SHORT QUESTIONS

1. **What is AVD? * mam**

An AVD is an Android Virtual Device. It represents an Android emulator, which emulates a particular configuration of an actual Android device.

2. **What is diff b/w android : versionCode and android : versionName attributes in the AndroidManifest.xml file?**

The android:versionCode attribute is used to programmatically check whether an application can be upgraded. It should contain a running number. The android:versionName attribute is used mainly for displaying to the user.

3. **What is use of Strings.xml file? * mam**

The strings.xml fi le is used to store all string constants in your application. This enables you to easily localize your application by simply replacing the strings and then recompiling your application.

4. **What will happen if you have two or more activities with same intent filter action name?**

The Android OS will display a dialog from which users can choose which activity they want to use.

5. **Write code to invoke built in browser application?**

Use the following code:

```
Intent i = new
Intent(android.content.Intent.ACTION_VIEW,
Uri.parse("http://www.amazon.com"));
startActivity(i);
```

6. **Which components can you specify in an intent filter?**

In an intent filter, you can specify the following: action, data, type, and category.

7. **What is different between Toast class and NotificationManager class? * mam**

The Toast class is used to display alerts to the user; it disappears after a few seconds. The NotificationManager class is used to display notifications on the device's status bar. The alert displayed by the NotificationManager class is persistent and can only be dismissed by the user when selected.

8. **Name the two ways to add fragments to an activity?**

You can either use the <fragment> element in the XML fi le, or use the FragmentManager and FragmentTransaction classes to dynamically add/remove fragments from an activity.

9. Name one key diff b/w fragment and an activity? *mam

One of the main differences between activities and fragments is that when an activity goes into the background, the activity is placed in the back stack. This allows an activity to be resumed when the user presses the Back button. Conversely, fragments are not automatically placed in the back stack when they go into the background.

10. What is diff b/w dp unit and px unit? Which one should you use to specify the dimensions of a view?

The dp unit is density independent and 1dp is equivalent to one pixel on a 160 dpi screen. The px unit corresponds to an actual pixel on screen. You should always use the dp unit because it enables your activity to scale properly when run on devices of varying screen size.

11. Why is the Absolute Layout not recommended for use? * mam

With the advent of devices with different screen sizes, using the AbsoluteLayout makes it difficult for your application to have a consistent look and feel across devices.

12. What is diff b/w onPause() method and onSaveInstanceState() method? * mam

The onPause() event is fi red whenever an activity is killed or sent to the background. The onSaveInstanceState() event is like the onPause() event, except that it is not always called, such as when the user presses the back button to kill the activity.

13. Name three method you can override to save an activity's state? In what instances should you use the various methods?

The three events are onPause(), onSaveInstanceState(), and onRetainNonConfigurationInstance(). You generally use the onPause() method to preserve the activity's state because the method is always called when the activity is about to be destroyed. However, for screen orientation changes, it is easier to use the onSaveInstanceState() method to save the state of the activity using a Bundle object. The onRetainNonConfigurationInstance() method is useful for momentarily saving data which might be too large to fi t into a Bundle object.

14.How do you add action items to the action bar?

Adding action items to the Action Bar is similar to creating menu items for an options menu — simply handle the onCreateOptionsMenu() and onOptionsItemSelected() events.

15.How do you programmatically determine whether a RadioButton is checked?

You should check the isChecked() method of each RadioButton to determine whether it has been checked.

16.How do you access the string resource stored in strings.xml file?

You can use the getResources() method.

17.Write the code snippet to obtain the current date?

The code snippet to obtain the current date is as follows:

```
//—-get the current date—-
Calendar today = Calendar.getInstance();
yr = today.get(Calendar.YEAR);
month = today.get(Calendar.MONTH);
day = today.get(Calendar.DAY_OF_MONTH);
showDialog(DATE_DIALOG_ID);
```

18.Name three specialized fragments you can use in your application and describe their use?

The three specialized fragments are ListFragment, DialogFragment, and PreferenceFragment. The **ListFragment** is useful for displaying a list of items, such as an RSS listing of news items. The **DialogFragment** allows you to display a dialog window modally and is useful to get a response from the user before allowing him to continue with your application. The **PreferenceFragment** displays a window containing your application's preferences and allows the user to edit them directly in your application.

19.What is the purpose of image switcher?

The ImageSwitcher enables images to be displayed with animation. You can animate the image when it is being displayed, as well as when it is being replaced by another image.

20.Name two methods you need to override when implementing on options menu in your activity?

The two methods are onCreateOptionsMenu() and onOptionsItemSelected().

21. Name two methods you need to override when implementing on context menu in your activity?

The two methods are onCreateContextMenu() and onContextItemSelected().

22. How do you prevent the web view from invoking the device's web browser when redirection occurs in the web view?

To prevent launching the device's web browser, you need to implement the WebViewClient class and override the shouldOverrideUrlLoading() method.

23. How do you display the prefrences of your application using an activity?

You can do so using the PreferenceActivity class.

24. Name method that enables you to obtain path of external storage of an android device?

The method name is getExternalStorageDirectory().

25. What is permission you need to declare when writing files to external storage?

The permission is WRITE_EXTERNAL_STORAGE.

26. Write the query to retrieve all contacts from the contacts application that contains the word "jack"?

The code is as follows:

```
Cursor c;
if (android.os.Build.VERSION.SDK_INT <11) {
//---before Honeycomb---
c = managedQuery(allContacts, projection,
ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?",
new String[] {"%jack"},
ContactsContract.Contacts.DISPLAY_NAME + " ASC");
} else {
//---Honeycomb and later---
CursorLoader cursorLoader = new CursorLoader(
this,
allContacts,
projection,
ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?",
new String[] {"%jack"},
```

ContactsContract.Contacts.DISPLAY_NAME + “ ASC”);
c = cursorLoader.loadInBackground();
}

27.Name the methods that you need to override in your own application of a content provider? * mam

The methods are getType(), onCreate(), query(), insert(), delete(), and update().

28.How do you register a content provider in your AndroidManifest.xml file? * mam

The code is as follows:

```
<provider android:name=”BooksProvider”
android:authorities=”net.learn2develop.provider.Books” />
```

29.Name two ways in which you can send SMS msg in your android application?

You can either programmatically send an SMS message from within your Android application or invoke the built-in Messaging application to send it on your application’s behalf.

30.Name the permission you need to declare in your AndroidManifest.xml file for sending and receiving sms msg?

The two permissions are SEND_SMS and RECEIVE_SMS.

31.How do you notify an activity from BroadcastReceiver?

The Broadcast receiver should fi re a new intent to be received by the activity. The activity should implement another BroadcastReceiver to listen for this new intent.

32.What is diff b/w geocoding and reverse geocoding? * mam

Geocoding is the act of converting an address into its coordinates (latitude and longitude). Reverse geocoding converts a pair of location coordinates into an address.

33.Name two location providers that you can use to obtain your location data? * mam

The two providers are as follows:

➤ LocationManager.GPS_PROVIDER

➤ LocationManager.NETWORK_PROVIDER

34.What method is used for monitoring locations? * mam

The method is addProximityAlert().

35. Name the permission you need to declare in your AndroidManifest.xml file for an HTTP connection?

The permission is INTERNET.

36. Name the classes used for dealing with JSON msg?

The classes are JSONArray and JSONObject.

37. Name the class for performing background asynchronous tasks?

The class is AsyncTask.

38. What is purpose of IntentService Class?

The IntentService class is similar to the Service class, except that it runs the tasks in a separate thread and automatically stops the service when the task has finished execution.

39. Name three methods you need to implement in AsyncTask Class?

The three methods are doInBackground(), onProgressUpdate(), and onPostExecute().

40. How can a service notify an activity of an event happening?

The service can broadcast an intent, and the activity can register an intent using an IntentFilter class.

41. How do you specify the minimum version of android required by your application?

You specify the minimum Android version required using the minSdkVersion attribute in the AndroidManifest.xml file.

42. How do you generate a self-signed certificate for signing your Android Application?

To generate a certifi cate, you can either use the keytool.exe utility from the Java SDK or use Eclipse’s Export feature.

43. How do you configure your android device to accept applications from non-market sources?

Go to the Settings application and select the Security item. Check the “Unknown sources” item.

1. **What is IDE?**
   An Integrated development environment (IDE) is software for building applications that combines common developer tools into a single GUI.
2. **What are Views?**
   An activity contains views and ViewGroups. A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and textboxes
3. **Three Methods to save data?**
   1.Shared Preferences
   2.Internal Storage
   3.External Storage
4. **Permission**
   1.Internet Permission
   2.Locations Permission
5. **How to set internal and external paths?**
   The method for external storage is getExternalStorageDirectory().
   The method name of internal storage is getFilesDir()
6. **Which protocols are used to configure emails?**
   The Gmail/Email application on Android enables you to confi gure an e-mail account using POP3 or IMAP.
7. **Android SDK & JDK?**
   An SDK is a set of software development kit or tools that allow applications to be created for certain software packages or platforms. JDK is most widely used sdk and is an extension of the SDK responsible for writing and running java programs.
8. **Menus?**
   Menus are useful for displaying additional options that are not directly visible on the main UI of an application.
   Options menu — Displays information related to the current activity. In Android, you activate the options menu by pressing the MENU button.
   Context menu — Displays information related to a particular view on an activity. In Android, to activate a context menu you tap and hold on to it.
9. **DB Adapter Class?**
   When implementing a database on Android, a common practice is to create a class which interacts with the SQLite database and also with the activities

of the app. This class is called DBAdapter and contains all the necessary code for creating the tables and the assigned fields.

LONG QUESTIONS

1. **What is Android? Its Features? Architecture with diagram? And Android Stack?**

Android is a mobile operating system that is based on a modifi ed version of Linux. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work (as well as its development team).

The main advantage of adopting Android is that it offers a unifi ed approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android. In the world of smartphones, applications are the most important part of the success chain.

Features of Android:

Because Android is open source and freely available to manufacturers for customization, there are no fixed hardware or software configurations. However, Android itself supports the following features:

➤ **Storage** — Uses SQLite, a lightweight relational database, for data storage.

➤ **Connectivity** — Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), Wi-Fi, LTE, and WiMAX.

➤ **Messaging** — Supports both SMS and MMS.

➤ **Web browser** — Based on the open source WebKit, together with Chrome's V8 JavaScript engine.

➤ **Media support** — Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.

➤ **Hardware support** — Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS.

➤ **Multi-touch** — Supports multi-touch screens.

➤ **Multi-tasking** — Supports multi-tasking applications.

➤ **Flash support** — Android 2.3 supports Flash 10.1.

➤ **Tethering** — Supports sharing of Internet connections as a wired/wireless hotspot.

Architecture of Android:

The Android OS is roughly divided into five sections in four main layers:

➤ **Linux kernel** — This is the kernel on which Android is based. This layer contains all the low-level device drivers for the various hardware components of an Android device.

➤ **Libraries** — These contain all the code that provides the main features of an Android OS. For example, the SQLite library provides database support so that an application can use it for data storage. The WebKit library provides functionalities for web browsing.

➤ **Android runtime** — At the same layer as the libraries, the Android runtime provides a set of core libraries that enable developers to write

Android apps using the Java programming language. The Android runtime also includes the Dalvik virtual machine, which enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.

➤ **Application framework** — Exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications.

➤ **Applications** — At this top layer, you will find applications that ship with the Android device (such as Phone, Contacts, Browser, etc.), as well as applications that you download and install from the Android Market. Any applications that you write are located at this layer.

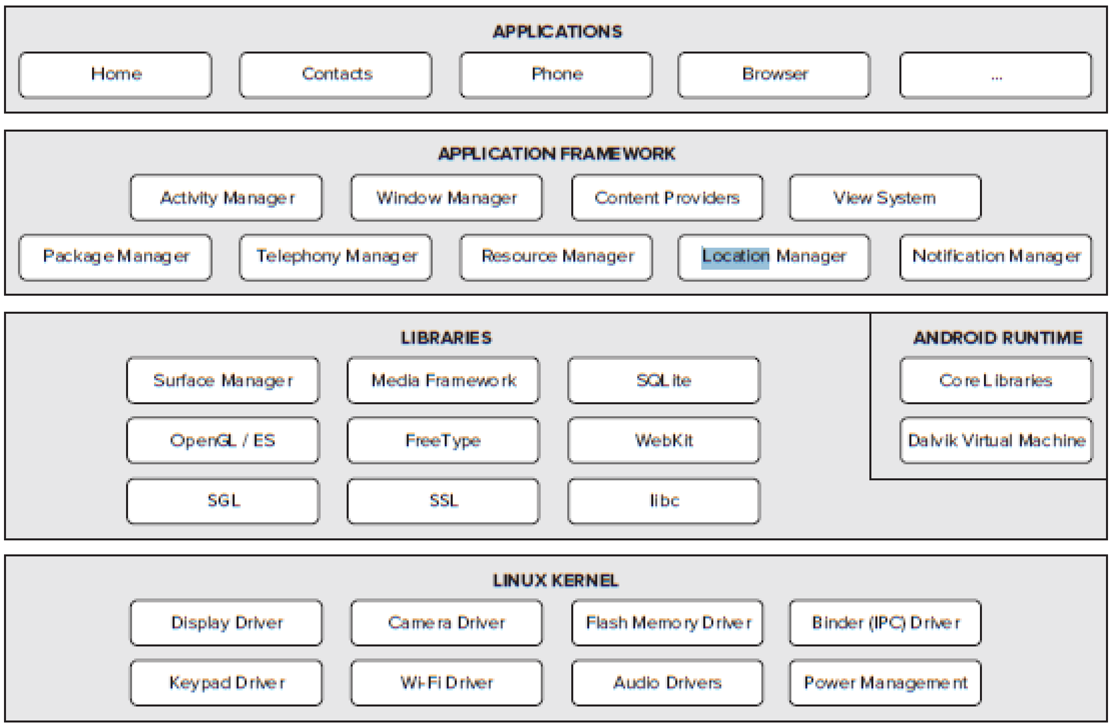


FIGURE 1-1

2. Define Activity? It's Lifecycle With diagram?

Activity:

An Activity is the single screen in android. It is like window or frame of java. By the help of activity you can place all your UI components or widgets in a single screen.

Life Cycle of Activity:

The Activity base class defines a series of events that govern the life cycle of an activity. The Activity class defines the following events:

➤ **onCreate()** — Called when the activity is first created

➤ **onStart()** — Called when the activity becomes visible to the user.

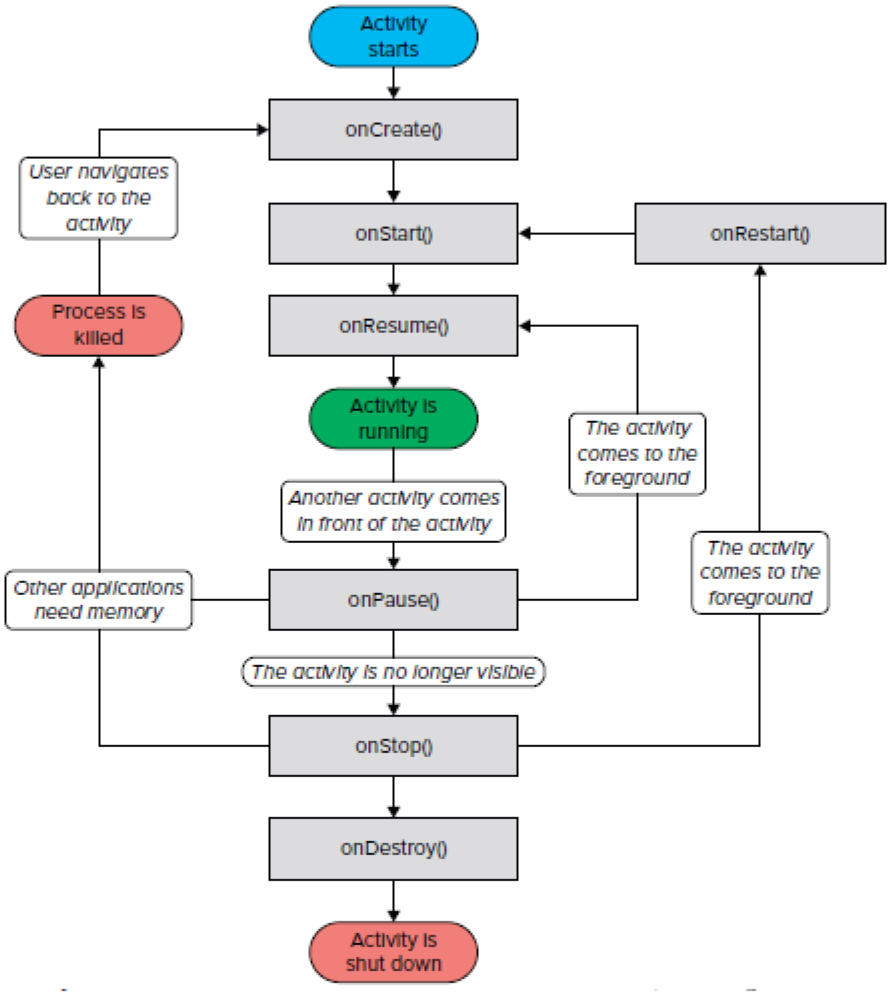
➤ **onResume()** — Called when the activity starts interacting with the user.

➤ **onPause()** — Called when the current activity is being paused and the previous activity is being resumed.

➤ **onStop()** — Called when the activity is no longer visible to the user.

➤ **onDestroy()** — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)

➤ **onRestart()** — Called when the activity has been stopped and is restarting again.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

➤ Use the onCreate() method to create and instantiate the objects that you will be using in your application.

➤ Use the onResume() method to start any services or code that needs to run while your activity is in the foreground.

➤ Use the onPause() method to stop any services or code that does not need to run when your activity is not in the foreground.

➤ Use the onDestroy() method to free up resources before your activity is destroyed.

3. Define Fragments? Lifecycle?

Fragments:

A Fragment is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of sub-activity. Fragments are also known as mini-activities.

Life Cycle of Fragments:

Like activities, fragments have their own life cycle. Understanding the life cycle of a fragment enables you to properly save an instance of the fragment when it is destroyed, and restore it to its previous state when it is recreated.

➤ **onAttached()** — Called when the fragment has been associated with the activity. The Fragment and activity is not fully initialized.

➤ **onCreate()** The system calls this method when creating the fragment.

➤ **onCreateView()** — Called to create the view for the fragment.

➤ **onActivityCreated()** — Called when the activity's onCreate() method has been returned.

➤ **onStart()** —The onStart() method is called once the fragment gets visible.

➤ **onResume()** — Fragment becomes active.

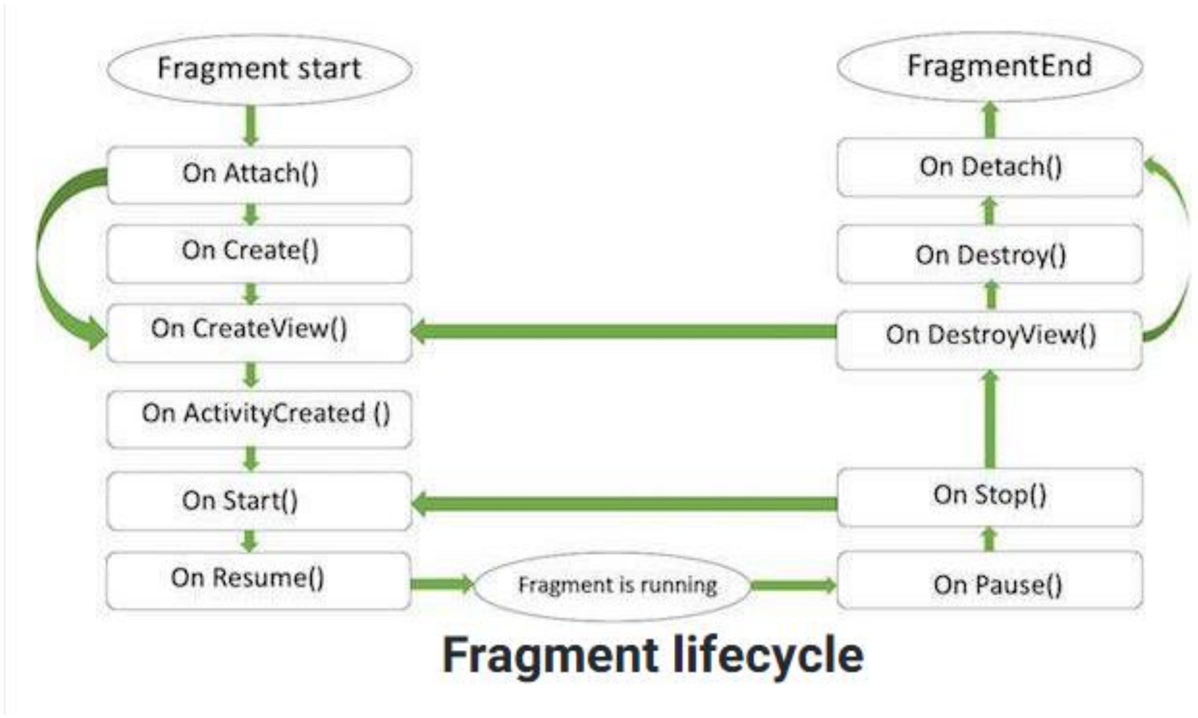
➤ **onPause()** — The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.

➤ **onStop()** — Fragment going to be stopped by calling onStop()

➤ **onDestroyView()** — Called when the fragment's view is being removed

➤ **onDestroy()** — Called to do final clean up of the fragment's state

➤ **onDetach()** — Called when the fragment is detached from the activity.



when a fragment is being created, it goes through the following states:

- ➤ onAttach()
- ➤ onCreate()
- ➤ onCreateView()
- ➤ onActivityCreated()

When the fragment becomes visible, it goes through these states:

- ➤ onStart()
- ➤ onResume()

When the fragment goes into the background mode, it goes through these states:

- ➤ onPause()
- ➤ onStop()

When the fragment is destroyed (when the activity it is currently hosted in is destroyed), it goes through the following states:

- ➤ onPause()
- ➤ onStop()
- ➤ onDestroyView()
- ➤ onDestroy()

➤ onDetach()
Like activities, you can restore an instance of a fragment using a Bundle object, in the following states:

➤ onCreate()
➤ onCreateView()
➤ onActivityCreated()

4. What is Layout?Frame Layout?Table Layout?Linear Layout?

Layout:

A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects.

1. Frame Layout:

Frame Layout is designed to block out an area on the screen to display a single item. Generally, Frame Layout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.

Frame Layout Attributes

Following are the important attributes specific to Frame Layout:

android:id
This is the ID which uniquely identifies the layout.

android:foreground
This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb".

android:foregroundGravity
Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center.

android:measureAllChildren
Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

Example:

Consider the following content in main.xml:
<?xml version=”1.0” encoding=”utf-8”?>

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent">

<ImageView
android:src="@drawable/ic_launcher"
android:scaleType="fitCenter"
android:layout_height="250px"
android:layout_width="250px"/>

<TextView
android:text="Frame Demo"
android:textSize="30px"
android:textStyle="bold"
android:layout_height="fill_parent"
android:layout_width="fill_parent"
android:gravity="center"/>
</FrameLayout>
```

2. Table Layout:

The TableLayout groups views into rows and columns. You use the <TableRow> element to designate a row in the table. Each row can contain one or more views. Each view you place within a row forms a cell. The width of each column is determined by the largest width of each cell in that column.

Table Layout Attributes:

android:id

This is the ID which uniquely identifies the layout.

android:collapseColumns

This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.

android:shrinkColumns

The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.

android:stretchColumns

The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

Example:

Consider the content of main.xml shown here:

```
<TableLayout
xmlns:android=”http://schemas.android.com/apk/res/android”
android:layout_height=”fill_parent”
android:layout_width=”fill_parent” >
<TableRow>
<TextView
android:text=”User Name:”
android:width =”120dp”/>
<EditText
android:id=”@+id/txtUserName”
android:width=”200dp” />
                              </TableRow>

<TableRow>
<TextView
android:text=”Password:”
/>
<EditText
android:id=”@+id/txtPassword”
android:password=”true”/>
                 </TableRow>

<TableRow>
<TextView />
<CheckBox android:id=”@+id/chkRememberPassword”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Remember Password”/>
   </TableRow>

<TableRow>
<Button
android:id=”@+id/buttonSignIn”
android:text=”Log In” />
      </TableRow>
</TableLayout>
```

3. Linear Layout:

The Linear Layout arranges views in a single column or a single row. Child views can be arranged either vertically or horizontally.

Linear Layout Attributes:

android:id
This is the ID which uniquely identifies the layout.
android:baselineAligned
This must be a boolean value, either "true" or "false".
android:baselineAlignedChildIndex
When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align.
android:divider
This is drawable to use as a vertical divider between buttons.
android:gravity
This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center etc.
android:orientation
This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.
android:weightSum
Sum up of child weight

Example:

Consider the following elements typically contained in the main.xml file:

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <Button android:id="@+id/btnStartService"
    android:layout_width="270dp"
    android:layout_height="wrap_content"
    android:text="start_service"/>

  <Button android:id="@+id/btnPauseService"
    android:layout_width="270dp"
    android:layout_height="wrap_content"
    android:text="pause_service"/>

  <Button android:id="@+id/btnStopService"
    android:layout_width="270dp"
    android:layout_height="wrap_content"
    android:text="stop_service"/>

</LinearLayout>
```

5. What are Views? Detail Explanation?

View:

An activity contains views and ViewGroups. A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and textboxes There are following View Groups:

➤ **Basic views** — Commonly used views such as the TextView, EditText, and Button views.

➤ **Picker views** — Views that enable users to select from a list, such as the TimePicker and DatePicker views.

➤ **List views** — Views that display a long list of items, such as the ListView and the SpinnerView views.

1. Basic Views:

➤ TextView

➤ EditText

➤ Button

➤ ImageButton

➤ CheckBox

➤ ToggleButton

➤ RadioButton

➤ RadioGroup

Definitions:

➤ **TextView** — TextView is commonly known as the label view. Its sole purpose is to display text on the screen.

➤ **Button** — Represents a push-button widget.
➤ **ImageButton** — Similar to the Button view, except that it also displays an image
➤ **EditText** — A subclass of the TextView view that allows users to edit its text content
➤ **CheckBox** — A special type of button that has two states: checked or unchecked
➤ **RadioGroup and RadioButton** — The RadioButton has two states: either checked or unchecked.
A **RadioGroup** is used to group together one or more RadioButton views, thereby allowing only one RadioButton to be checked within the RadioGroup.
➤ **ToggleButton** — Displays checked/unchecked states using a light indicator

2. Picker Views:

Selecting the date and time is one of the common tasks you need to perform in a mobile application. Android supports this functionality through the **TimePicker** and **DatePicker** views.

➤ **TimePicker View** — The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.

➤ **DatePicker View** — Another view that is similar to the TimePicker is the DatePicker. Using the DatePicker, you can enable users to select a particular date on the activity.

3. List Views:

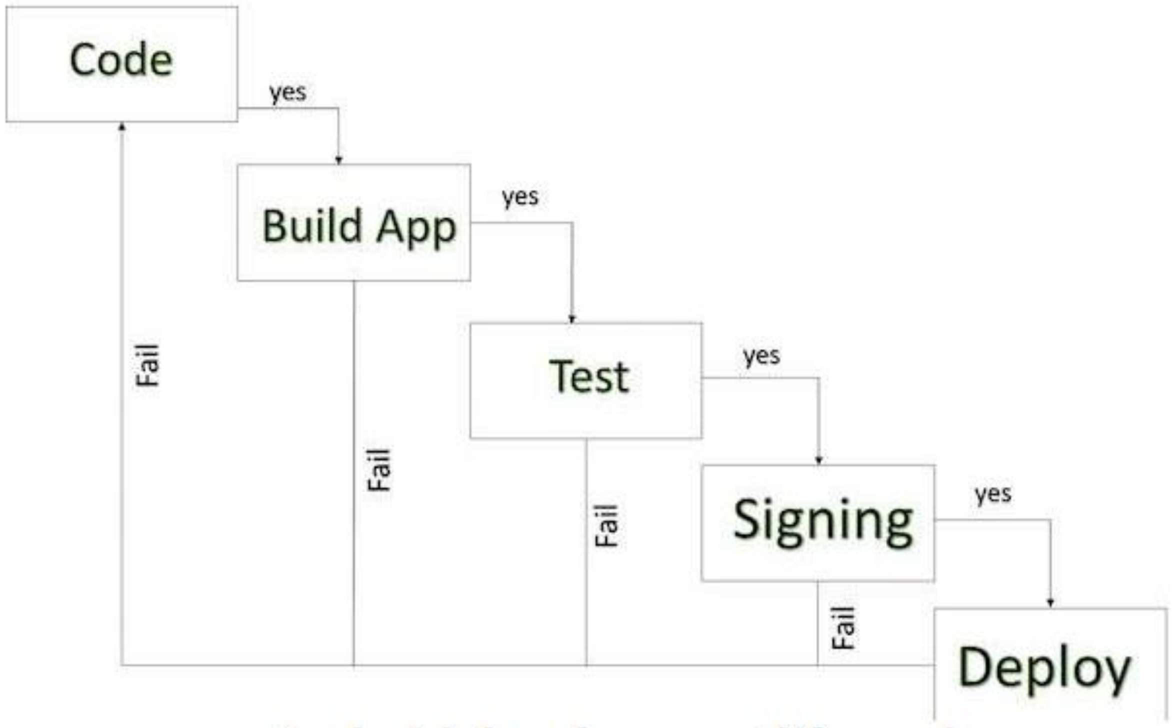
List views are views that enable you to display a long list of items. In Android, there are two types of list views: **ListView** and **SpinnerView**.

➤ **Listview View** — The ListView displays a list of items in a vertically scrolling list.

➤ **Spinner View** — The ListView displays a long list of items in an activity, but sometimes you may want your user interface to display other views, and hence you do not have the additional space for a full-screen view like the ListView. In such cases, you should use the SpinnerView. The SpinnerView displays one item at a time from a list and enables users to choose among them.

6. How to Publish Android Application? Explain its four steps?

Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.



Android development life cycle

Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

Here is a simplified check list which will help you in launching your Android application –

- Regression Testing
- Application Rating
- Targeted Regions
- Application Size
- SDK & Screen Compatibility
- Application Pricing
- Promotional Content
- Build and Upload release-ready APK
- Finalize Application Detail

Export Your Application As an APK

Before exporting the apps, you must some of tools

Dx tools(Dalvik executable tools): It going to convert .class file to .dex file. it has useful for memory optimization and reduce the boot-up speed time

AAPT(Android assistance packaging tool):it has useful to convert .Dex file to.Apk

APK(Android packaging kit): The final stage of deployment process is called as .apk.

1) Exporting and Signing Your Android Application:

1. Select the LBS project in Eclipse and then select File ➪ Export. . . .

2. In the Export dialog, expand the Android item and select Export Android Application and Click Next.

3. The LBS project should now be displayed. Click Next.

4. Select the "Create new keystore" option to create a new certificate (keystore) for signing your application. Enter a path to save your new keystore and then enter a password to protect the keystore.

5. Provide an alias for the private key (name it DistributionKeyStoreAlias;) and enter a password to protect the private key. Finally, enter your name in the field labeled First and Last Name. Click Next.

6. Enter a path to store the destination APK file. Click Finish. The APK fi le will now be generated.

7. LBS application requires the use of the Google Maps API key, which you applied by using your debug.keystore's MD5 fingerprint. Because you are

now generating your new keystore to sign your application for deployment, you need to apply for the Google Maps API key again, using the new keystore's MD5 fingerprint.
8. Using the MD5 fingerprint obtained from the previous step, go to google maps api sign up page and sign up for a new Maps API key.
9. Enter the new Maps API key in the main.xml file.
10. With the new Maps API key entered in the main.xml file, you now need to export the application once more and resign it.
11. Select the "Use existing key" option and enter the password you set earlier to secure the private key (enter key password). Click Next.
12. Click Finish to generate the APK file again. That's it! The APK is now generated and contains the new Map API key that is tied to the new keystore.

2) Deploying APK Files

After you have signed your APK fi les, you need a way to get them onto your users' devices. The following sections describe the various ways to deploy your APK fi les. Three methods are covered:

➤ Deploying manually using the adb.exe tool

➤ Hosting the application on a web server

➤ Publishing through the Android Market

Besides these methods, you can install your applications on users' devices using e-mail, an SD card, and so on. As long as you can transfer the APK fi le onto the user's device, the application can be installed.

1) Using the adb.exe Tool

Once your Android application is signed, you can deploy it to emulators and devices using the adb.exe (Android Debug Bridge) tool (located in the platform-tools folder of the Android SDK).

Using the command prompt in Windows, navigate to the <Android_SDK>\platform-tools folder.
To install the application to an emulator/device (assuming the emulator is currently up and running or a device is currently connected), issue the following command:

```
adb install "C:\Users\Wei-Meng Lee\Desktop\LBS.apk"
```

Besides using the adb.exe tool to install applications, you can also use it to remove an installed application. To do so, use the uninstall option to remove an application from its installed folder:

```
adb uninstall net.learn2develop.LBS
```

2) Using a Web Server

If you wish to host your application on your own, you can use a web server to do that. This is ideal if you have your own web hosting services and want to provide the application free of charge to your users.

To demonstrate this, I use the Internet Information Server (IIS) on my Windows 7 computer. Copy the signed LBS.apk file to c:\inetpub\wwwroot\.
In addition, create a new HTML file named index.html with the following content:

```
<html>
<title>Where Am I application</title>
<body>
Download the Where Am I application <a href="LBS.apk">here</a>
</body>
</html>
```

On your web server, you may need to register a new MIME type for the APK file. The MIME type for the .apk extension is application/vnd.android.package-archive.

To install the LBS.apk application from the IIS web server running on your computer, launch the Browser application on the Android emulator/device and navigate to the URL pointing to the APK file. To refer to the computer running the emulator, you should use the computer's IP address. Clicking the “here” link will download the APK fi le onto your device. Click the status bar at the top of the screen to reveal the download’s status.

To install the downloaded application, simply tap on it. It will show the permission(s) required by the application. Click the Install button to proceed with the installation. When the application is installed, you can launch it by clicking the Open button.

3) Publishing On Android Market:

So far, you have learned how to package your Android application and distribute it in various ways — via web server, the adb.exe fi le, e-mail, and SD card.
A better way is to host your application on the Android Market, a Google-hosted service that makes it very easy for users to discover and download applications for their Android devices. Users simply need to launch the Market application on their Android device in order to discover apps.
In this section, you will learn how to publish your Android application on the Android Market. You will walk through each of the steps involved, including the various items you need in order to prepare your application for submission to the Android Market.

- **Creating a Developer Profile**
  The first step toward publishing on the Android Market is to create a developer profile at http:// market.android.com/publish/Home.
  For this, you need a Google account (such as your Gmail account). Once you have logged in to the Android Market, you first create your developer profile.
  This will require Developer Name, Email Address, Website URL, Phone Number. Click Continue after entering the required information.

  For publishing on the Android Market, you need to pay a one-time registration fee, currently U.S. $25. Click the Google Checkout button to be redirected to a page where you can pay the registration fee. After paying, click the Continue link.

Next, you need to agree to the Android Market Developer Distribution Agreement. Check the “I agree” checkbox and then click the “I agree. Continue” link.

- **Submitting Your Apps**
  After you have set up your profile, you are ready to submit your application to the Android Market. If you intend to charge for your application, click the Setup Merchant Account link located at the bottom of the screen. Here you enter additional information such as bank account and tax ID.
  For free applications, click the Upload Application link.

  You will be asked to supply some information about your application. A dialog box will show set of details you need to provide.

  1) **Among the information needed, the following are compulsory:**

  ➤ The application must be in APK format

  ➤ You need to provide at least two screenshots. You can use the DDMS perspective in Eclipse to capture screenshots of your application running on the emulator or real device.

  ➤ You need to provide a high-resolution application icon. This size of this image must be 512 _ 512 pixels.

  2) **The other information details are optional, and you can always supply them later.**

  In particular, note that based on the APK file that you have uploaded, users are warned about any specific permissions required, and your application’s features are used to filter search results. For example, because my application requires GPS access, it will not appear in the search result list if a user searches for my application on a device that does not have a GPS receiver.

  3) **AFTER THIS**

  The next set of information you need to supply, includes the title of your application, its description, as well as details about recent changes (useful for application updates). You can also select the application type and the category in which it will appear in the Android Market.

4) LAST DIALOG

In the last dialog, you indicate whether your application employs copy protection, and specify a content rating. You also supply your website URL and your contact information. When you have given your consent to the two guidelines and agreements, click Publish to publish your application on the Android Market.

5) FINAL

That's it. Your application is now available on the Android Market. You will be able to monitor any comments submitted about your application, as well as bug reports and total number of downloads.
Good luck! All you need to do now is wait for the good news; and hopefully you can laugh your way to the bank soon!

7. Write a detail note on screen orientation with XML Code?

ADAPTING TO DISPLAY ORIENTATION

One of the key features of modern smartphones is their ability to switch screen orientation, and Android is no exception. Android supports two screen orientations: **portrait and landscape.**

Portrait format refers to a vertical orientation or a canvas taller than it is wide.
Landscape usually involves subjects that are too wide to shoot with a portrait orientation, and so, you have to turn the camera sideways, and shoot horizontally.

By default, when you change the display orientation of your Android device, the current activity that is displayed automatically redraws its content in the new orientation. This is because the onCreate() method of the activity is fired whenever there is a change in display orientation.

In general, you can employ two techniques to handle changes in screen orientation:

➤ **Anchoring —** The easiest way is to “anchor” your views to the four edges of the screen. When the screen orientation changes, the views can anchor neatly to the edges.
➤ **Resizing and repositioning —** Whereas anchoring and centralizing are simple techniques to ensure that views can handle changes in screen orientation, the ultimate technique is resizing each and every view according to the current screen orientation.

1) Anchoring Views

Anchoring can be easily achieved by using RelativeLayout. Consider the following main.xml file, which contains five Button views embedded within the <RelativeLayout> element:

```
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Note the following attributes found in the various Button views:

➤ **layout_alignParentLeft —** Aligns the view to the left of the parent view

➤ **layout_alignParentRight —** Aligns the view to the right of the parent view

➤ **layout_alignParentTop —** Aligns the view to the top of the parent view

➤ **layout_alignParentBottom —** Aligns the view to the bottom of the parent view

➤ **layout_centerVertical —** Centers the view vertically within its parent view.

➤ **layout_centerHorizontal —** Centers the view horizontally within its parent view

When the screen orientation changes to landscape mode, the four buttons are aligned to the four edges of the screen, and the center button is centered in the middle of the screen with its width fully stretched.

2) Resizing and Repositioning

Apart from anchoring your views to the four edges of the screen, an easier way to customize the UI based on screen orientation is to create a separate res/layout folder containing the XML fi les for the UI of each orientation. To support landscape mode, you can create a new folder in the res folder and name it as layout-land (representing landscape).

Basically, the main.xml fi le contained within the layout folder defines the UI for the activity in portrait mode, whereas the main.xml fi le in the layout-land folder defines the UI in landscape mode.

The following shows the content of main.xml under the layout-land folder (the statements in bold are the additional views to display in landscape mode):

FOR LANDSCAPE MODE

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top Right"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true" />
    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Left"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom Right"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true" />
    <Button
        android:id="@+id/button5"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Middle"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
    <Button
        android:id="@+id/button6"
        android:layout_width="180px"

                android:layout_height="wrap_content"
                android:text="Top Middle"
                android:layout_centerVertical="true"
                android:layout_centerHorizontal="true"
                android:layout_alignParentTop="true" />
            <Button
                android:id="@+id/button7"
                android:layout_width="180px"
                android:layout_height="wrap_content"
                android:text="Bottom Middle"
                android:layout_centerVertical="true"
                android:layout_centerHorizontal="true"
                android:layout_alignParentBottom="true" />
        </RelativeLayout>
```

8. Data Saving? Explain Three Ways to save data?

Persisting data is an important topic in application development, as users typically expect to reuse data in the future.
For Android, there are primarily three basic ways of persisting data:

- Shared Preferences
- Internal Storage
- External Storage (SD Card)

1) Shared Preferences

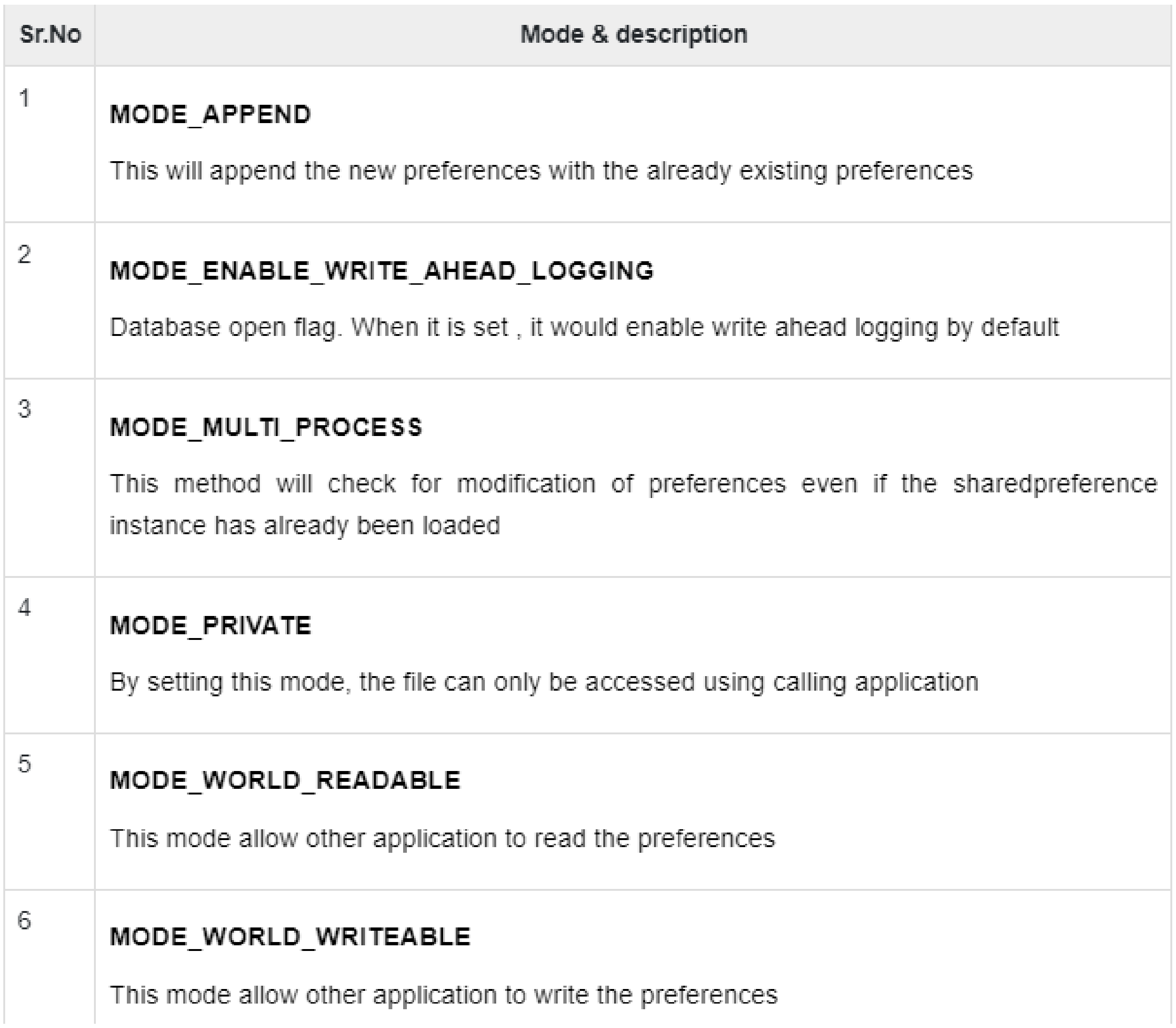
Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of key,value pair.

Using the SharedPreferences object, you save the data you want through the use of name/value pairs — specify a name for the data you want to save, and then both it and its value will be saved automatically to an XML file for you.

In order to use shared preferences, you have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

```
SharedPreferences sharedpreferences =
getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);
```

The first parameter is the key and the second parameter is the MODE. Apart from private there are other modes of SharedPreferences available that are listed below –



Sr.No	Mode & description
1	**MODE_APPEND** This will append the new preferences with the already existing preferences
2	**MODE_ENABLE_WRITE_AHEAD_LOGGING** Database open flag. When it is set , it would enable write ahead logging by default
3	**MODE_MULTI_PROCESS** This method will check for modification of preferences even if the sharedpreference instance has already been loaded
4	**MODE_PRIVATE** By setting this mode, the file can only be accessed using calling application
5	**MODE_WORLD_READABLE** This mode allow other application to read the preferences
6	**MODE_WORLD_WRITEABLE** This mode allow other application to write the preferences

You can save something in the sharedpreferences by using SharedPreferences.Editor class. You will call the edit method of SharedPreference instance and will receive it in an editor object. Its syntax is −

```
Editor editor = sharedpreferences.edit();
editor.putString("key", "value");
editor.commit();
```

2) Internal Storage

In this section we are going to look at the internal storage. Internal storage is the storage of the private data on the device memory.

By default these files are private and are accessed by only your application and get deleted, when user delete your application.

Writing file

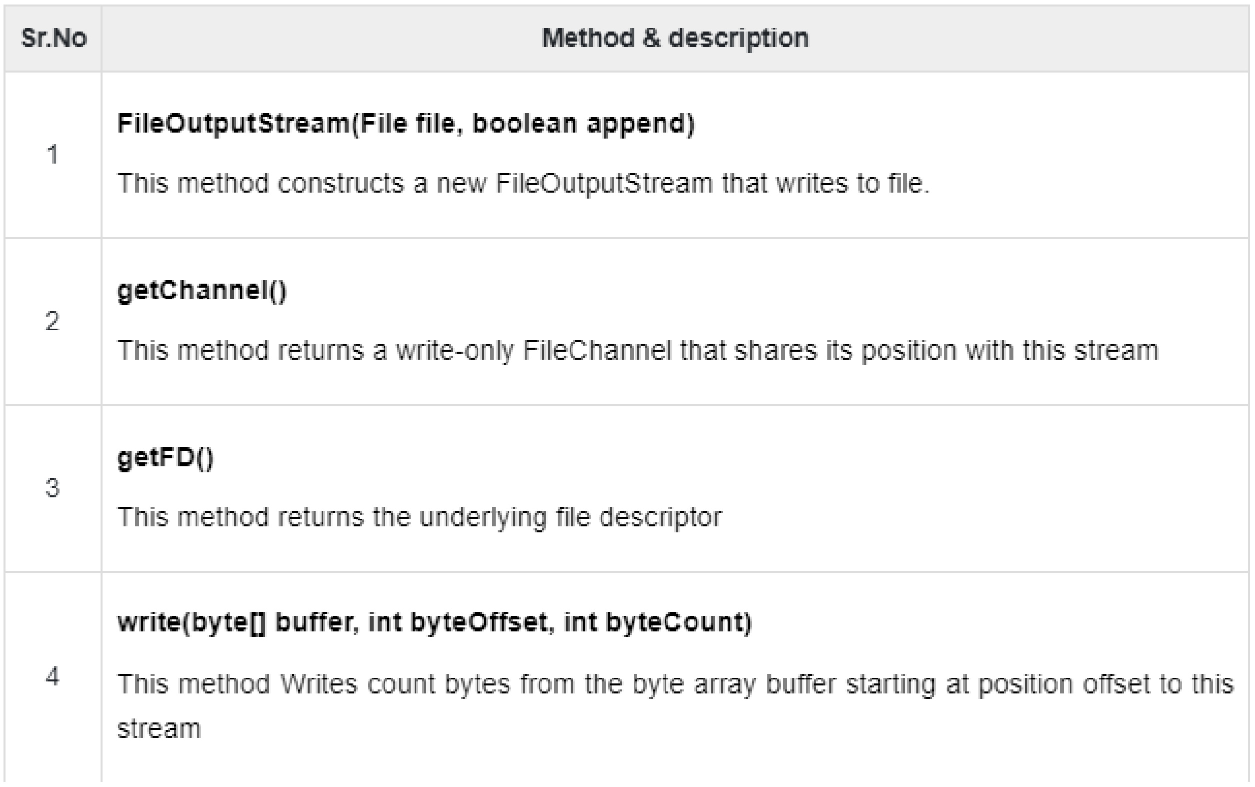
In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode. In this example, you used the MODE_WORLD_READABLE constant to indicate that the fi le is readable by all other applications. Mode can be private, append, and mode_world_writable.

```
FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```

The method openFileOutput() returns an instance of FileOutputStream. So you receive it in the object of FileInputStream. After that you can call write method to write data on the file. Its syntax is given below –

```
String str = "data";
fOut.write(str.getBytes());
fOut.close();
```

Apart from the the methods of write and close, there are other methods provided by the FileOutputStream class for better writing files. These methods are listed below –



Sr.No	Method & description
1	**FileOutputStream(File file, boolean append)** This method constructs a new FileOutputStream that writes to file.
2	**getChannel()** This method returns a write-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**write(byte[] buffer, int byteOffset, int byteCount)** This method Writes count bytes from the byte array buffer starting at position offset to this stream

Reading Files

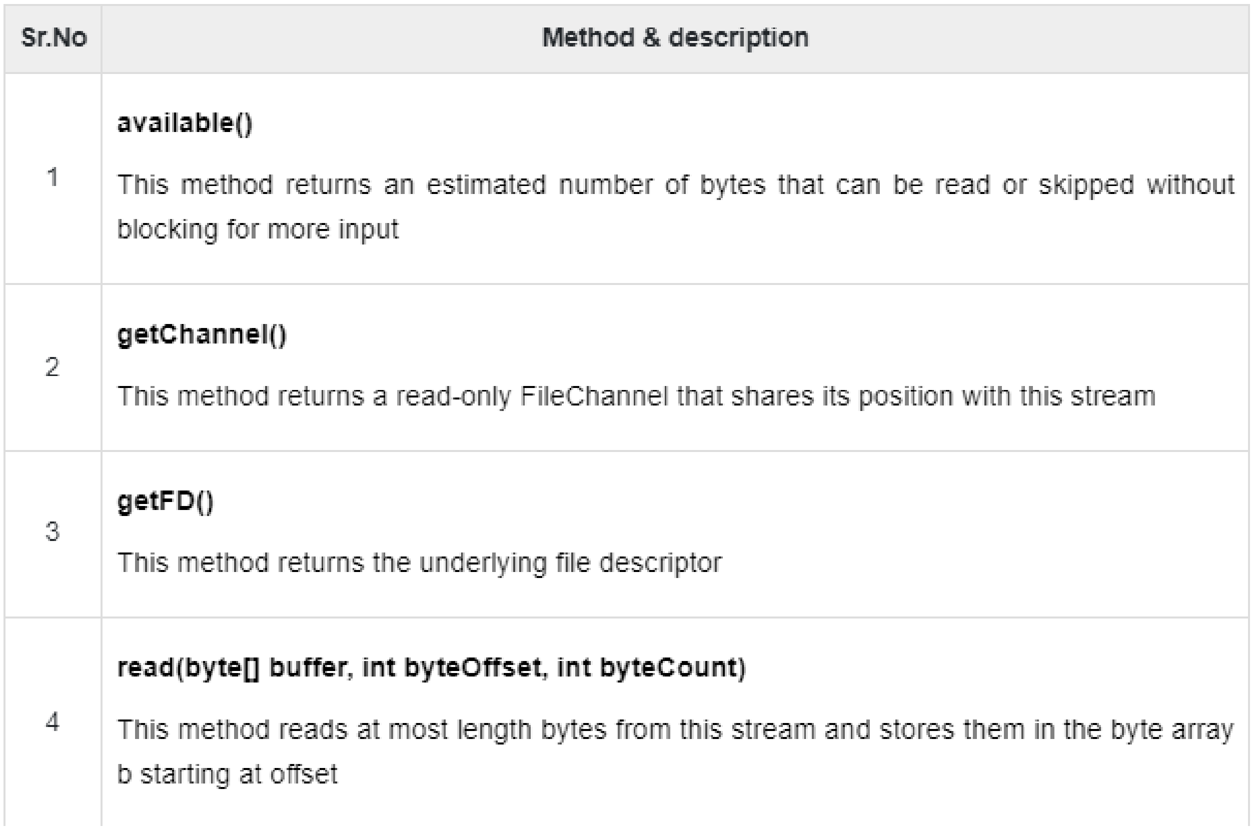
In order to read from the file you just created , call the openFileInput() method with the name of the file. It returns an instance of FileInputStream. Its syntax is given below –

```
FileInputStream fin = openFileInput(file);
```

After that, you can call read method to read one character at a time from the file and then you can print it. Its syntax is given below –

```
int c;
String temp="";
while( (c = fin.read()) != -1){
  temp = temp + Character.toString((char)c);
}
fin.close();
```

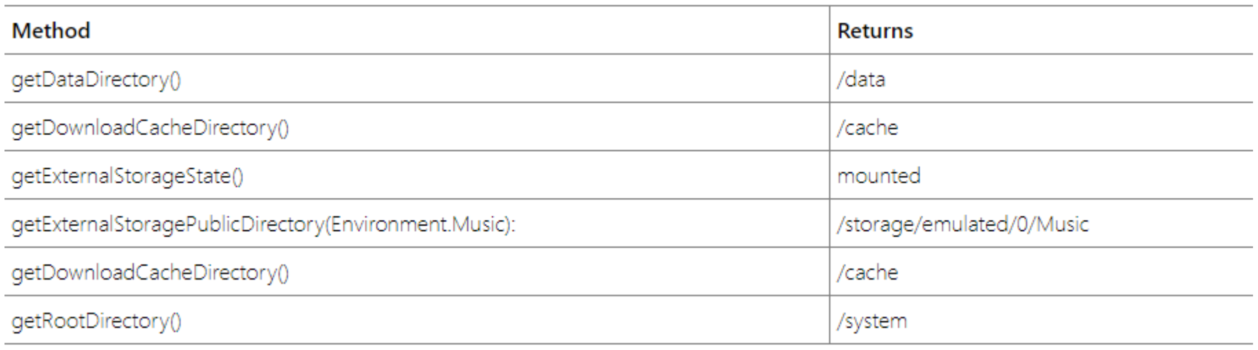
Apart from the the methods of read and close, there are other methods provided by the FileInputStream class for better reading files. These methods are listed below –



Sr.No	Method & description
1	**available()** This method returns an estimated number of bytes that can be read or skipped without blocking for more input
2	**getChannel()** This method returns a read-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**read(byte[] buffer, int byteOffset, int byteCount)** This method reads at most length bytes from this stream and stores them in the byte array b starting at offset

3) External Storage

The previous section showed how you can save your files to the internal storage of your Android device. Sometimes, it would be useful to save them to external storage (such as an SD card) because of its larger capacity, as well as the capability to share the fi les easily with other users (by removing the SD card and passing it to somebody else). Table of methods:



Method	Returns
getDataDirectory()	/data
getDownloadCacheDirectory()	/cache
getExternalStorageState()	mounted
getExternalStoragePublicDirectory(Environment.Music):	/storage/emulated/0/Music
getDownloadCacheDirectory()	/cache
getRootDirectory()	/system

Grant Access to External Storage

To read or write files on the external storage, our app must acquire the **WRITE_EXTERNAL_STORAGE** and **READ_EXTERNAL_STORAGE** system permissions. For that, we need to add the following permissions in the android manifest file like as shown below.

```
<manifest>

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STO
RAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STOR
AGE"/>

</manifest>
```

Checking External Storage Availability

Before we do any work with external storage, first we need to check whether the media is available or not by calling **getExternalStorageState()**. The media might be mounted to a computer, missing, read-only or in some other state. To get the media status, we need to write the code like as shown below.

Write a File to External Storage

By using android FileOutputStream object and getExternalStoragePublicDirectory method, we can easily create and write data to the file in external storage public folders.

Following is the code snippet to create and write a public file in the device Downloads folder.

we are creating and writing a file in device public Downloads folder by using getExternalStoragePublicDirectory method. We used write() method to write the data in file and used close() method to close the stream.

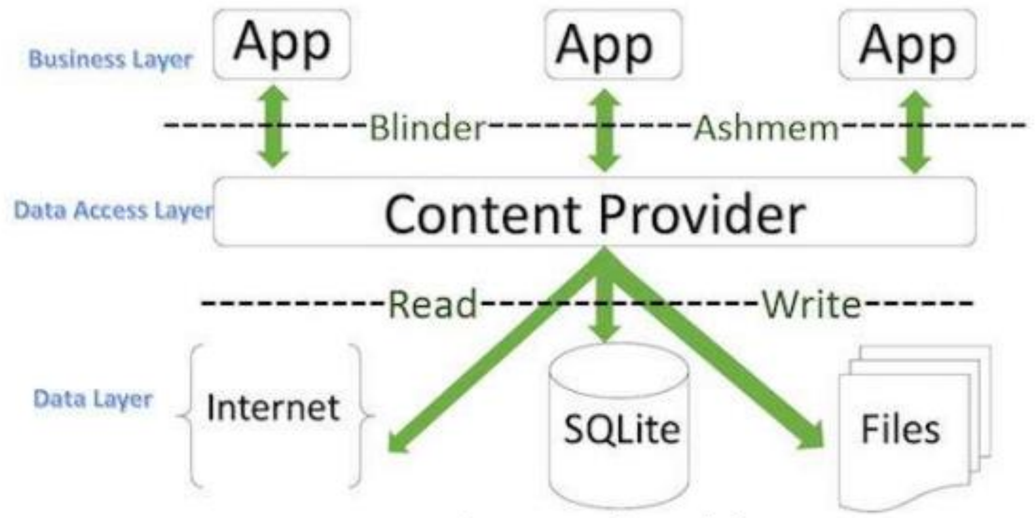
Read a File from External Storage

By using the android FileInputStream object and getExternalStoragePublicDirectory method, we can easily read the file from external storage.

9. Write a note on Content Providers?

A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data. However, content providers are primarily intended to be used by other applications, which access the provider using a provider client object.

A content provider behaves very much like a database — you can query it, edit its content, as well as add or delete content. However, unlike a database, a content provider can use different ways to store its data. The data can be stored in a database, in files, or even over a network.



ContentProvider

Android ships with many useful content providers, including the following:

➤ **Browser —** Stores data such as browser bookmarks, browser history, and so on

➤ **CallLog —** Stores data such as missed calls, call details, and so on
➤ **Contacts —** Stores contact details
➤ **MediaStore —** Stores media fi les such as audio, video, and images
➤ **Settings —** Stores the device's settings and preferences

Besides the many built-in content providers, you can also create your own content providers.
To query a content provider, you specify the query string in the form of a URI, with an optional specifier for a particular row. The format of the query URI is as follows:

<standard_prefix>://<authority>/<data_path>/<id>

The various parts of the URI are as follows:

➤ The **standard prefix** for content providers is always content://.
➤ The **authority** specifies the name of the content provider. An example would be contacts for the built-in Contacts content provider. For third-party content providers, this could be the fully qualified name, such as com.wrox.provider or net.learn2develop.provider.
➤ The **data path** specifies the kind of data requested. For example, if you are getting all the contacts from the Contacts content provider, then the data path would be people, and the URI would look like this: content://contacts/people.
➤ The **id** specifies the specific record requested. For example, if you are looking for contact number 2 in the Contacts content provider, the URI would look like this:
content:// contacts/people/2.

Create Content Provider

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the ContentProviderbaseclass.
- Second, you need to define your content provider URI address which will be used to access the content.

- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override onCreate() method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the onCreate() handler of each of its Content Providers is called on the main application thread.
- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using <provider> tag.

Methods of Content Provider:

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working –

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.