What is BroadcastReceiver? What is a broadcast? How can I send a broadcast?

Android apps can send or receive broadcast messages from the Android system and other Android apps, similar to the [publish-subscribe](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern) design pattern. These broadcasts are sent when an event of interest occurs. For example, the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging. Apps can also send custom broadcasts.

To send you use sendOrderedBroadcast, sendBroadcast and LocalBroadcastManager.sendBroadcast

How do you setup a BroadcastReceiver in your project?

Either declare it in the manifest or use the subclass BraodcastReciever and use onRecieve

Explain step by step how would you communicate using a broadcast. How can I pass data using a broadcast?

Step one make class and extend it with BroadcastReciever. Next override the function onRecieve to do what I need when I get the broadcast. Next I make an instance of a BroadcastReciever and create a Intent filter to find only what I need to receive. Then you register the receiver. Then call finish to close the receiver to avoid memory leaks. Next I make an Intent with what I want to broadcast, making sure it uses the proper key so my receiver picks it up. Then I use the sendBroadcast method with the intent created.

What are fragments?

A fragment is an independent Android component which can be used by an activity. A fragment encapsulates functionality so that it is easier to reuse within activities and layouts. A fragment runs in the context of an activity, but has its own life cycle and typically its own user interface.

What is the fragment lifecycle?

onAttach, onCreate, onCreateView, onActivityCreated, onStart, onResume, onPause, onStop, onDestroyView, onDestroy, onDetach

How is a fragment lifecycle associated with the activity lifecycle?

Created onAttach(), onCreate(), onCreateView(), onActivityCreated() Fragment is added and its layout is inflated.

Started onStart() Fragment is active and visible.

Resumed onResume() Fragment is active and ready for user interaction.

Paused onPause() Fragment is paused because the Activity is paused.

Stopped onStop() Fragment is stopped and no longer visible.

Destroyed onDestroyView(), onDestroy(), onDetach() Fragment is destroyed.

What are the ways you can add a fragment to the activity?

Either statically by using the xml or dynamically using the FragmentManager class

Write the code to add a fragment to a layout in the activity?

<fragment

Stuff

/>

Is a fragment added to the back-stack by default? How would you add it if not?

No. You must call transaction.addToBackStack

And then transaction.commit()

How do you remove a fragment from the activity?

getSupportFragmentManager().beginTransaction().

remove(getSupportFragmentManager().findFragmentById(R.id.frame)).commit();

In which method would you bind the views in a fragment? And how? Write the code to bind a textview.

onCreateView

public View onCreateView(LayoutInflater inflater, ViewGroup parent, Bundle savedInstanceState) {

// Defines the xml file for the fragment

return inflater.inflate(R.layout.fragment\_foo, parent, false);

}

How to send data to a fragment though an activity before fragment is added?

You use a bundle to send the information

How do you communicate with the activity from a fragment?

To allow a Fragment to communicate up to its Activity, you can define an interface in the Fragment class and implement it within the Activity. The Fragment captures the interface implementation during its onAttach() lifecycle method and can then call the Interface methods in order to communicate with the Activity.

Can you add a fragment without the xml?

getSupportFragmentManager().beginTransaction().add(android.R.id.content, profileFragment).commit();

What are the factory methods in fragment class used for creating a new instance?

public static Fragment newInstance(String name)

What are Runtime permissions? Which Android version introduced it? Why was this introduced?

Google has changed the way that applications handle permissions. Before, we only dealt with permissions in AndroidManifest.xml, but starting with Android 6.0, we need to check every time for permission-related tasks. Applications need to ask the permission at runtime while they are running and also have to provide enough context on why the permissions are required. Though we have to declare in manifest whenever an application wants to access APIs that need the runtime permission, apps have to check whether that permission has been granted or request the required permission using the support library.

What are normal and dangerous permissions? Name some in each.

Normal permissions are permission once declared in the manifest the system will just grant the app such as internet and Bluetooth. Dangerous permissions contain sensitive data and so not only must be declared in the manifest but the user must authorize the use of those permissions

What is the callback after the user responds to a permission?

onRequestPermissionsResult (int requestCode,

String[] permissions, int[] grantResults) { }

How would you ask for permissions at runtime?

requestPermissions()

Why do we need to send a request code while asking for a permission?

Your app passes the permissions it wants and an integer request code that you specify to identify this permission request. This method functions asynchronously. It returns right away, and after the user responds to the prompt, the system calls the app's callback method with the results, passing the same request code that the app passed to requestPermissions().

What does ActivityCompat.shouldShowRequestPermissionRationale do? What is the return type?

Gets whether you should show UI with rationale for requesting a permission. Boolean