Recreating an Activity

There are a few scenarios in which your activity is destroyed due to normal app behavior, such as when the user presses the *Back* button or your activity signals its own destruction by calling `finish()`. The system may also destroy your activity if it's currently stopped and hasn't been used in a long time or the foreground activity requires more resources so the system must shut down background processes to recover memory.

When your activity is destroyed because the user presses *Back* or the activity finishes itself, the system's concept of that `Activity` instance is gone forever because the behavior indicates the activity is no longer needed. However, if the system destroys the activity due to system constraints (rather than normal app behavior), then although the actual `Activity` instance is gone, the system remembers that it existed such that if the user navigates back to it, the system creates a new instance of the 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.

Caution: Your activity will be destroyed and recreated each time the user rotates the screen. When the screen changes orientation, the system destroys and recreates the foreground activity because the screen configuration has changed and your activity might need to load alternative resources (such as the layout).

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` object). 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.

To save additional data about the activity state, you must override the `onSaveInstanceState()` callback method. The system calls this method when the user is leaving your activity and passes it the `Bundle` object that will be saved in the event that your activity is destroyed unexpectedly. If the system must recreate the activity instance later, it passes the same `Bundle` object to both the `onRestoreInstanceState()` and `onCreate()` methods.

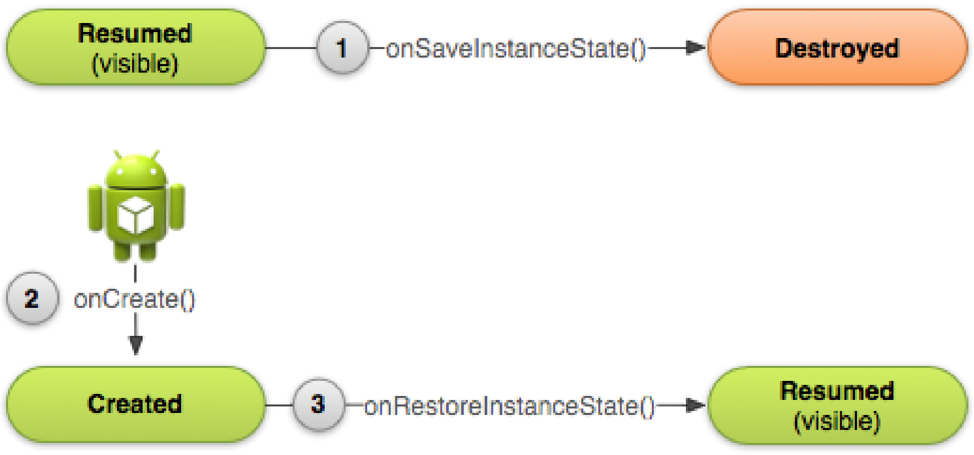

Figure 2. As the system begins to stop your activity, it calls `onSaveInstanceState()` (1) so you can specify additional state data you'd like to save in case the `Activity` instance must be recreated. If the activity is destroyed and the same instance must be recreated, the system passes the state data defined at (1) to both the `onCreate()` method (2) and the `onRestoreInstanceState()` method (3).

Save Your Activity State

As your activity begins to stop, the system calls `onSaveInstanceState()` so your activity can save state information with a collection of key-value pairs. The default implementation of this method saves 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`.

To save additional state information for your activity, you must implement `onSaveInstanceState()` and add key-value pairs to the `Bundle` object. For example:

```
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);
}
```

Caution: Always call the superclass implementation of `onSaveInstanceState()` so the default implementation can save the state of the view hierarchy.

Restore Your Activity State

When your activity is recreated after it was previously destroyed, you can recover your saved state from the `Bundle` that the system passes 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, here's 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
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = 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
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```

Caution: Always call the superclass implementation of `onRestoreInstanceState()` so the default implementation can restore the state of the view hierarchy.

To learn more about recreating your activity due to a restart event at runtime (such as when the screen rotates), read Handling Runtime Changes.