Interview Questions

ANDROID DEVELOPER

by

EDYODA

Q1. Explain the difference between **commit()** and **apply()** in SharedPreferences?

Q2. Can you run **bytecode** in Android?

Q3. Explain the **'build process'** in Android.

Q4. Does Java support **'multiple inheritance'**?

Q5. Can Android application only be written in **Java**?

Q6. When should you use a **FrameLayout**?

Q7. Difference between **Serializable and Parcelable**?

Q8. What is a **Pending Intent**?

Q9. Difference between **AsyncTasks & Threads**?

Q10. Why would you do the **setContentView() in onCreate()** of Activity class?

Q11. Scenario in which only **onDestroy()** is called for an activity without **onPause()** and **onStop()**?

Q12. What is **NDK**?

Q13. Which scenario can only be tested on real devices but not on emulator?

Q14. What is **bitmap pooling** in android?

Q15. What's the difference between **onCreate()** and **onStart()**?

Q16. How does the activity respond when the user rotates the screen?

Q17. How can you clear the back stack when creating a new Activity?

Q18. What is **singleton class** in Android?

Q19. Why is it recommended to use the **default constructor** to create a Fragment?

Q20. When should you use a fragment rather than an activity?

ANSWERS

Q1. Explain the difference between **commit()** and **apply()** in SharedPreferences?

A1. commit() writes the data synchronously and returns true for success and false for failure whereas, **apply()** is asynchronous and it won't return any value.

Q2. Can you run **bytecode** in Android?

A2. No, we cannot. Because android uses **Dalvik Virtual Machine (DVM)** which requires a special bytecode. The build tools convert Java class files into Dalvik Executable files using an Android tool called **"dx".**

Q3. Explain the **'build process'** in Android.

A3. Following are the steps followed by the build tools to create an APK file:
1. The resources folder is compiled using the AAPT (Android Asset Packaging Tool). These are compiled to a single class file called R.java which only contains constants.
2. The java source code is compiled to .class files by javac, and then the class files are converted to Dalvik bytecode by the "dx" tool, which is included in the sdk 'tools'.
3. The android apk-builder takes all the input and builds the APK (Android Packaging Key) file.

Q4. Does Java support **'multiple inheritance'**?

A4. Java supports multiple inheritance by interface only since it can implement multiple interfaces but can extend only one class.

Q5. Can Android application only be written in **Java?**

A5. No, android applications can be written in **Java**(using Android SDK), **Kotlin**(using Android SDK), **C++**(using Android NDK), **JavaScript**(using React Native), **Dart**(using Flutter).

Q6. When should you use a **FrameLayout**?

A6. It comes handy when you need to **stack views one above the other** or in other words it's useful if you need overlapping views, for eg, if you're implementing an overlay or overlay animated views.

Q7. Difference between **Serializable and Parcelable**?

A7. Serialization is the process of **converting an object into a stream of bytes** in order to store an object into memory, so that it can be recreated later, while keeping the object's original state and data.
Serialization is a standard Java interface whereas, **Parcelable** is an Android specific interface where you implement the serialization yourself. It was created to be far more efficient than Serializable.

Q8. What is a **Pending Intent**?

A8. If you want someone to perform any Intent operation at future point of time on behalf of you, then we will use Pending Intent. For example, launch an activity from notification.

Q9. Difference between **AsyncTasks & Threads**?

A9. Thread is used to separate long running operations from UI thread so that performance is improved. One major drawback is you can't update UI from Thread. **AsyncTask** can be used to handle work items shorter than **5ms** in duration like **network calls**. With AsyncTask, you can update UI unlike java Thread. But many long running tasks will choke the performance.

Q10. Why would you do the **setContentView() in onCreate()** of Activity class?

A10. setContentView() is a heavy operation and as **onCreate()** of an Activity is called only once that is why view should be initialized here.

Q11. Scenario in which only **onDestroy()** is called for an activity without **onPause()** and **onStop()**?

A11. If **finish()** is called in the **onCreate()** method of an activity, the system will invoke **onDestroy()** method directly.

Q12. What is **NDK**?

A12. NDK stands for Native Development Kit. By using NDK, you can develop a part of app using native language such as C/C++ to boost the performance.

Q13. Which scenario can only be tested on real devices but not on emulator?

A13. There are scenarios which can only be experienced on a real device. For example, reading data from message, application journey interruptions by a phone call, listening to low battery alerts and handling application functions/components accordingly, bluetooth related actions, memory card mount and unmount etc.

Q14. What is **bitmap pooling** in android?

A14. The purpose of **bitmap pooling** is to reuse bitmaps instead of creating new ones every time. You can create a **bitmap stack**, so when you need **a bitmap**, you check this bitmap stack

to see if the bitmap is available. If it's not available, then you create a new bitmap otherwise you pop a bitmap from the stack and reuse it.

Q15. What's the difference between **onCreate()** and **onStart()**?

A15. The **onCreate()** method is called once during the Activity lifecycle, either when the application starts, or when the Activity has been destroyed and then recreated. The **onStart()** method is called whenever the Activity becomes visible to the user, typically after onCreate() or **onRestart().**

Q16. How does the activity respond when the user rotates the screen?

A16. When the screen is rotated, the current instance of activity is destroyed a new instance of the Activity is created in the new orientation. The **onRestart()** method is invoked first when a screen is rotated. The other lifecycle methods get invoked in the similar flow as they were when the activity was first created.

Q17. How can you clear the back stack when creating a new Activity?

A17: You can either use **FLAG_ACTIVITY_CLEAR_TOP** flag in intent or you can use **FLAG_ACTIVITY_CLEAR_TASK** with **FLAG_ACTIVITY_NEW_TASK**.

Q18. What is **singleton class** in Android?

A18. A **singleton class** creates an object which can be shared with all other classes. For example, your **application instance class**.

Q19. Why is it recommended to use the **default constructor** to create a Fragment?

A19. The benefit of using a default constructor is when the system restores a fragment it will automatically restore your bundle which means that your fragment gets restored to the initial state.

Q20. When should you use a fragment rather than an activity?

A20. Following are some cases when you should create **Fragments** instead of **Activity**:

1. You need a UI component that is going to be used across multiple activities.
2. You need to show multiple views side by side. For example, Tabs Content with View Pager.
3. You need to support for Tablets, Phones for both portrait and landscape. You can weave your layouts with fragments and re-arrange them to fit different layouts and orientations.