Android - Mock Technical Interview Questions

🟢 BEGINNER 🟢

What is an Android activity?

An Android activity is a screen that the user can interact with. It is a single, focused task that the user can perform within an app, and it is a building block of an Android app. Activities are launched by the system when the user opens an app, and the user can navigate to different activities within the app to perform different tasks.

What is an Android Intent?

An Android intent is an abstract description of an operation to be performed. It is used to start an activity, launch a service, send a broadcast, or deliver a message between two apps. Intents are a fundamental part of the Android application framework and provide a way for apps to communicate with each other and with the system. There are two types of intents:

1. An **explicit intent** in Android is an intent that specifies the exact component to be called, by name. For example, if an app has a settings activity, the app can create an explicit intent that specifies the name of the settings activity and use that intent to start the activity.

```
// Starts a new activity from the current activity
val i = Intent(this, SecondActivity::class.java)
startActivity(i)
```

2. An **implicit intent** in Android is an intent that does not specify a specific component to be called. Instead, it declares the action that should be performed and lets the system decide which component should be called to handle the intent. For example, an app can create an implicit intent that describes the action of taking a picture, and the system will determine which app's camera activity should be called to handle the intent.

```
// Open the system's built-in camera
val i = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
startActivity(i)
```

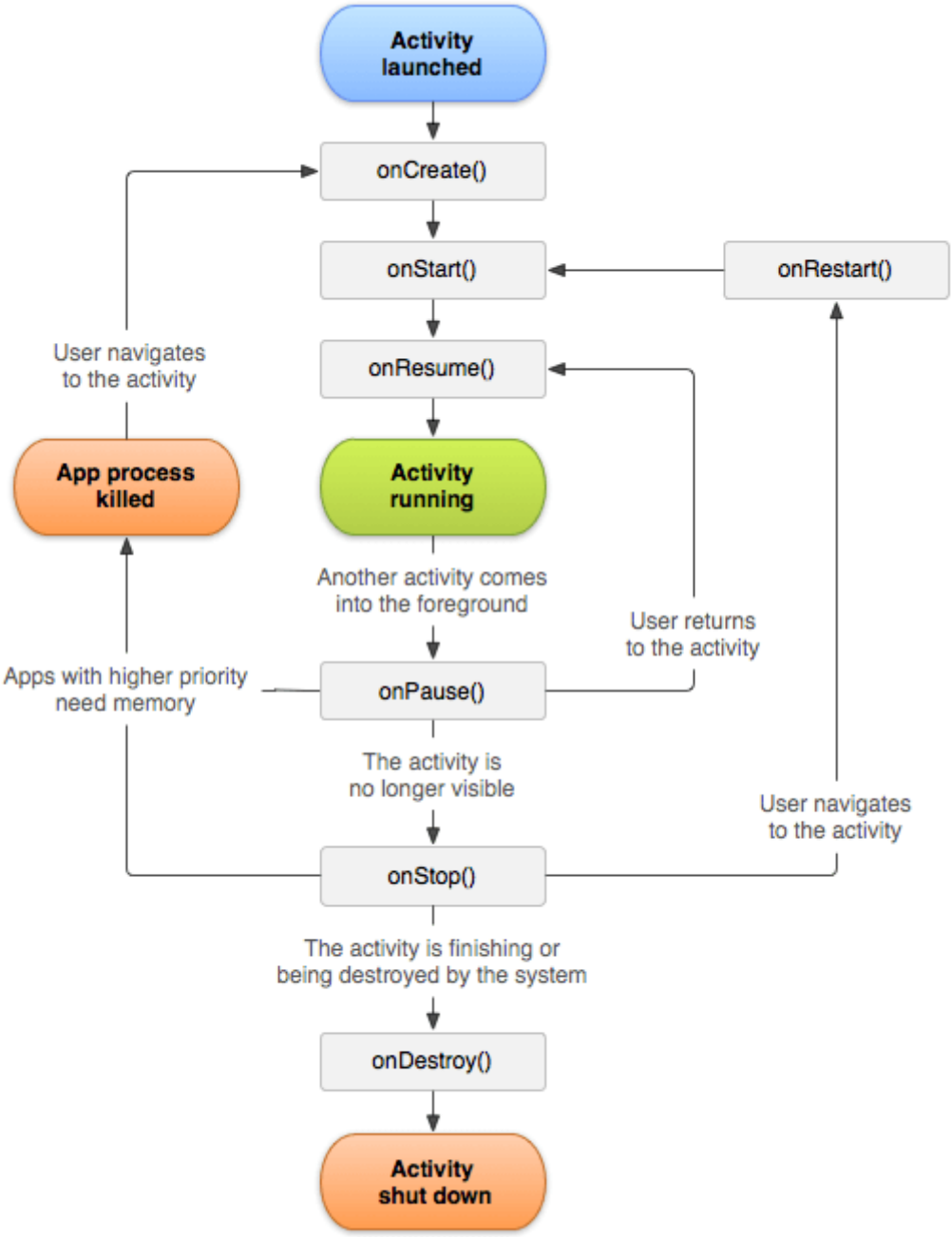
What is the Android manifest file, and what is its purpose?

The Android manifest file is an XML file that provides essential information about an Android app. It is included in the root directory of the app's project and contains information such as the app's **package name**, the **minimum required version** of the Android operating system, and the app's **declared components** (such as activities, services, and broadcast receivers).

The manifest file plays a crucial role in the app development process, as it **serves as the "glue"** that holds the app together. It is used by the Android system to determine which app to launch when the user taps on an app icon, which activities to launch when the app receives an intent, and **which permissions the app has declared**.

🟡 INTERMEDIATE 🟡

What is the Activity lifecycle?



The Activity lifecycle is the set of states that an app can go through during its lifetime. These states include:

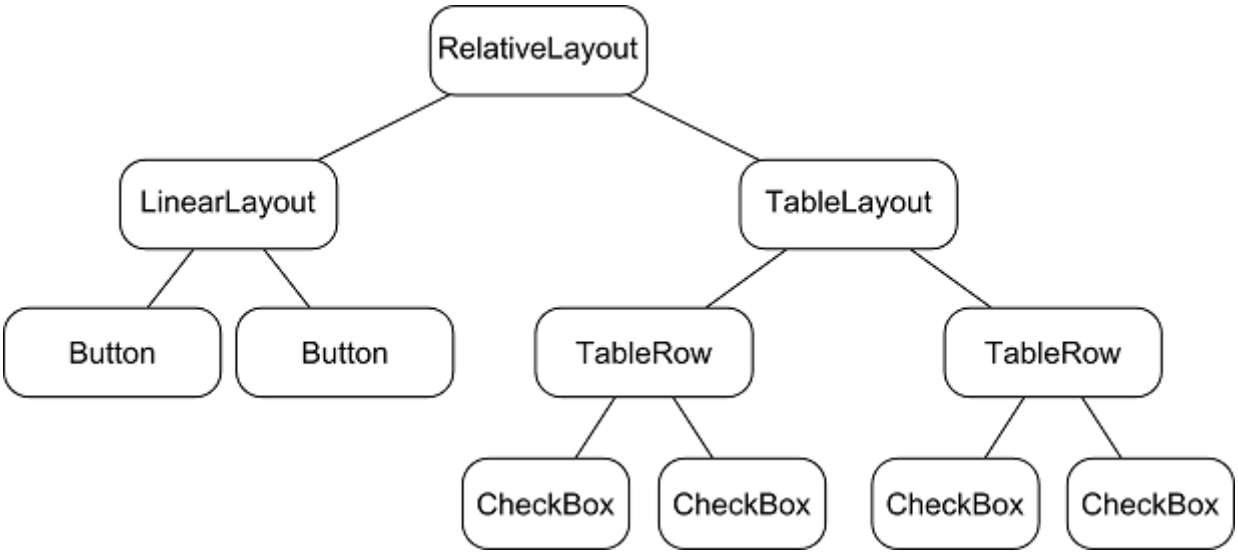
1. **Created:** When the app is first launched, it is in the created state.
2. **Started:** When the app becomes visible to the user, it is in the started state.
3. **Resumed:** When the app is in the foreground and the user is interacting with it, it is in the resumed state.
4. **Paused:** When the app is in the background but still visible to the user, it is in the paused state.
5. **Stopped:** When the app is no longer visible to the user, it is in the stopped state.

6. **Destroyed:** When the app is closed, it is in the destroyed state.

What is the Android View Hierarchy, and how does it work?

The Android View Hierarchy is the set of views that make up the user interface of an Android app. A view is a visual element on the screen, such as a button, a text field, or an image. The view hierarchy is organized as a tree, with the root node being the top-level view and the child nodes being the views contained within that view.

Ex.:



How do you handle user input and touch events?

In Android, user input and touch events can be handled by implementing the appropriate callback methods in the View class. For example, the onTouchEvent() method can be overridden to listen for touch events on a custom view, and the onKeyDown() method can be overridden to listen for key presses. The most common methods are **onClickListener** and **onLongClickListener** Additionally, Android provides the InputMethodManager class, which can be used to handle user input from on-screen keyboards and other input methods.

How do you handle data persistence in Android, using SQLite databases or Shared Preferences?

To handle data persistence in Android, you can use SQLite databases or Shared Preferences.

- **SQLite** is a database system built into Android that allows you to create and manipulate databases within your app.
- **Shared Preferences** is a simpler key-value storage system that lets you store small amounts of data in your app.

The right solution will depend on the specific needs of your app. SQLite is more powerful but also more complex, while Shared Preferences is simpler but less capable.

What is an Android fragment, and when should it be used?

An Android fragment is a modular section of an activity, with its **own lifecycle** and input events. A fragment is like a "**sub activity**" that can be added or removed from an activity during runtime. This allows the app to adapt its user interface to the available screen space and to provide a more flexible and dynamic user experience.

They should be used when you want to **modularize** your app's code and when you want to provide a **dynamic and responsive** user experience on tablets and other large-screen devices.

What is Context?

The context in Android refers to the current state of your app or object. The context provides services such as granting access to databases and preferences, resolving resources, and much more. In a context, there are two types:

- **Activity Context:** It is connected to the lifecycle of an activity. Use this method when passing the context within the scope of activity or when requesting a context attached to the current context.
- **Application Context:** This context is connected to the lifecycle of an application. You can use this when you need a context whose lifecycle is separate from the current context or when you are passing a context outside of the scope of the current activity.

🔴 ADVANCED 🔴

What is the difference between lazy initialization and lateinit?

Lazy initialization is a technique for delaying the creation of an object until it is needed. Lateinit is a Kotlin keyword that is used to mark a non-null property that is not yet initialized. Lazy initialization creates a new instance of the property when it is accessed, while lateinit simply checks that the property has been initialized before allowing access to it.

What is ViewModel?

The `ViewModel` class is a useful and powerful tool for preparing and managing the data for an `Activity` or `Fragment`. It allows you to separate the data management logic from the UI logic, which can help improve the performance and maintainability of your app. The ViewModel also provides a way to survive configuration changes, which can help provide a better user experience.

What is view binding?

View binding is a feature in Android that allows you to more easily access views in your layout without having to use `findViewById()` to look them up. View binding generates a class for your layout file, which contains direct references to all the views in the layout. This makes it easier and faster to access the views in your code, and reduces the risk of null pointer exceptions if you try to access a view that doesn't exist.

How do you test and debug your Android app, using tools such as the Android Studio debugger or the Android Emulator?

To test and debug your Android app, you can use the Android Studio debugger and the Android Emulator. The debugger allows you to pause the app and inspect its state, while the emulator simulates different Android devices and provides tools for simulating device events. These tools can help you find and fix errors and improve the app's performance.

- The debugger allows you to set breakpoints in your code, which **pause the execution** of the app and allow you to inspect the app's state at that point. You can then use the debugger to step through the code, view and **modify variables**, and **evaluate expressions**.
- The debugger provides a console for **logging messages**, which can be useful for tracking down errors and understanding the app's behavior.
- You can use the emulator to test your app on different **device configurations**, such as **different screen sizes** and Android **versions**, without needing physical devices.
- The emulator provides a range of tools for **simulating device events**, such as touch input and sensor data, which can be useful for testing the app's user interface and behavior.