**Separation of concerns**

The most important principle to follow is separation of concerns. It's a common mistake to write all your code in an Activity or a Fragment. These UI-based classes should only contain logic that handles UI and operating system interactions. By keeping these classes as lean as possible, you can avoid many lifecycle-related problems.

Keep in mind that you don't own implementations of Activity and Fragment; rather, these are just glue classes that represent the contract between the Android OS and your app. The OS can destroy them at any time based on user interactions or because of system conditions like low memory. To provide a satisfactory user experience and a more manageable app maintenance experience, it's best to minimize your dependency on them

**Drive UI from a model**

Another important principle is that you should drive your UI from a model, preferably a persistent model. Models are components that are responsible for handling the data for an app. They're independent from the View objects and app components in your app, so they're unaffected by the app's lifecycle and the associated concerns.

Persistence is ideal for the following reasons:

* Your users don't lose data if the Android OS destroys your app to free up resources.
* Your app continues to work in cases when a network connection is flaky or not available.

By basing your app on model classes with the well-defined responsibility of managing the data, your app is more testable and consistent.