la pila; de lo contrario crear nueva instancia.
 - **singleTask**: Reutilizar una instancia de actividad si existe en una pila ya existente; de lo contrario crear una nueva pila de tareas.
 - **singleInstance**: Igual que singleTask pero la actividad siempre es el único miembro de su tarea; cualquier actividad iniciada por esta se abre en una tarea aparte.

standard



```
<activity android:launchMode="standard"/>
```

- Default Launch Mode
- All the Activities belong to the same task in the application

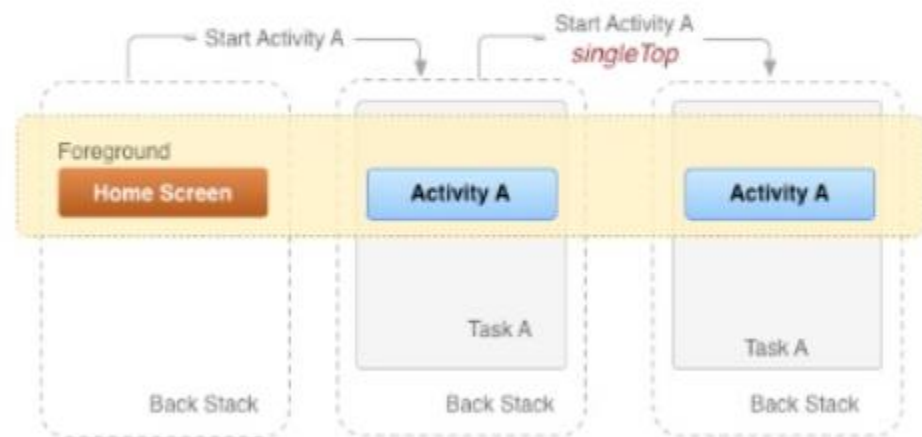


singleTop



```
<activity  
    android:launchMode="singleTop" />
```

- Single instance of an Activity can exist at the top of the back stack
- Multiple instances can exist if there are activities between them



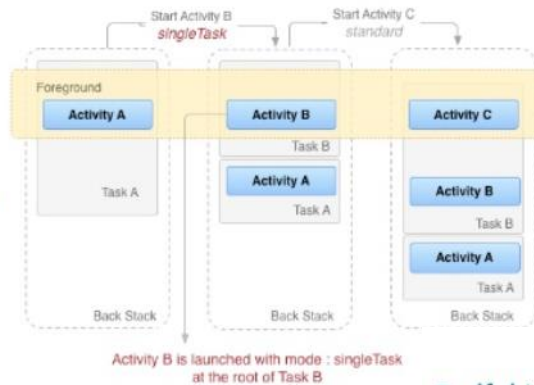
New instance of Activity A not created
same instance re-used

singleTask

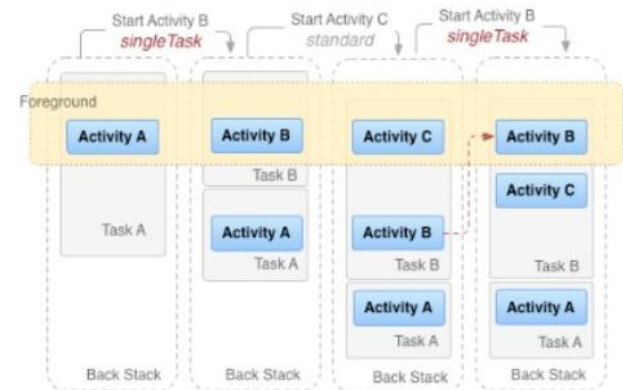


```
<activity android:launchMode="singleTask" />
```

- Activity is the root of a task
- Other activities are part of this task if they are launched from this activity
- If this activity already exists Intent is routed to that Instance



- If this activity already exists Intent is routed to that instance and `onNewIntent()` Lifecycle method is called

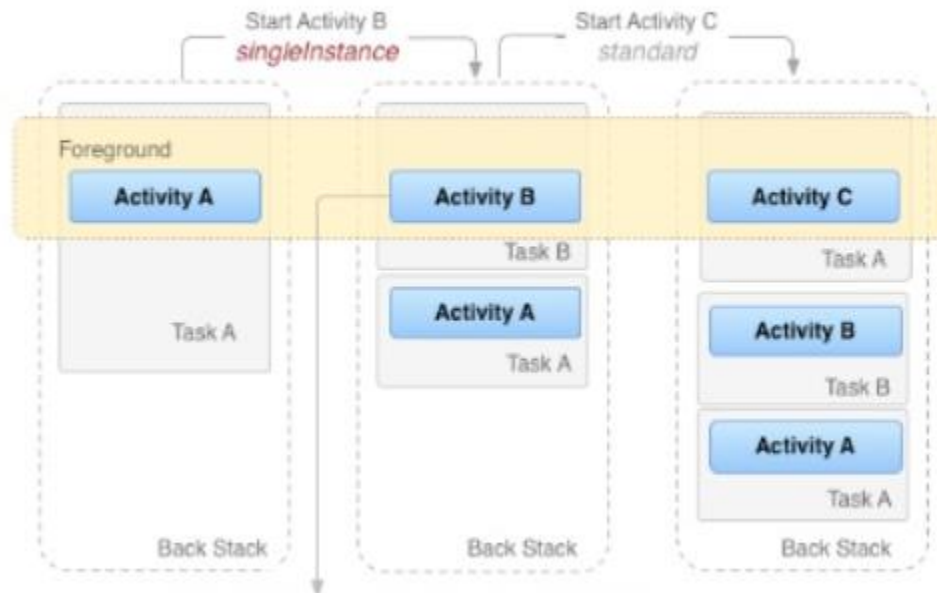


singleInstance



```
<activity  
    android:launchMode="singleInstance" />
```

- Activity B is launched with launchMode singleInstance, it is always the only activity of its task



Activity B is launched with mode : singleInstance at the root of Task B. This will be the only activity in the task.

Definir modo de inicio en Intent



- Algunos de los flags que se incluyen en el Intent y que se pueden utilizar para modificar el comportamiento por defecto son:
 - **FLAG_ACTIVITY_NEW_TASK**: consigue el mismo comportamiento que el valor "singleTask" en atrib. launchMode.
 - **FLAG_ACTIVITY_SINGLE_TOP**: consigue el mismo comportamiento que el valor "singleTop".
 - **FLAG_ACTIVITY_CLEAR_TOP**: Si la actividad que se inicia ya se está ejecutando en la tarea en curso, en vez de iniciar una nueva instancia de la actividad, todas las actividades encima suya son destruidas y se reanuda la actividad instanciada. No hay un valor del atributo launchMode que consigue este comportamiento.
 - **FLAG_ACTIVITY_CLEAR_TOP**: Se utiliza normalmente junto con **FLAG_ACTIVITY_NEW_TASK**. Cuando se usan juntos, estos flags son una manera de colocar una actividad ya existente dentro de otra tarea en una posición donde puede responder al intent.
- Ejemplo:

```
Intent intentA=new Intent (this,ActivityA.class);
intentA.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
startActivity(intentA);
```

Limpiar el back stack



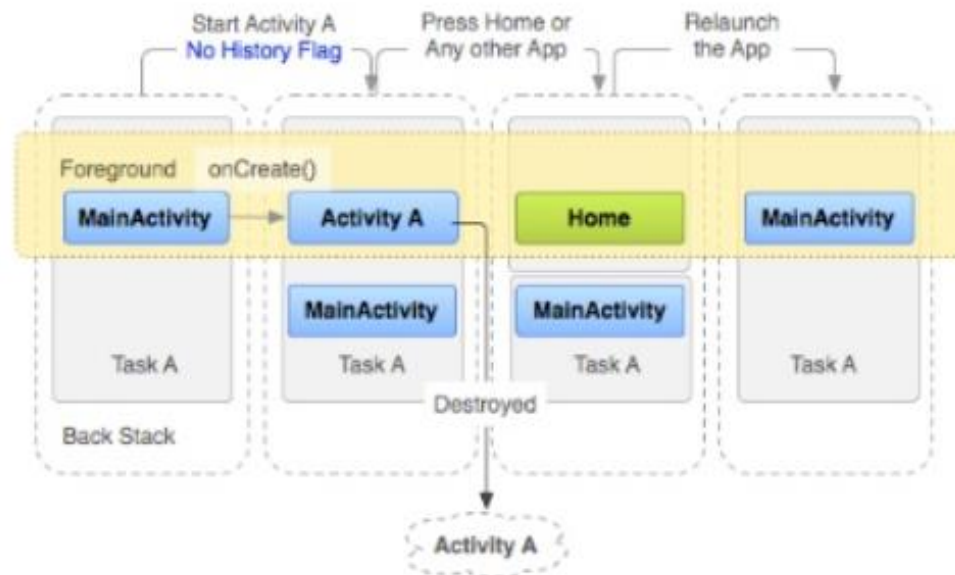
- Si el usuario abandona una tarea por mucho tiempo, el sistema limpia todas las actividades de la tarea, excepto la actividad principal. Cuando el usuario retorna a la tarea, solo se restaura la actividad principal. El sistema se comporta de esta manera, porque después de un tiempo determinado, los usuarios normalmente han abandonado lo que estaban haciendo y vuelven a la tarea para iniciar algo nuevo.
- Existen algunos atributos de las actividades que se pueden utilizar para modificar este comportamiento:
 - **alwaysRetainTaskState**: Si en la actividad principal de la tarea este atributo está puesto a "true" , el comportamiento por defecto descrito no ocurre. La tarea mantiene todas las actividades en su pila por un largo periodo de tiempo.
 - **clearTaskOnLaunch**: Si en la actividad principal de la tarea este atributo está puesto a "true", la pila elimina todo excepto la actividad principal cada vez que el usuario abandona la tarea y retorna a ella. El usuario siempre retorna a la tarea en su estado inicial, aunque haya abandonado la tarea sólo por un instante.
 - **finishOnTaskLaunch**: Este atributo es como clearTaskOnLaunch, pero trabaja en una sola actividad, no en una tarea completa.

Actividad no histórica



- Si una actividad no se debe añadir a la pila de tareas, se usa el atributo noHistory o el flag FLAG_ACTIVITY_NO_HISTORY:

- An Activity can be set to have no history on the back stack - it will be destroyed as soon as user moves away from it using the the intent flag **FLAG_ACTIVITY_NO_HISTORY**

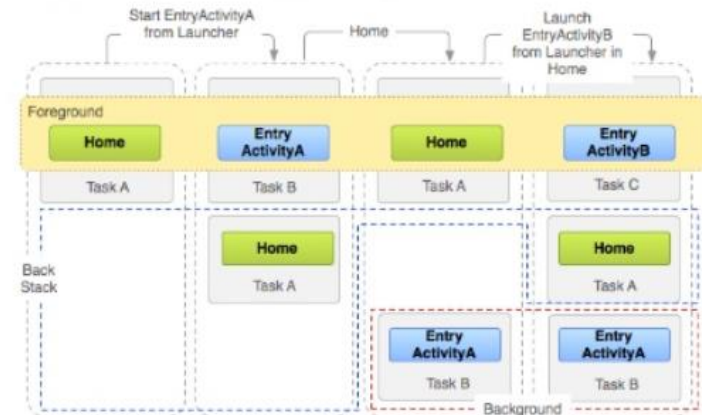


Gestión de afinidades



- La afinidad indica la tarea a la que una actividad prefiere pertenecer.
- Por defecto, todas las actividades de la misma aplicación prefieren estar en la misma tarea.
- Sin embargo, se puede modificar esta afinidad por defecto de una actividad.
- Las actividades definidas en aplicaciones diferentes pueden compartir una afinidad o las actividades definidas en la misma aplicación pueden tener asignadas diferentes afinidades de tareas.
- Se pueden modificar las afinidades para cualquier actividad usando el atributo taskAffinity de <activity>.
- [Información](#)

- TaskAffinity can be used to launch activities in a new task, e.g having two Launcher Activities which need to have their own task
- EntryActivityA and EntryActivityB will Launch in their own tasks



- EntryActivityA and EntryActivityB will Launch in their own tasks
- Manifest file entries for the two activities

```
<activity android:name=".EntryActivityA"
    android:label="@string/activitya"
    android:taskAffinity="stacksample.activitya">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".EntryActivityB"
    android:label="@string/activityb"
    android:taskAffinity="stacksample.activityb">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Iniciar una tarea



- Se puede configurar una actividad como el punto de entrada para una tarea dándole un *intent filter* con "android.intent.action.MAIN" como la acción específica y "android.intent.category.LAUNCHER" como la categoría especificada.
- Por ejemplo:

<activity ... >

<intent-filter ... >

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

...

</activity>

- Un *intent filter* de este tipo hace que se muestren un icono y una etiqueta para la actividad en el *Launcher* (HOME), dando al usuario una manera de iniciar la actividad y de volver a la tarea que se crea en cualquier momento.
- Esta segunda habilidad es importante: los usuarios deben poder abandonar una tarea y volver utilizando este iniciador de la actividad. Por esta razón, los modos de inicio "singleTask" y "singleInstance", deberían ser utilizadas sólo cuando la actividad tenga un ACTION_MAIN y un filtro CATEGORY_LAUNCHER. ¿Qué pasaría por ejemplo, si falta el filtro? Un intent lanza una actividad "singleTask" iniciando una nueva tarea, y el usuario gasta algún tiempo trabajando en esa tarea. El usuario hace click en HOME. La tarea se manda a un segundo plano y no es visible. Ya que no se representa en el *Launcher*, el usuario no tiene forma de volver a la tarea.

Prácticas propuestas



- Realiza la hoja de ejercicios
Ejercicios_07_Ciclo_de_vida_actividad