Contents

1. Rotation & Intents 2
1.1 App lifecycle on rotation? 2
1.2 Does rotation affect the input? 2
1.3 Deletion of the ‘android:id’ attribute? 2
1.4 Different way to influence such behaviour? 2
1.5 Android mechanisms for saving and restoring activity states? 2
2. Services 2
2.1 What are services, when should they be used. What is the difference to threads? 2
2.2 Steps for creating a service 3
2.3 States of a service 3
3. Fragments 4
3.1 What was the original intention for the introduction of Fragments? 4
3.2 How are Fragments used in modern Android development? 4
3.3 The term ‘Single Activity Architecture’ & Pros / Cons & Similarities to Web-Dev? 4
4. Touch Control 4
4.4 Main differences between Figure 1 and Figure 2 & Pros / Cons 4
Figure 1 4
Figure 2 4

1. Rotation & Intents

1.1 App lifecycle on rotation?

The activity will be reopened, thus terminated, and then reopened, as an activity could have different layouts for different orientations. By default, Android Activities will be reopened when the apps-configuration is modified.

1.2 Does rotation affect the input?

The value of the input field was not affected by the layout change as the field had already an id set.

1.3 Deletion of the 'android:id' attribute?

When the id attribute of the text input field is deleted, and the phone is rotated then any change to its input value is lost, and the default value is set.

1.4 Different way to influence such behaviour?

Generally speaking, one can set a custom behaviour for the app when the configuration changes. This can be achieved by overriding the "onConfigurationChanged()" method.

Additionally, the behaviour of the app can be modified for a specific configuration change. To modify such a behaviour the "Android:configChanges" attribute has to be added to activity in the AndroidManifest. The value set for this attribute will force the activity to handle this type of change on its own. Thus, the activity will not be restarted for that specific configuration change.

1.5 Android mechanisms for saving and restoring activity states?

Android offers the following mechanisms:

- onSaveInstanceState():
  This method is called before an activity is paused or destroyed and therefore the state of an activity can be saved, so that it can be retrieved later.

- ViewModel:
  This class is meant to store or manage UI related data in general, thus it is not terminated when the configuration is modified. Hence this call can be used to hold values inside of it.

- onRestoreInstanceState()
  This method is called whenever an activity is recreated, which creates an opportunity to reload the previously saved states of an activity.

2. Services

2.1 What are services, when should they be used. What is the difference to threads?

Generally speaking, both Services and Threads are designed to perform tasks in the Background, thus not in the Main Thread of the application.

The difference between those two is, that Services are designed to run in the background and perform tasks that may run for a longer period.

Threads on the other hand are designed to run parallel to the main thread, thus they are used when a user wants to perform multiple tasks at the same time or when the user wants to perform a task that may take longer.

2.2 Steps for creating a service

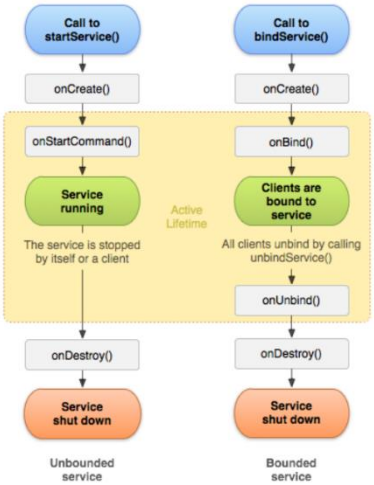
A service always has the following structure:

```
class AudioService : Service() {

    @Override
    override fun onBind(p0: Intent?): IBinder? {
        // Initialize the service
    }

    @Override
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
      // Implement the functionality of the service

        return START_STICKY;
    }

    @Override
    override fun onDestroy() {
        // Clean up when the service is stopped
    }
}
```

As the code snippet shows it has 3 methods that will be executed at different stages. In each stage code can be written that defines how the service will work at runtime.

2.3 States of a service

A service can reach different stages.



Difference between a bound service and a normal service, is that a bound service will only run as long as the other component is bound to it.

3. Fragments

3.1 What was the original intention for the introduction of Fragments?

The main goal was to provide a more flexible and modular option when developing an android application, thus the option to reuse such a component multiple times inside an activity. For instance, previously developers had to setup activities for different screen sizes or orientations, but nowadays the activity can remain the same, and the small components with which the activity is filled can vary.

3.2 How are Fragments used in modern Android development?

Well, they are used as a sort of Component type, which allows developers to create small custom reusable components, which can also be used dynamically.

3.3 The term 'Single Activity Architecture' & Pros / Cons & Similarities to Web-Dev?

The term "Single Activity Architecture" refers to a pattern that was used back in the day in which an application had only **one** main activity, but multiple fragments used inside of it, instead of multiple activities. Thus, the main activity was used as a container of fragments.

A big advantage of this concept is that the container remains the same, thus settings, orientation handling and more have to be set once, for the entire application instead of for each activity. Additionally, the overall code readability and reusability could be improved as well, as the code is generally speaking more reusable as it is grouped into smaller pieces.

On the other hand, cons could be, that the different pages / stages of the application may not be as clear as they would be when there were multiple activities. Thus, such a concept may not be as user friendly as the most common concept of having multiple activities, which each represent a page of the application.

In terms of similarities, modern frameworks in the area of Web development are based on the same concept, of having a container and dynamically inserting small components and therefore building the page with them. Popular frameworks are React, NextJS and others.

4. Touch Control

4.4 Main differences between Figure 1 and Figure 2 & Pros / Cons

Figure 1

An advantage could be that this layout has been setup using fragments, as the Button may be in a separate Fragment then the Surface View, thus it may be more flexible.

On the other hand, it does not look as good as the second layout, due to the fact the background colour of the SurfaceView does not match the background colour of the Fragment in which the button is located.

Figure 2

This layout may be more visually appealing as the background colours do match.

On the other hand, it does not seem as flexible as the first one as its modification in runtime may be more difficult due to the lack of fragments.