

# SALVAR EL ESTADO DE UNA ACTIVITY

# Salvar el estado de una Activity (I)

- ▶ Cuando una **activity** es **pausada** (*onPause*) o **detenida** (*onStop*), el **estado** de la actividad es **retenido AUTOMÁTICAMENTE**.
- ▶ Esto sucede porque **la Activity aún sigue en memoria**: *su estado, sus miembros*. Está **viva**.
- ▶ Por tanto, cuando la **activity REARRANQUE** (*onResume*), **seguirá como el usuario la dejó** (*salvo que sea destruida antes*).
- ▶ Sin embargo, **cuando el sistema destruye una activity** para recuperar memoria, **esto no sucede** como se explicó arriba.
- ▶ El **objeto Activity es destruido** y, por tanto, su **estado no es mantenido**.

# Salvar el estado de una Activity: `onSaveInstanceState()` (II)

- ▶ En caso de **destrucción de la actividad**, el estado se pierde **en principio** (el objeto Activity es realmente destruido).
- ▶ En ese caso, **si el usuario vuelve**, es necesario crear de **nuevo la actividad**.
- ▶ **El usuario no tiene ni idea de que la actividad ha sido destruida**, por lo que esperará que todo siga como lo dejó.
- ▶ En esta situación de destrucción de la actividad, tenemos la oportunidad de **salvar información importante sobre el estado de la actividad** implementando un **método callback adicional**.
- ▶ El método es **`onSaveInstanceState()`**.

# Salvar el estado de una Activity: `onSaveInstanceState()` (III)

- ▶ El sistema llama siempre a este callback justo antes de hacer que una activity sea vulnerable a ser destruida.
- ▶ El sistema pasa a este método un “**Bundle**” en el cual se puede **guardar información** relevante sobre el estado de la activity.
- ▶ Un **Bundle** (paquete o fardo) **es un tipo de almacén genérico de datos**, que **funciona de una forma similar a un diccionario o MAP**, con **pares nombre-valor** o **clave-valor**.
- ▶ **Todos los datos que necesitemos guardar** (referentes al *estado de la actividad, interfaz de usuario*, etc) **se pueden meter en este Bundle**.

# Salvar el estado de una Activity:

## Bundle (IV)

- ▶ Un **Bundle** proporciona *métodos* tipo: **putString()**, **putInt()**, ... para guardar los datos.
- ▶ **onSaveInstanceState()** como es un **método callback**, cumple la **norma** de:
  - 👍 **llamar SIEMPRE al método correspondiente de la superclase “super” en la primera línea de código.**
- ▶ El **método por defecto**, **onSaveInstanceState()** de la superclase **Activity**, **se hace cargo de la mayoría de la interfaz de usuario por nosotros.**

# Salvar el estado de una Activity: `onSaveInstanceState()` (V)

- ▶ En particular, la **implementación por defecto llama al método `onSaveInstanceState()` por cada View del layout**, lo que **permite a cada vista suministrar información de si misma que debería ser guardada**.
  - ▶ Así, los EditText se guarda el texto que tengan, en un checkBox se guarda si está marcado o no,...
- ▶ Para que esto suceda **cada widget : `EditText`, `Checkbox`,... debe tener un ID único**.
- ▶ Si un **widget no tiene un ID**, el **sistema no guarda** su estado.
- ▶ A pesar de que la implementación por defecto del método `onSaveInstanceState()` guarda información útil sobre la actividad del IU, **se puede necesitar sobrescribirlo para guardar información adicional como valores calculados,...**

# Salvar el estado de una Activity: `onSaveInstanceState()` (VI)

- ▶ Dado que la **implementación por defecto** de `onSaveInstanceState()` ayuda a guardar el estado de la IU, **si se sobrescribe**:
  - ▶ se debería llamar siempre a la implementación de la superclase del método `onSaveInstanceState()` antes de nada **para evitar tener que guardar nosotros toda la interfaz de usuario**.



# Notas sobre onSaveInstanceState() (I)

- ▶ **NO HAY GARANTÍA** de que `onSaveInstanceState()` sea llamado antes de que la activity sea destruida.
- ▶ Hay casos en los que no es necesario guardar el estado:
  - ◉ Como **cuando el usuario deja la activity** pulsando el **botón BACK**.
  - ◉ O cuando el **usuario cierra la activity explícitamente**.



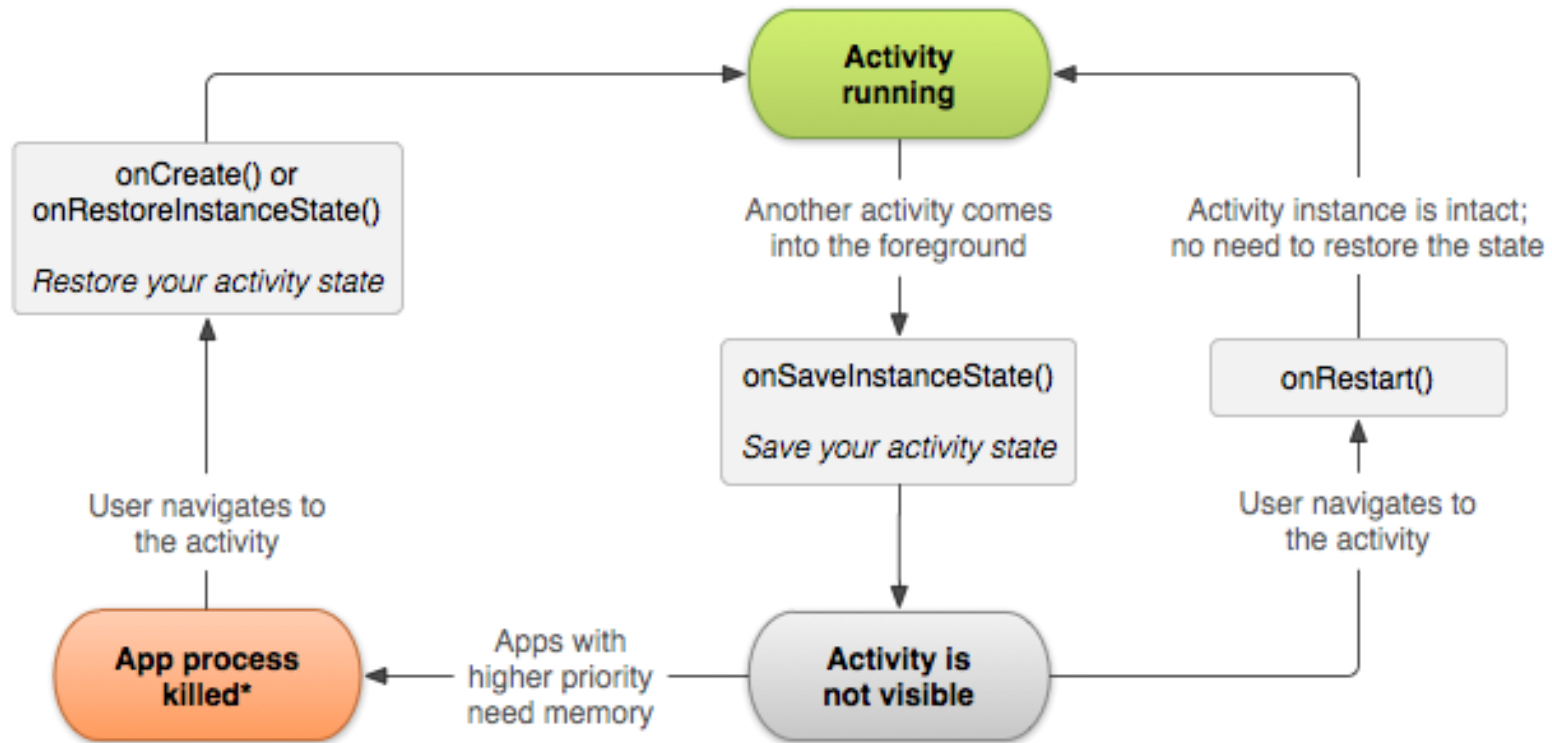
## Notas sobre onSaveInstanceState() (II)

- ▶ Si el **sistema llama** a onSaveInstanceState() **SIEMPRE** lo hace **antes de onStop()** y **posiblemente antes de onPause()**.
- ▶ Es **incorrecto** creer que este método es **llamado siempre antes de cada llamada a onPause o onStop**.
- ▶ En general, **sólo se llamará si “peligra” el estado de la interfaz de usuario**.

# Notas sobre onSaveInstanceState() (III)

- ▶ Dado que **no se garantiza que se llame a onSaveInstanceState()**, se debería **utilizar solo para guardar el estado del IU**.
- ▶ **No se debe utilizar para almacenar datos de forma persistente.**
- ▶ En su lugar, se debería **utilizar el método onPause()** para **almacenar datos de forma persistente** (guardar en una base de datos).
- ▶ Una buena **manera para probar la habilidad de la aplicación para restaurar el estado** es **rotar el dispositivo para que la orientación de la pantalla cambie**.
- ▶ Cuando la orientación de la pantalla cambia, el sistema destruye y vuelve a crear la actividad para aplicar recursos alternativos que estén disponibles para la nueva orientación.

# Notas sobre onSaveInstanceState() (IV)



\*Activity instance is destroyed, but the state from onSaveInstanceState() is saved

Las dos maneras en las que una actividad devuelve el focus al usuario con su estado intacto son cuando la actividad está parada, reanudada y el estado de la actividad se mantiene intacto (**rama derecha**), o cuando la actividad es destruida, creada de nuevo y la actividad ha de restaurar el estado previo (**rama izquierda**).

## EJEMPLO DE USO onSaveInstanceState()



```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

# RESTAURAR EL ESTADO DE UNA ACTIVITY

# Restaurar el estado de una Activity (I)

- ▶ Si el **sistema mata el proceso** de nuestra aplicación y el usuario navega de nuevo con el **botón VOLVER** a la actividad, **el sistema debe volver a crear la actividad y dejarla tal como estaba antes de su destrucción.**
- ▶ Para eso, **el sistema pasa el Bundle guardado en onSaveInstanceState()** de nuevo a la actividad cuando la reanuda.
- ▶ **Ese Bundle se le pasa a dos métodos:**
  -  **onCreate()**
  -  **onRestoreInstanceState()**
- ▶ Usando **cualquiera de esos métodos**, **se puede extraer el estado guardado del Bundle** y restaurar la actividad a su estado.
- ▶ **Si no hay información que restaurar** (*por ejemplo cuando se crea la actividad por primera vez*) **el Bundle será null.**

# Restaurar el estado de una Activity (II)

## ► USO DE onCreate():

- Se debe chequear si Bundle es NULL antes de intentar leer su valor, porque onCreate también se ejecuta al crear una nueva instancia de una Activity y en este caso sería NULL.

```
@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 of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
    ...
}
```



# Restaurar el estado de una Activity (III)

## ► USO DE `onRestoreInstanceState()`:

- El método `onRestoreInstanceState()` es llamado después de `onStart()` .
- Este método sólo se llama SI HAY UN ESTADO PARA RECUPERAR
  - Así que, no necesitamos chequear si BUNDLE NO ES NULL. NUNCA SERÁ NULL en este caso.
- `onRestoreInstanceState` tiene una implementación por defecto (en la superclase) que se hace cargo automáticamente de la información guardada por la implementación por defecto en `onSaveInstanceState` y la restaura.
- 🔗 Por tanto, debemos llamar a su implementación en super y pasarle el Bundle.

## Restaurar el estado de una Activity (III)

- ▶ Por ejemplo, si creamos una **vista** sobre la marcha y esta **no tiene un ID**, **habrá que guardar su estado “a mano”**. Lo mismo ocurre con los miembros de la actividad y cualquier otra información.

# CAMBIOS DE CONFIGURACIÓN DURANTE LA EJECUCIÓN

# Cambios de configuración durante la ejecución (I)

- ▶ Algunas **configuraciones de dispositivo** pueden **cambiar durante la ejecución** (*orientacion de la pantalla, disponibilidad del teclado, idioma...*)
- ▶ Si esto ocurre, **Android destruye e inmediatamente crea de nuevo la actividad** que está corriendo (**onDestroy** seguido de **onCreate**)
- ▶ Con esto, se **carga** la aplicación con los **nuevos recursos para esta configuración** (*por ejemplo, un layout para formato horizontal*).

# Cambios de configuración durante la ejecución (II)

- ▶ Para lidiar con esos cambios, la mejor estrategia es implementar correctamente `onRestoreInstanceState` y `onSaveInstanceState` (también podría ser `onCreate`).
- ▶ Pero a veces tenemos **muchos datos que guardar**:
  - Datos que estaban descargando de Internet o ya se habían descargado
  - En estos casos es **mejor usar un Fragment** donde se guardan esos datos y se restaura el fragment.
  - Ver cómo se haría aquí: Handling Runtime Changes.