1. **Activity Lifecycle - full**
2. **can we call onDestroy directly**

No, you cannot call `onDestroy()` directly in the same way you might call other methods. `onDestroy()` is a lifecycle callback method managed by the Android system and is called when the system is about to destroy the activity. If you need to trigger the destruction of an activity, you should call `finish()`, which will lead to `onDestroy()` being called by the system.

**Example**

```java

// Correct way to trigger activity destruction

finish();

```

After calling `finish()`, the system will follow the appropriate sequence of lifecycle callbacks:

1. `onPause()`

2. `onStop()`

3. `onDestroy()`

**Summary**

While you cannot directly call `onDestroy()`, you can use `finish()` to signal that you want to close the activity, which will result in `onDestroy()` being invoked by the Android system.

1. **what is async task**

\*\*AsyncTask\*\* in Android helps keep the UI responsive by running heavy tasks in the background.

**It consists of:**

- \*\*doInBackground()\*\*: Executes background code and can send results to the UI thread.

- \*\*onPreExecute()\*\*: Runs before background processing.

- \*\*onPostExecute()\*\*: Called after background processing completes.

- \*\*onProgressUpdate()\*\*: Receives progress updates from background tasks.

It uses three generic types:

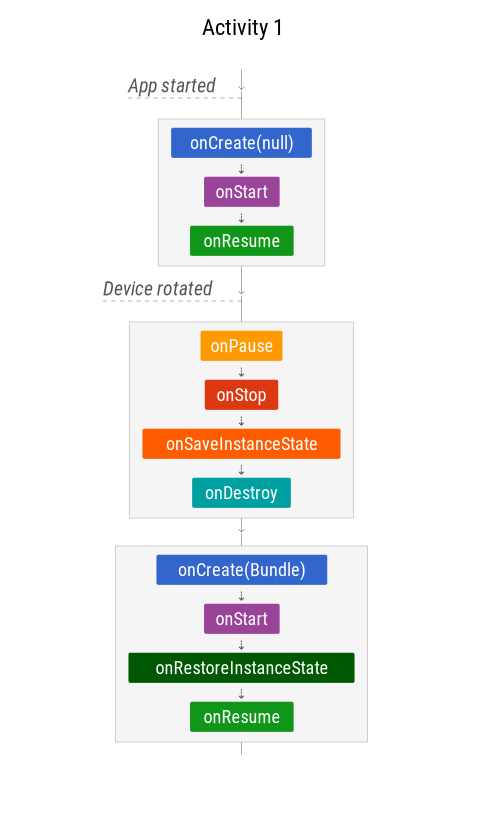
- \*\*Params\*\*: Input parameters for the task.

- \*\*Progress\*\*: Progress units during computation.

- \*\*Result\*\*: Result of the computation.

1. **BroadcastReciever**
2. **Threads**
3. **Threads vs Coroutines**
4. **Services**
5. **Context - types of Context**
6. **Application Class**
7. **How the lifecycle works when the screen is rotated**

<https://medium.com/@JoseAlcerreca/the-android-lifecycle-cheat-sheet-part-i-single-activities-e49fd3d202ab>



1. **which is good for network call retrofit or coroutines**

**Retrofit:**

- Established library for RESTful API consumption.

- Automatic JSON parsing using Gson or other converters.

- Requires more setup, especially for beginners.

- Best suited for Java-based projects due to its compatibility and robustness.

- Offers integration with RxJava for reactive programming, suitable for complex asynchronous scenarios.

**Coroutines:**

- Built-in support in Kotlin for asynchronous programming.

- Concise syntax with suspending functions simplifies code readability.

- Ideal for Kotlin projects, leveraging its simplicity and modern language features.

- Lightweight and efficient, although relatively newer compared to Retrofit, with a growing community and ecosystem support.

1. **Launch modes in Android**
2. **How to handle configuration change like screen rotation**
3. **\*\*Manual Handling\*\*:**

- Declare `android:configChanges` in manifest.

- Override `onConfigurationChanged` in activity.

1. **\*\*State Saving/Restoring\*\*:**

- Use `onSaveInstanceState` to save data.

- Use `onCreate` or `onRestoreInstanceState` to restore data.

1. **\*\*ViewModel and LiveData\*\*:**

- Use `ViewModel` to manage UI-related data.

- Observe changes in `LiveData` to update UI.

These methods ensure smooth handling of configuration changes such as screen rotation in Android apps.

1. explain parcelable