

Activity link Continued..

Activity state and ejection from memory

- The system kills processes when it needs to free up RAM; the likelihood of the system killing a given process depends on the state of the process at the time. Process state, in turn, depends on the state of the activity running in the process.

Table 1 shows the correlation among process state, activity state, and likelihood of the system's killing the process.

Likelihood of being killed	Process state	Activity state
Least	Foreground (having or about to get focus)	Created Started Resumed
More	Background (lost focus)	Paused
Most	Background (not visible)	Stopped
	Empty	Destroyed

Table 1. Relationship between process lifecycle and activity state

- The system never kills an activity directly to free up memory. Instead, it kills the process in which the activity runs, destroying not only the activity but everything else running in the process, as well.
- A user can also kill a process by using the Application Manager under Settings to kill the corresponding app.

Saving and restoring transient UI state

- A user expects an activity's UI state to remain the same throughout a configuration change, such as rotation or switching into multi-window mode.
- The system destroys the activity by default when such a configuration change occurs, wiping away any UI state stored in the activity instance. Similarly, a user expects UI state to remain the same if they temporarily switch away from your app to a different app and then come back to your app later. However, the system may destroy your application's process while the user is away and your activity is stopped.
- When the activity is destroyed due to system constraints, you should preserve the user's transient UI state using a combination of ViewModel, onSaveInstanceState(), and/or local storage.

- This section outlines what instance state is and how to implement the `onSaveInstanceState()` method, which is a callback on the activity itself.
- If your UI data is simple and lightweight, such as a primitive data type or a simple object (like `String`), you can use `onSaveInstanceState()` alone to persist the UI state across both configuration changes and system-initiated process death. In most cases, though, you should use both `ViewModel` and `onSaveInstanceState()` (as outlined in `Saving UI State`) since `onSaveInstanceState()` incurs serialization/deserialization costs.

Instance state

- There are a few scenarios in which your activity is destroyed due to normal app behaviour, such as when the user presses the Back button or your activity signals its own destruction by calling the `finish()` method.
- When your activity is destroyed because the user presses Back or the activity finishes itself, both the system's and the user's concept of that Activity instance is gone forever. In these scenarios, the user's expectation matches the system's behaviour and you do not have any extra work to do.
- If the system destroys the activity due to system constraints (such as a configuration change or memory pressure), then although the actual Activity instance is gone, the system remembers that it existed.
- If the user attempts to navigate back to the activity, the system creates a new instance of that activity using a set of saved data that describes the state of the activity when it was destroyed.
- The saved data that the system uses to restore the previous state is called the instance state and is a collection of key-value pairs stored in a Bundle object.
- By default, the system uses the Bundle instance state to save information about each View object in your activity layout (such as the text value entered into an `EditText` widget). So, if your activity instance is destroyed and recreated, the state of the layout is restored to its previous state with no code required by you.
- However, your activity might have more state information that you'd like to restore, such as member variables that track the user's progress in the activity.
- Note: In order for the Android system to restore the state of the views in your activity, each view must have a unique ID, supplied by the `android:id` attribute.
- A Bundle object isn't appropriate for preserving more than a trivial amount of data because it requires serialization on the main thread and consumes system-process memory.
- To preserve more than a very small amount of data, you should take a combined approach to preserving data, using persistent local storage, the `onSaveInstanceState()` method, and the `ViewModel` class, as outlined in `Saving UI States`.
- Save simple, lightweight UI state using `onSaveInstanceState()`
- As your activity begins to stop, the system calls the `onSaveInstanceState()` method so your activity can save state information to an instance state bundle.
- The default implementation of this method saves transient information about the state of the activity's view hierarchy, such as the text in an `EditText` widget or the scroll position of a `ListView` widget.

- To save additional instance state information for your activity, you must override `onSaveInstanceState()` and add key-value pairs to the `Bundle` object that is saved in the event that your activity is destroyed unexpectedly.
- If you override `onSaveInstanceState()`, you must call the superclass implementation if you want the default implementation to save the state of the view hierarchy. For example:

JAVA

```
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, currentScore);
    savedInstanceState.putInt(STATE_LEVEL, currentLevel);

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

Note: `onSaveInstanceState()` is not called when the user explicitly closes the activity or in other cases when `finish()` is called.

To save persistent data, such as user preferences or data for a database, you should take appropriate opportunities when your activity is in the foreground. If no such opportunity arises, you should save such data during the `onStop()` method.

Restore activity UI state using saved instance state

When your activity is recreated after it was previously destroyed, you can recover your saved instance state from the `Bundle` that the system passes to your activity. Both the `onCreate()` and `onRestoreInstanceState()` callback methods receive the same `Bundle` that contains the instance state information.

Because the `onCreate()` method is called whether the system is creating a new instance of your activity or recreating a previous one, you must check whether the state `Bundle` is null before you attempt to read it. If it is null, then the system is creating a new instance of the activity, instead of restoring a previous one that was destroyed.

For example, the following code snippet shows how you can restore some state data in `onCreate()`:

```
@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  
    currentScore = savedInstanceState.getInt(STATE_SCORE);  
    currentLevel = savedInstanceState.getInt(STATE_LEVEL);  
} else {  
    // Probably initialize members with default values for a new instance  
}  
// ...  
}
```

Instead of restoring the state during `onCreate()` you may choose to implement `onRestoreInstanceState()`, which the system calls after the `onStart()` method. The system calls `onRestoreInstanceState()` only if there is a saved state to restore, so you do not need to check whether the `Bundle` is null:

```
public void onRestoreInstanceState(Bundle savedInstanceState) {  
    // Always call the superclass so it can restore the view hierarchy  
    super.onRestoreInstanceState(savedInstanceState);  
  
    // Restore state members from saved instance  
    currentScore = savedInstanceState.getInt(STATE_SCORE);  
    currentLevel = savedInstanceState.getInt(STATE_LEVEL);  
}
```

For any doubt contact on whatsapp no. 9873961590