Version 1.0

2018-JuL-08



Interview Feedback

Homework

Presenter: Arturo Sanchez

Mobile Apps Company

3301 Windy Ridge Parkway, Suite 400, Atlanta, GA 30339

# **Interview Feedback**

## Introduction

### Elevator

My name is Arturo Sanchez Chavarria, I am 28 years old and I have been working as an Analyst Developer since 2012.

I like to walk in the park, listen to music, play video games, and watch some movies.

I consider myself as a friendly, responsible, cheerful, optimistic and hardworking person.

I really like to be a developer and working as a team.

### Previous Experience

I have mostly worked with .NET technologies using SQL Server as the main Database server, maintaining, updating and creating new applications for the companies that I have been assign.

I have worked with Visual Studio 2010 up to 2017 using C# as primary language (also used VB .NET about 1 year).

In Visual Studio, I have created Web Applications (WebForms, MVC, and Classic ASP), Windows Applications (WinForms, WPF) and Web Services (WCF, ASMX).

In the information storage, I have mostly used SQL Server (2000 up to 2014), but also worked with Progress (10.1 b), XML, SQL LITE.

I have been a code auditor as one of my responsibilities in previous jobs so I need to work with the best practices in code implementation (OOP), that also includes validating design in databases, web services, and interfaces.

I have been involved in almost all of the steps in the creation of an application:

* Creating work plan
* Requirements survey
* Clients interviews
* Creating prototypes
* Designing the application
  + database solution
  + interfaces design (if applies)
  + Web Services Design (if applies)
* Developing the application
* Unit testing
* Publishing the application
  + Initial loads (Database)
* Post-implementation support.

I have also been involved in optimizing the performance of certain existing process as part of the developing phase.

In Android, I have worked with the platform architecture, supporting multiple languages, units of measure, components, context, intents, life cycle of an activity, intents, serializable and Parcelable classes.

## Questions Asked

### What are the components in android?

App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter your app. Some components depend on others.

There are four different types of app components:

* Activities
* Services
* Broadcast receivers
* Content providers

Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

#### Activities

An activity is the entry point for interacting with the user.

It represents a single screen with a user interface.

An activity facilitates the following key interactions between system and app:

* Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
* Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
* Helping the app handle having its process killed so the user can return to activities with their previous state restored.
* Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The classic example here being share.)

#### Services

A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons.

A Service is a component that runs in the background to perform long-running operations or to perform work for remote processes.

A service does not provide a user interface.

There are two type of services:

* Started Services
  + Tell the system to keep them running until their work is completed
* Bound Services
  + Run because some other app (or the system) has said that it wants to make use of the service.
* A service is implemented as a subclass of Service.

#### BroadCast Receivers

A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.

Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that are not currently running

Although broadcast receivers do not display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work

A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object

#### Content Providers

A content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access.

Through the content provider, other apps can query or modify the data if the content provider allows it.

To the system, a content provider is an entry point into an app for publishing named data items, identified by a URI scheme. Thus, an app can decide how it wants to map the data it contains to a URI namespace, handing out those URIs to other entities, which can in turn use them to access the data.

Content providers are also useful for reading and writing data that is private to your app and not shared.

A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other apps to perform transactions.

### How to use a Navigation Drawer in an Activity?

The navigation drawer is a UI panel that shows your app's main navigation menu. It is hidden when not in use, but appears when the user swipes a finger from the left edge of the screen or, when at the top level of the app, the user touches the drawer icon in the app bar.

You use a Navigation Drawer you need to add the following dependencies to your apps module's build.gradle file:

dependencies {

implementation 'com.android.support:appcompat-v7:27.1.1'

implementation 'com.android.support:design:27.1.1'

}

To add a navigation drawer, declare your layout with a DrawerLayout as the root view. Inside the DrawerLayout, add a layout for the main content for the UI (your primary layout when the drawer is hidden) and another view that contains the contents of the navigation drawer.

To configure the menu items listed in the drawer, specify a menu resource with the app:menu attribute and create the menu resource with the corresponding file name.

You can make a group of items single-selectable by applying android:checkableBehavior="single" to a group. This allows you to show which list item is currently selected.

Optionally, you can add a header at the top of the drawer, by specifying a layout with the app:headerLayout.

To receive callbacks when the user taps a list item in the drawer, implement the OnNavigationItemSelectedListener interface and attach it to your NavigationView by calling setNavigationItemSelectedListener().

If your UI design includes an app bar, you should also allow users to open and close the drawer by touching the drawer icon on the top left of the app bar.

open your manifest file and set the app theme to one without the action bar, such as Theme.AppCompat.Light.NoActionBar:

<manifest ...>

<application

...

android:theme="@style/Theme.AppCompat.Light.NoActionBar" >

...

</manifest>

To apply the toolbar as the app bar, first make sure your activity extends from AppCompatActivity. Then call setSupportActionBar() and pass the Toolbar object from your layout.

Now add the button that opens the navigation drawer. You first need to add the menu button's icon to your project.

To open the drawer when the user taps on the nav drawer button, override onOptionsItemSelected() to hook into the options menu framework and listen for when the user taps the item with the ID android.R.id.home. Then call openDrawer() to open the nav drawer.

### What is the overflow menu used for? Implement?

The overflow menu (also referred to as the options menu) is a menu that is accessible to the user from the device display and allows the developer to include other application options beyond those included in the user interface of the application.

This allows the user to scroll through all of the options in the menu that are not visible.

An overflow menu is created by overriding the onCreateOptionsMenu() method of the corresponding activity and then inflating the menu’s XML file.

@Override

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.activity\_menu\_app, menu);

return true;

}

### What are launch modes? Explain each

Launch mode is an instruction for Android OS which specifies how the activity should be launched. It instructs how any new activity should be associated with the current task.

There are four launch modes for activity. They are:

1. Standard
   1. This is the default launch mode of an activity (If not specified). It creates a new instance of an activity in the task from which it was started. Multiple instances of the activity can be created and multiple instances can be added to the same or different tasks.
   2. You can create the same activity multiple times in the same task as well as in different tasks.
2. SingleTop
   1. In this launch mode if an instance of activity already exists at the top of the current task, a new instance will not be created and Android system will route the intent information through onNewIntent(). If an instance is not present on top of task then new instance will be created.
   2. Using this launch mode you can create multiple instance of the same activity in the same task or in different tasks only if the same instance does not already exist at the top of stack.
3. SingleTask
   1. In this launch mode a new task will always be created and a new instance will be pushed to the task as the root one. If an instance of activity exists on the separate task, a new instance will not be created and Android system routes the intent information through onNewIntent() method. At a time only one instance of activity will exist.
4. SingleInstance
   1. This is very special launch mode and only used in the applications that has only one activity. It is similar to singleTask except that no other activities will be created in the same task. Any other activity started from here will create in a new task.

### What are intent flags?

Android also provides Activity flags by which you can change the default behavior of Activity association with Tasks while starting it via startActivity() method. These flags values can be pass through Intent extra data.

FLAG\_ACTIVITY\_NEW\_TASK

* This flag works similar to “launchMode = singleTask”.

FLAG\_ACTIVITY\_CLEAR\_TASK

* This flag will cause any existing task that would be associated with the activity to be cleared before the activity is started. The activity becomes the new root of an otherwise empty task, and any old activities are finished.

FLAG\_ACTIVITY\_SINGLE\_TOP

* This flag works similar to “launchMode = singleTop”.

FLAG\_ACTIVITY\_CLEAR\_TOP

* If set, and the activity being launched is already running in the current task, then instead of launching a new instance of that activity, all of the other activities on top of it will be closed and this Intent will be delivered to the (now on top) old activity as a new Intent.

### What are different ways to create worker thread?

By creating a Thread, a Runnable and an AsyncTask

### Difference between Thread and Runnable?

1. Runnable uses “implements” while Thread uses “extends”.
2. Implementing Runnable is the preferred way to do it. Here, you are not really specializing or modifying the thread’s behavior. You are just giving the thread something to run. That means composition is the better way to go.
3. Java only supports single inheritance, so you can only extend one class.
4. Instantiating an interface gives a cleaner separation between your code and the implementation of threads.
5. Implementing Runnable makes your class more flexible. If you extend thread then the action you are doing is always going to be in a thread. However, if you extend Runnable it does not have to be. You can run it in a thread, pass it to some kind of executor service, or just pass it around as a task within a single threaded application.
6. By extending Thread, each of your threads has a unique object associated with it, whereas implementing Runnable; many threads can share the same runnable instance.

### How to communicate with a thread?

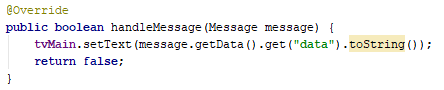
Every app has its own special thread that runs UI objects such as View objects; this thread is called the UI thread. Only objects running on the UI thread have access to other objects on that thread. Because tasks that you run on a thread from a thread pool aren't running on your UI thread, they don't have access to UI objects. To move data from a background thread to the UI thread, use a Handler that's running on the UI thread.

Handler is part of the Android system's framework for managing threads. A Handler object receives messages and runs code to handle the messages. Normally, you create a Handler for a new thread, but you can also create a Handler that's connected to an existing thread. When you connect a Handler to your UI thread, the code that handles messages runs on the UI thread.

### How to use Handlers to communicate?

Instantiate the Handler object in the constructor for the class that creates your thread pools, and store the object in a global variable. Connect it to the UI thread by instantiating it with the Handler(Looper) constructor. This constructor uses a Looper object, which is another part of the Android system's thread management framework. When you instantiate a Handler based on a particular Looper instance, the Handler runs on the same thread as the Looper.

Inside the Handler, override the handleMessage() method. The Android system invokes this method when it receives a new message for a thread it is managing; all of the Handler objects for a particular thread receive the same message.



### What are Loopers?

Looper is a class which is used to execute the Messages(Runnables) in a queue. Normal threads have no such queue, e.g. simple thread does not have any queue. It executes once and after method execution finishes, the thread will not run another Message(Runnable).

If someone wants to execute multiple messages(Runnables) then he should use the Looper class which is responsible for creating a queue in the thread.

There is prepare() method to prepare the Looper. Then you can use loop() method to create a message loop in the current thread and now your Looper is ready to execute the requests in the queue until you quit the loop.

### What is the lifecycle of an AsyncTask? Which Thread?

AsyncTask enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

When an asynchronous task is executed, the task goes through 4 steps:

1. onPreExecute()
   1. Invoked on the UI thread before the task is executed.
   2. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
2. doInBackground(Params...)
   1. Invoked on the background thread immediately after onPreExecute() finishes executing.
   2. This step is used to perform background computation that can take a long time.
   3. The parameters of the asynchronous task are passed to this step.
   4. The result of the computation must be returned by this step and will be passed back to the last step.
   5. This step can also use publishProgress(Progress...) to publish one or more units of progress.
   6. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.
3. onProgressUpdate(Progress...)
   1. Invoked on the UI thread after a call to publishProgress(Progress...).
   2. The timing of the execution is undefined.
   3. This method is used to display any form of progress in the user interface while the background computation is still executing.
   4. For instance, it can be used to animate a progress bar or show logs in a text field.
4. onPostExecute(Result)
   1. Invoked on the UI thread after the background computation finishes.
   2. The result of the background computation is passed to this step as a parameter.
   3. Invoked on the UI thread.

### What are the Generics that we pass in AsyncTask?

The three types used by an asynchronous task are the following:

1. Params, the type of the parameters sent to the task upon execution.
2. Progress, the type of the progress units published during the background computation.
3. Result, the type of the result of the background computation.

Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type Void:

private class MyTask extends AsyncTask<Void, Void, Void> { ... }

### What are different types of intents? Explain

There are two types of intents:

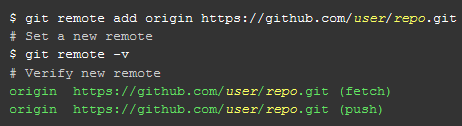
1. Explicit intents.
   1. Specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name.
   2. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.
   3. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.
2. Implicit intents.
   1. These do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
   2. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

### How to add a remote to the git repo?

To add a new remote, use the git remote add command on the terminal, in the directory your repository is stored at.

The git remote add command takes two arguments:

* A remote name, for example, origin
* A remote URL, for example, <https://github.com/user/repo.git>



### What does Git diff do?

Diffing is a function that takes two input data sets and outputs the changes between them.

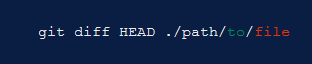
git diff is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be commits, branches, files and more.

The git diff command is often used along with git status and git log to analyze the current state of a Git repo.

#### Comparing files: git diff file

The git diff command can be passed an explicit file path option.

When a file path is passed to git diff the diff operation will be scoped to the specified file.



This example is scoped to ./path/to/file when invoked, it will compare the specific changes in the working directory, against the index, showing the changes that are not staged yet. By default git diff will execute the comparison against HEAD. Omitting HEAD in the example above git diff ./path/to/file has the same effect.

#### Comparing all changes

Invoking git diff without a file path will compare changes across the entire repository.

#### Changes since last commit

By default git diff will show you any uncommitted changes since the last commit.

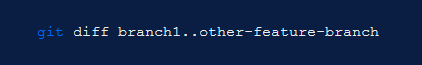


#### Comparing files between two different commits

git diff can be passed Git refs to commits to diff. Some example refs are, HEAD, tags, and branch names. Every commit in Git has a commit ID which you can get when you execute GIT LOG. You can also pass this commit ID to git diff.

#### Comparing two branches

Branches are compared like all other ref inputs to git diff.



#### Comparing files from two branches

To compare a specific file across branches, pass in the path of the file as the third argument to git diff



### How to make branches in Git? Why do we need them?

Branching means you diverge from the main line of development and continue to do work without messing with that main line.

The command for creating a new Branch is:

git branch [Name of the branch]

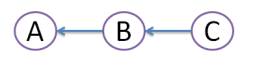
Example

git branch MyBranch

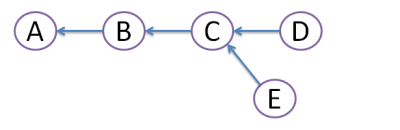
A branch helps you if you want to have a working version of your code and also testing some new functionality, or making changes without affecting the main branch, this can be fixing a bug or having a coworker work with the same project.

### What is the different between rebase and merge?

Suppose originally there were 3 commits, A,B,C:

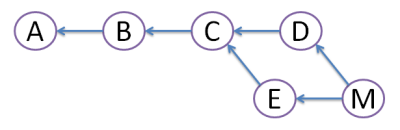


Then developer Dan created commit D, and developer Ed created commit E:



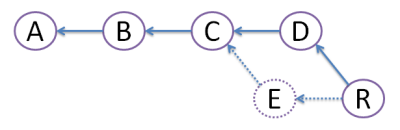
Obviously, this conflict should be resolved somehow. For this, there are 2 ways:

**MERGE**:



Both commits D and E are still here, but we create merge commit M that inherits changes from both D and E. However, this creates *diamond* shape, which is somewhat unclear.

**REBASE**:



We create commit R, which actual file content is identical to that of merge commit M above. But, we get rid of commit E, like it never existed (denoted by dots - vanishing line). Because of this obliteration, E should be local to developer Ed and should have never been pushed to any other repository.

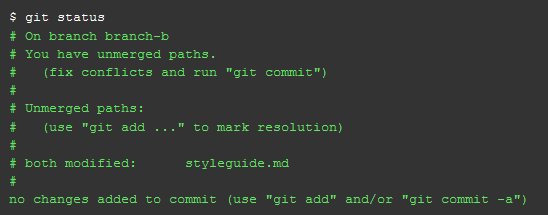
Advantage of rebase is that *diamond* shape is avoided, and history stays in straight line.

### How to resolve conflicts when merging branches?

To resolve a merge conflict caused by competing line changes, you must choose which changes to incorporate from the different branches in a new commit.

If you and another person both edited the file styleguide.md on the same lines in different branches of the same Git repository, you will get a merge conflict error when you try to merge these branches. You must resolve this merge conflict with a new commit before you can merge these branches.

Steps to resolve conflicts:

1. Open Git Bash.
2. Navigate to the local Git repository that has the merge conflict.
   1. 
3. Generate a list of the files affected by the merge conflict.
   1. 
   2. The above it’s an example where the conflict it’s generated in the styleguide.md
4. Open your favorite text editor, and navigate to the file that has merge conflicts.
5. Search the file and identify the changes:
   1. 
      1. Beginning of the changes in the HEAD
   2. 
      1. Divides your changes with the other branch changes
   3. 
      1. End of the changes in the other branch.
6. Decide if you want to keep only your branch's changes, keep only the other branch's changes, or make a brand new change, which may incorporate changes from both branches. Delete the conflict markers <<<<<<<, =======, >>>>>>> and make the changes you want in the final merge.
7. Add or stage your changes.
   1. 
8. Commit your changes with a comment.
   1. 