What's your favorite tool or library for Android? Why is it so useful?

I've been very pleased with the Retrofit, and more recently, Retrofit2, libraries to facilitate the retrieval of data via REST-based webservices. The nice thing about Retrofit is that it simplifies a lot of the boilerplate code I'd previously created to retrieve and parse JSON data.

In addition to the simplicity of its implementation, Retrofit integrates well with Android patterns and best practices, and offers performance benefits over other networking tools like AsyncTasks.

Because I'm always looking to stay up to date with the Android ecosystem and open-source libraries, I'm currently studying RxJava and I look forward to using it along with Retrofit to simplify networking and thread management in my code.

You want to open a map app from an app that you're building. The address, city, state, and ZIP code are provided by the user. What steps are involved in sending that data to a map app?

Sending the data to a map app requires an implicit intent.

First, I'd create a geolocation `Uri` from the address string.

Then, I'd create a new Intent passing it `Intent.ACTION_VIEW`

Then, I'd set the geolocation `Uri` to the `Intent` using `setData()`.

Then, I'd check to see that an app exists with the ability to resolve the `Intent` and call `startActivity()`, passing it the `Intent`, if so.

If there's no app on the device to show the map, I'd alert the user with a `Toast`.

```
public void showMapFromAddress(String address) {

   Uri geoLocation = Uri.parse("geo:" + address);
   Intent intent = new Intent(Intent.ACTION_VIEW);
   intent.setData(geoLocation);
   if (intent.resolveActivity(getContext().getPackageManager()) != null) {
       startActivity(intent);
   } else {
       Toast.makeText(getContext(), "There was a problem showing the map",
Toast.LENGTH_SHORT).show();
   }
}
```

Implement a method to perform basic string compression using the counts of repeated characters. For example, the string aabccccccaaa would become a2b1c5a3. If the "compressed" string would not become smaller than the original string, your method should return the original string. The method signature is: “public static String compress(String input)” You must write all code in proper Java, and please include import statements for any libraries you use.

```
public static String compress(String input){
	String currentLetter;
	String streakingLetter = input.substring(0, 1);
	String resultString= "";
	int streak = 1;
	for(int i  = 1; i < input.length();  i++) {
		currentLetter = input.substring(i, i+1);
			if(currentLetter.equals(streakingLetter)) {
				streak++;
			} else {
				resultString = resultString.concat(streakingLetter);
				resultString =
resultString.concat(Integer.toString(streak));
				streak = 1;
				streakingLetter = currentLetter;
			}
	}
	resultString = resultString.concat(streakingLetter);
	resultString = resultString.concat(Integer.toString(streak));
	if(input.length() <= resultString.length()) {
		return input;
	} else {
		return resultString;
	}
}
```

Rationale:

1. Set variables to keep track of the `currentLetter`, the `streakingLetter`, the `resultString` to be returned, and the length of the current `streak`.
2. Begin by setting the first letter of the `input` as the `streakingLetter` and the `streak` length to 1.
3. Loop through each letter in the `input` String beginning with the second letter, and set each letter to `currentLetter`.
4. If the `currentLetter` is the ***same as*** the one before it, increment the `streak` length.
5. If the `currentLetter` is ***different than*** the one before it, add the one before it (the current `streakingLetter`) to the `resultString` followed by the `streak` length.
6. Then, reset the `streak` to 1, and make the `currentLetter` the new `streakingLetter`.
7. When the loop is finished, add the final `streakingLetter` and `streak` to the `resultString`.

8. If the original `input` String is shorter than the final `resultString`, return the original `input` String, otherwise, return the `resultString`.

List and explain the differences between four different options you have for saving data while making an Android app. Pick one, and explain (without code) how you would implement it.

Shared Preferences are used for storing key-value pairs of the primitive data types: `int`, `float`, `long`, `boolean`, `String`, as well as string arrays. Shared Preference files will persist across app sessions, are ideally suited for storing user options, settings, preferences, and other simple values.

SQL Databases are provided by Android for dealing with larger sets of tabular data. Ideally, SQL databases in Android should be wrapped in a `ContentProvider`. `ContentProvider`s not only provide the ability to share data with other apps, but also provide various utilities for performing CRUD operations and keeping the UI in sync with database content.

Internal Storage can be utilized by Android to save files on a device. By default these files are only accessible to the application and are removed when the application is uninstalled.

External Storage can also be utilized by Android to save files on a device. Unlike internal storage, however, files saved to external storage are world-readable to both the user and other apps. External storage includes files which are stored in non-removable device storage as well as removable storage media such as an SD card. Reading and writing files to external storage requires a `WRITE_EXTERNAL_STORAGE`permission in the app's `manifest.xml` file.

Assuming a single preference file is all that is needed for a given application, a `SharedPreferences`object can be returned by calling the `getDefaultSharedPreferences()`method on the `PreferenceManager`and passing it the relevant `Context`.

To read values from the `SharedPreferences`object, we can call various get() methods (`getBoolean()`, `getInt()`, `getString()`, etc.) passing a key `String` and a value to return if the requested value doesn't exist.

To store and edit preferences we can call the `edit()` method on the `SharedPreferences` object to obtain an `Editor` instance (`SharedPreferences.Editor`Interface). We can then edit the values by chaining various put methods (`putBoolean()`, `putInt()`, `putString()`, etc.) on to the `Editor`, passing these methods both the key `String` and value.

If we're editing preference values on the Main Thread, we call `apply()` on the `Editor` to perform the disk-write operation asynchronously. If we're editing preference values on a background thread, we can call `commit()` on the `Editor` which writes to disk synchronously.

What are your thoughts about Fragments? Do you like or hate them? Why?

At first, Fragments were a bit perplexing to me. As they were initially presented in the Udacity Android Developer Nanodegree course, they took up the full dimension of the UI screen just like the Activity in which they were contained. As such, I didn't immediately see the need for them. As the course went on, however, I saw exactly how Fragments are necessary for building dynamic and custom UIs across a wide range of devices. It was in building a Master-Detail UI--where two fragments are visible in the same Activity on tablet-sized devices--that I really began to see how important Fragments are.

One of the reasons I decided to learn Android development was because I saw such remarkable promise in the multitude of devices on which Android can and will run. With these many devices, however, come various form factors for which we must design. This means designing for screens as small as watches and as big as large-screen televisions. As such, I've come to enjoy working with Fragments--they always remind me of the wide range of devices which support Android, and of the possibilities of the platform for the future.

If you were to start your Android position today, what would be your goals a year from now?

Given that the opening at Dropsource is a contract position, my goal in a year would be to work for Dropsource as a full-time developer. Beyond that, I look forward to honing my skills further and bringing these talents to bear at Dropsource, a company which will be in the forefront of the exciting and ever-changing landscape of mobile development.

I share the Dropsource mission of helping others create production-quality mobile applications by breaking down technological barriers. In my volunteer work with young people, for example, I've utilized and created various tools to spur creativity and interest in mobile development, despite the fact that my students lack the technological skills to bring their ideas to fruition via traditional development practices.

The Dropsource platform seems like an amazing vehicle to make mobile application development accessible to the masses, and I would be proud to be a part of the Dropsource team.