Document of Understanding Android

Version 0.1

28-Aug-2018

Sourav Kumar Verma

Integra Micro Software Services Pvt. Ltd. (IMSSPL)

No. 4, 1st Floor, Bellary Road,

12th KM, Jakkur,

Bangalore 560 064, INDIA

Tel: 080-28565801-5

Fax: 080-28565800

www.integramicroservices.com

Helping customers shorten the development cycle

Table of contents

Revision History

Version Number	Brief description of change	Reference for change	Affected Section(s)	Effective Date	Reviewed By	Reviewed Date
0.1	Initial draft version		All	28 Aug. 18		
0.2	Include the workshop section		Workshop Day1, Day2, Day3	12 Sep. 18	Murthy A N, Sudarshan Vadyar	17 Sep. 18

Introduction

This document describes how to install the Android software development kit (SDK) and all the related software you're likely to need and walk through with android basic **COMPONENTS.** By the end, you'll be able to run a program on an emulator/mobile device.

Windows, Mac OS X, and Linux systems can all be used for Android application development. We will load the software, introduce you to the tools in the SDK, and point you to sources of example code.

we refer to instructions available on various websites for installing and updating the tools you will use for creating Android programs. The most important place to find information and links to tools is the Android Developers site:

http://developer.android.com

A three-day program designed to enable Android application development on Mobile and Tablet platforms. During the workshop, you will be taken through the Android features and concepts. By end of the day, you will be aware of where to start and how to develop your own application.

Topics Covered

- Android Then & Now!!
- Understanding Android architecture
- Android Building Blocks –
- Debugging & Testing tools
- Android Installations & Configurations
- Writing an App - Live demo with a piece of code
- Launching App into Market space

Who should attend

- Professionals who want to upgrade their skills.
- Individuals who are planning to develop their own applications.
- Enthusiasts who wants to know how Android Apps are developed and published

Pre-requisites

- Basic Java knowledge

- Internet connection

Key Takeaways

- You will be able to Develop Android Applications by your Own!!
- Motivation to become ANDROID Expert!!

Android installation prerequisites:

Windows OS

1. Microsoft Windows 7/8/10 (32-bit or 64-bit)
2. 2 GB RAM minimum, 8 GB RAM recommended
3. 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
4. 1280 x 800 minimum screen resolution
5. JDK 8
6. For accelerated emulator: 64-bit operating system and Intel processor with support for Intel VT-x, Intel EM64T (Intel 64), and Execute Disable (XD) Bit functionality

Mac OS

1. Mac OS X 10.8.5 or higher, up to 10.11.4 (El Capitan)
2. 2 GB RAM minimum, 8 GB RAM recommended
3. 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
4. 1280 x 800 minimum screen resolution
5. JDK 8

Linux OS

1. GNOME or KDE desktop: Tested on Ubuntu 12.04, Precise Pangolin (64-bit distribution capable of running 32-bit applications)
2. 64-bit distribution capable of running 32-bit applications
3. GNU C Library (glibc) 2.11 or later
4. 2 GB RAM minimum, 8 GB RAM recommended
5. 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
6. 1280 x 800 minimum screen resolution
7. JDK 8
8. For accelerated emulator: Intel processor with support for Intel VT-x, Intel EM64T (Intel 64), and Execute Disable (XD) Bit functionality, or AMD processor with support for AMD Virtualization (AMD-V)

Download and Install JAVA

It is recommended, before you proceed with online installation you may want to disable your Internet firewall. In some cases, the default firewall settings are set to reject all automatic or online installations such as the Java online installation. If the firewall is not configured appropriately it may stall the download/install operation of Java under certain conditions. Refer to your specific Internet firewall manual for instructions on how to disable your Internet Firewall.

Note: Installing Java requires that you can gain administrator access to Windows on your computer.

Windows OS

- Go to the Manual download page
- (https://java.com/en/download/manual.jsp)
- Click on Windows Online
- The File Download dialog box appears prompting you to run or save the download file
- To run the installer, click Run.
- To save the file for later installation, click Save.
  Choose the folder location and save the file to your local system.
  Tip: Save the file to a known location on your computer, for example, to your desktop.
  Double-click on the saved file to start the installation process.
- The installation process starts. Click the Install button to accept the license terms and to continue with the installation.

- Oracle has partnered with companies that offer various products. The installer may present you with option to install these programs when you install Java. After ensuring that the desired programs are selected, click the Next button to continue the installation.
- A few brief dialogs confirm the last steps of the installation process; click Close on the last dialog. This will complete Java installation process.

Mac OS

- Download the jre-8u65-macosx-x64.pkg file.
  Review and agree to the terms of the license agreement before downloading the file.
- Double-click the .pkg file to launch it
- Double-click on the package icon to launch install Wizard

- The Install Wizard displays the Welcome to Java installation screen. Click Next

- After the installation has completed, a confirmation screen appears. Click Close to finish the installation process.

Linux OS

- Change to the directory in which you want to install. Type:
  cd **directory_path_name**
  For example, to install the software in the /usr/java/ directory, Type:
  cd /usr/java/
- Move the .tar.gz archive binary to the current directory.
- Unpack the tarball and install Java
  tar zxvf jre-8u73-linux-x64.tar.gz

  The Java files are installed in a directory called jre1.8.0_73 in the current directory. In this example, it is installed in the /usr/java/jre1.8.0_73 directory. When the installation has completed, you will see the word Done.
- Delete the .tar.gz file if you want to save disk space.

Download and Install Android Studio

Download Android Studio

Google provides Android Studio for the Windows, Mac OS X, and Linux platforms.

You can download this software from the Android Studio homepage.

Installing Android Studio

Installing Android Studio on 64-bit or 32-bit Windows 8.1

launched android-studio-bundle-143.2821654-windows.exe to start the installation process. The installer responded by presenting the Android Studio Setup dialog box shown in Figure 1.

Figure 1. Set up Android Studio

Clicking Next took me to the following dialog box, which gives you the option to decline installing the Android SDK (included with the installer) and an Android Virtual Device (AVD).

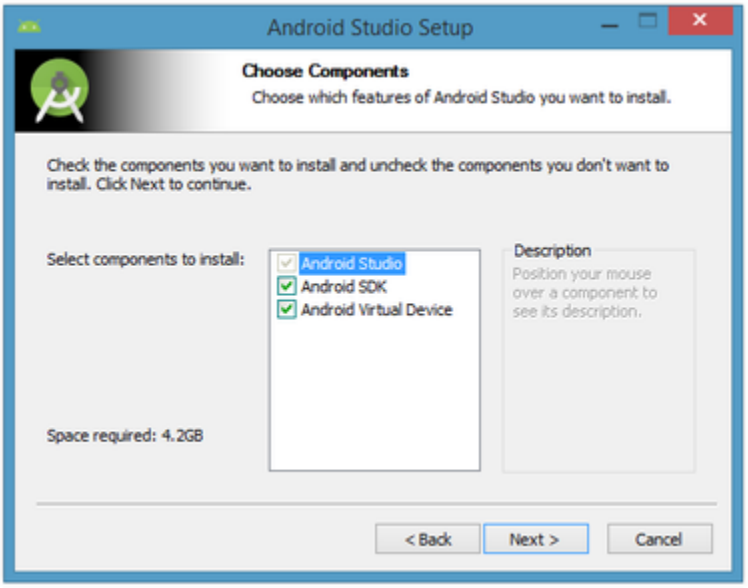


Figure 2. Do you want to install the Android SDK and AVD?

You chose to keep the default settings. After clicking Next, you'll be taken to the license agreement dialog box. Accept the license to continue the installation.

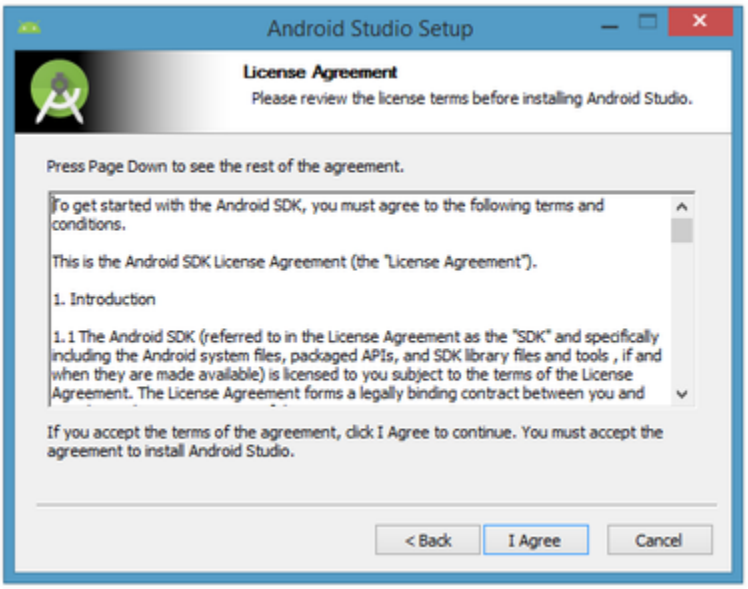


Figure 3. Accept the license agreement to continue installation

The next dialog box invites you to change the installation locations for Android Studio and the Android SDK.

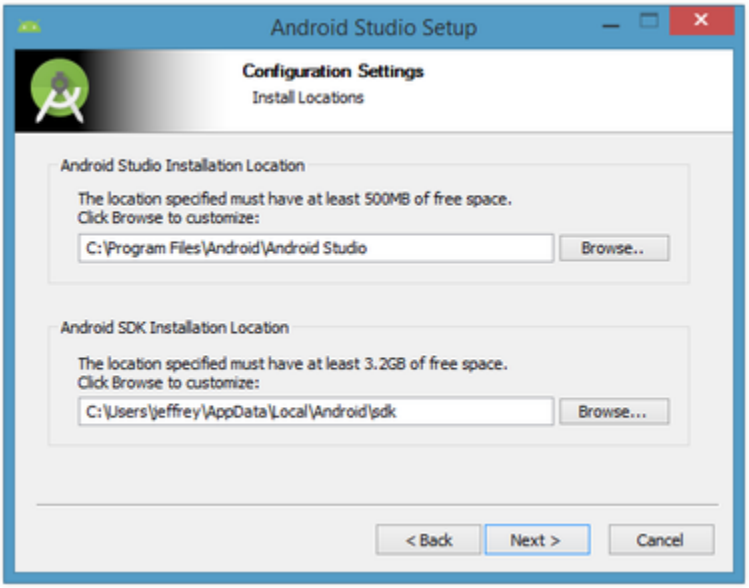


Figure 4. Set the Android Studio and Android SDK installation locations

Change the location or accept the default locations and click Next.

The installer defaults to creating a shortcut for launching this program, or you can choose to decline. I recommend that you create the shortcut, then click the Install button to begin installation.

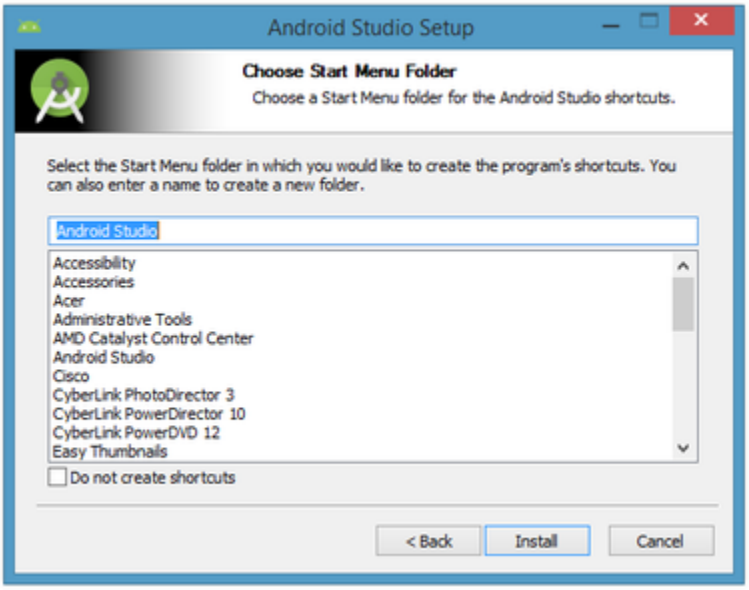


Figure 5. Create a new shortcut for Android Studio

The resulting dialog box shows the progress of installing Android Studio and the Android SDK. Clicking the Show Details button will let you view detailed information about the installation progress.

The dialog box will inform you when installation has finished. When you click Next, you should see the following:

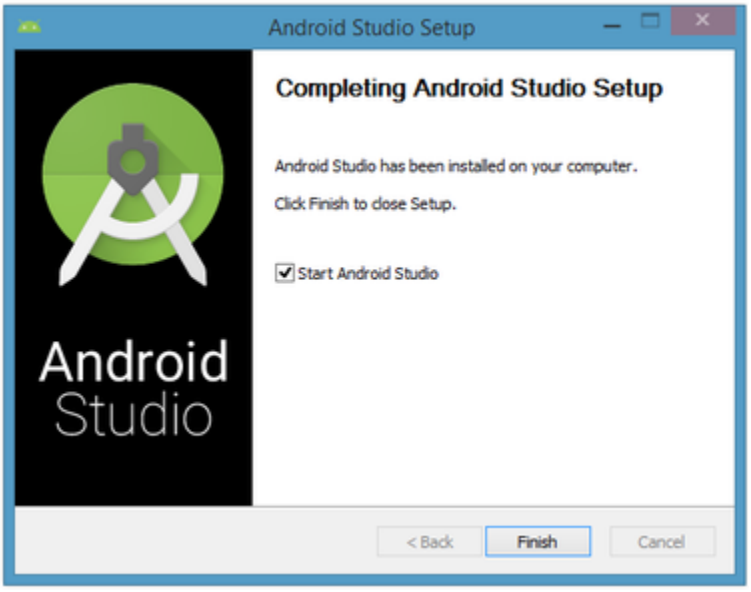


Figure 6. Leave the Start Android Studio check box checked to run this software

To complete your installation, leave the Start Android Studio box checked and click Finish.

Running Android Studio

Android Studio presents a splash screen when it starts running:

Figure 7. Android Studio's start screen

On your first run, you'll be asked to respond to several configuration-oriented dialog boxes. The first dialog box focuses on importing settings from any previously installed version of Android Studio.

Figure 8. Import settings

If you're like me, and don't have a previously installed version, you can just keep the default setting and click OK. Android Studio will respond with a slightly enhanced version of the splash screen, followed by the Android Studio Setup Wizard dialog box:

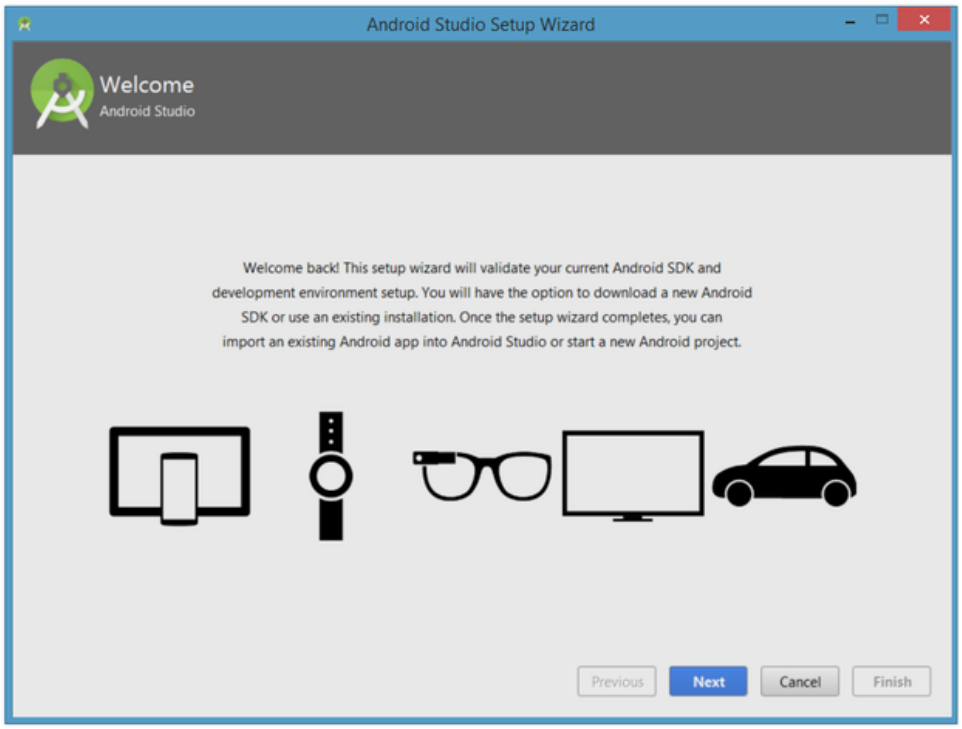


Figure 9. Validate your Android SDK and development environment setup

When you click Next, the setup wizard invites you to select an installation type for your SDK components. For now I recommend you keep the default standard setting.

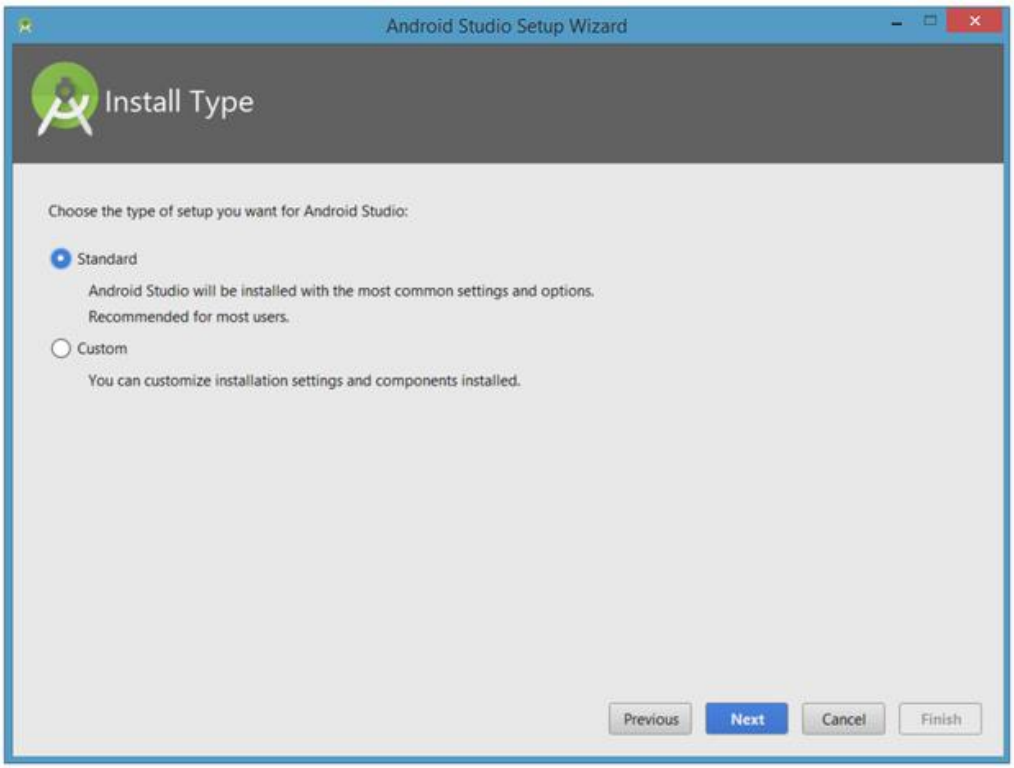


Figure 10. Choose an installation type

Click Next and verify your settings, then click Finish to continue.

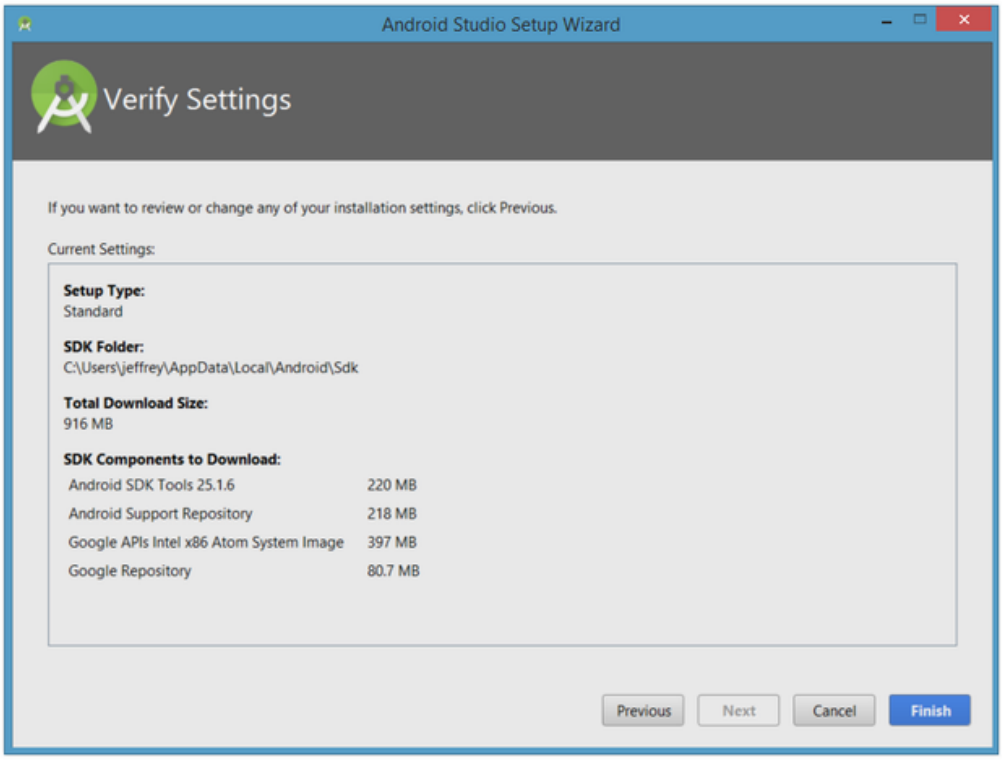


Figure 11. Review settings

The wizard will download and unzip various components. Click Show Details if you want to see more information about the archives being downloaded and their contents.

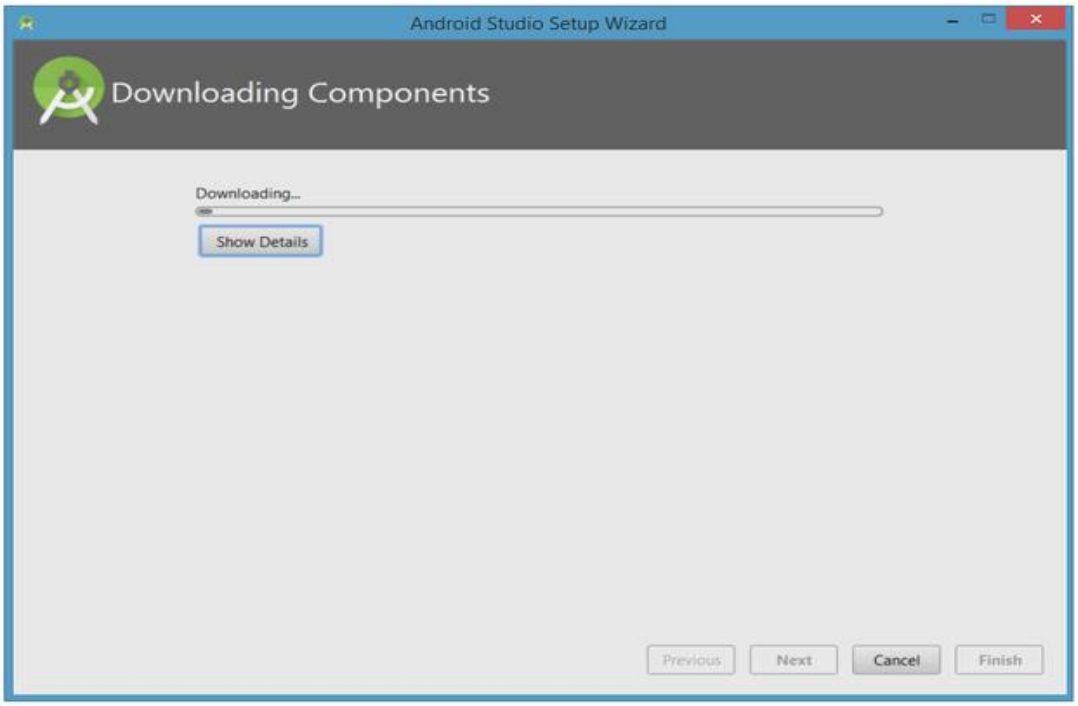


Figure 12. The wizard downloads and unzips Android Studio components

If your computer isn't Intel based, you might get an unpleasant surprise after the components have completely downloaded and unzipped:

```
m2repository/com/google/firebase/firebase-core/9.0
 .1/firebase-core-9.0.1.pom.md5
Android SDK is up to date.
Unable to install Intel HAXM
Your CPU does not support required features (VT-x or SVM).
Unfortunately, your computer does not support hardware
 accelerated virtualization.
Here are some of your options:
 1) Use a physical device for testing
 2) Develop on a Windows/OSX computer with an Intel processor
 that supports VT-x and NX
 3) Develop on a Linux computer that supports VT-x or SVM
 4) Use an Android Virtual Device based on an ARM system image
   (This is 10x slower than hardware accelerated
 virtualization)

Creating Android virtual device
Unable to create a virtual device: Unable to create Android
 virtual device
```

Figure 13. Intel-based hardware acceleration is unavailable

Your options are to either put up with the slow emulator or use an Android device to speed up development. I'll discuss the latter option later in the tutorial.

Finally, click Finish to complete the wizard. You should see the Welcome to Android Studio dialog box:

Figure 14. Welcome to Android Studio

You'll use this dialog to start up a new Android Studio project, work with an existing project, and more. You can access it anytime by double-clicking the Android Studio shortcut on your desktop.

Workshop

Workshop Schedule

Day	9:30 AM to 11:00 AM Introduction to concepts, aspects, topics	11:00 AM to 11:15 AM	11:15 AM to 12:45 PM Insights through code snippets	12:45 PM to 1:30 PM	1:30 PM to 4:30 PM Lab session
Day 1 Tuesday 19 Sep 2018	Android Architecture, Android app building blocks / components, Android project folder structure, Localization (supporting multiple languages in user interface screen)	Tea Break	Activity, Fragments Layout XML Android Manifest XML 'res' folder details	Lunch Break	Developing an app that has Activity / Fragments, using UI Layout XML and declaring activity in the Manifest XML file.
Day 2 Wednesday 20 Sep 2018	Android components - Service, Broadcast Receiver, Content Provider with local database and additional items - Asynctask, Intents		Service, Asynctask, Intents, Broadcast Receiver, Content Provider		Developing an app that uses the Service, Asynctask, Intents, Broadcast Receiver, Content Provider
Day 3 Thursday 21 Sep 2018	Additional items - Notifications, Alert dialogs, custom list view, about 3rd party libraries (Volley, Retrofit etc), About Android SDK Tools, About Push Notification		Local notification, Alert dialog, Custom list view		Develop an app having the Alert dialog, Local Notification

DAY1

Android architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.

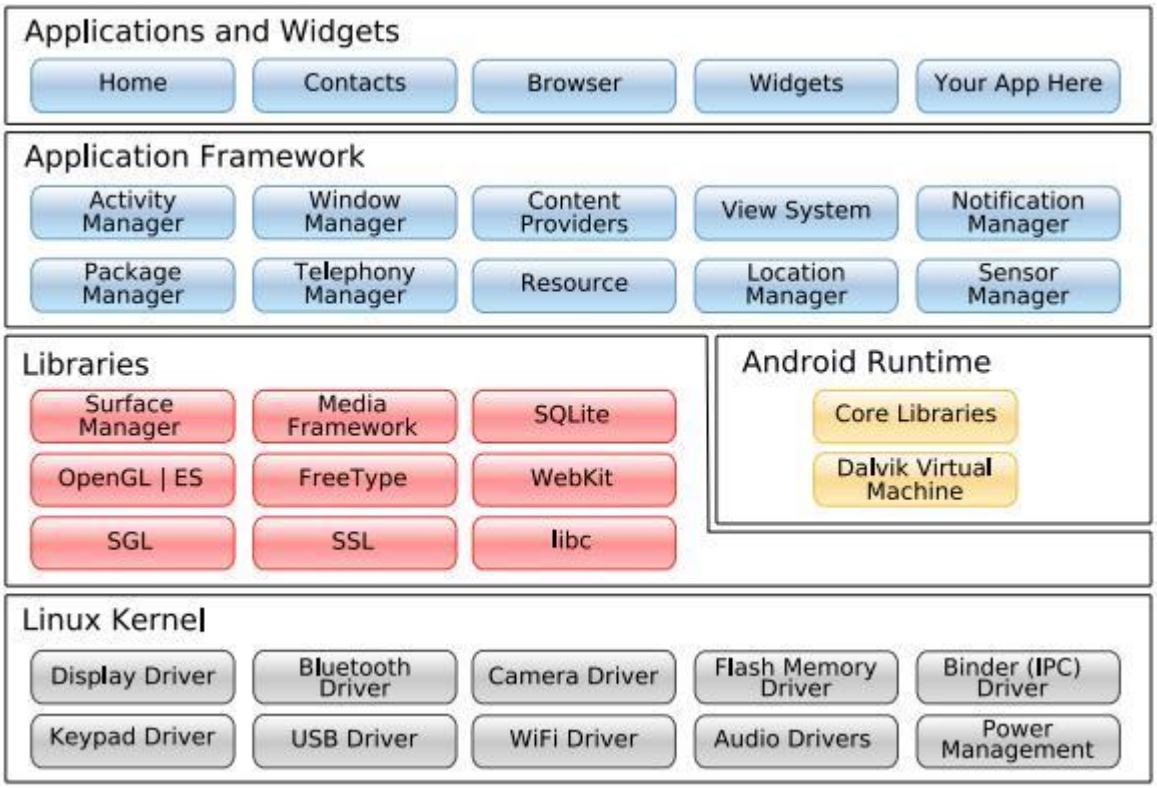


Figure 14. Android architecture digram

- **Linux kernel**

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

- **Libraries**

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for

storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

- **Android Libraries**

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows –

android.app	Provides access to the application model and is the cornerstone of all Android applications.
android.content	Facilitates content access, publishing and messaging between applications and application components.
android.database	Used to access data published by content providers and includes SQLite database management classes.
android.opengl	A Java interface to the OpenGL ES 3D graphics rendering API.
android.os	Provides applications with access to standard operating system services including messages, system services and inter-process communication.
android.text	Used to render and manipulate text on a device display.
android.view	The fundamental building blocks of application user interfaces.
android.widget	A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
android.webkit	A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

- **Android Runtime**

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Note: Android Lollipop, Google has replaced DVM with ART because ART is faster than DVM

- **Application Framework**

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services –

Activity Manager	Controls all aspects of the application lifecycle and activity stack.
Content Providers	Allows applications to publish and share data with other applications.
Resource Manager	Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
Notifications Manager	Allows applications to display alerts and notifications to the user.
View System	An extensible set of views used to create application user interfaces.

- **Applications**

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

Android Building Blocks (Android Components) - Introduction

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact.

There are following four main components that can be used within an Android application –

Activities	They dictate the UI and handle the user interaction to the smart phone screen.
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

- **Activities**

An activity represents a single screen with a user interface, in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of **Activity** class as follows –

```
public class MainActivity extends Activity {

}
```

- **Services**

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of **Service** class as follows –

```
public class MyService extends Service {

}
```

- **Broadcast Receivers**

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcaster as an **Intent** object.

```
public class MyReceiver extends BroadcastReceiver {

   public void onReceive(context,intent){}

}
```

- **Content Providers**

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolverclass. The data may be stored in the file system, the database or somewhere else entirely.

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

```
public class MyContentProvider extends  ContentProvider {

   public void onCreate(){}

}
```

We will go through these tags in detail while covering application components in individual chapters.

Android project folder structure

- **app folder**

Once created, the project files contain various folders and files about project settings as well as your Android Application module.

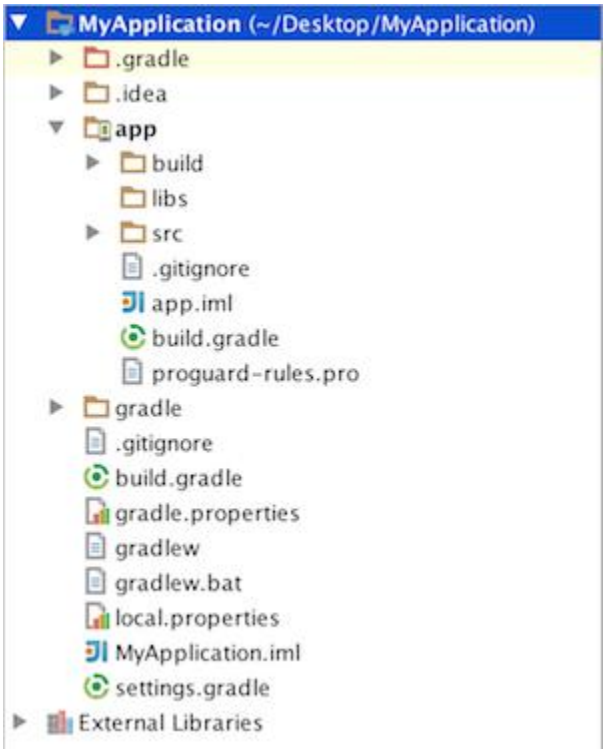


Figure 15. Top-level view of Android project in Android Studio

.idea directory	contains Intellij IDEA settings.
.gitignore file	specifies which files/directories to be ignored by Git
gradle	directory contains gradle-wrapper files.
build.gradle	allows you to customise properties for build system such as setting location of your keystore used for signing the app-release.apk.
settings.gradle	tells sub-projects for gradle to build.

- **Closer look inside your app folder**

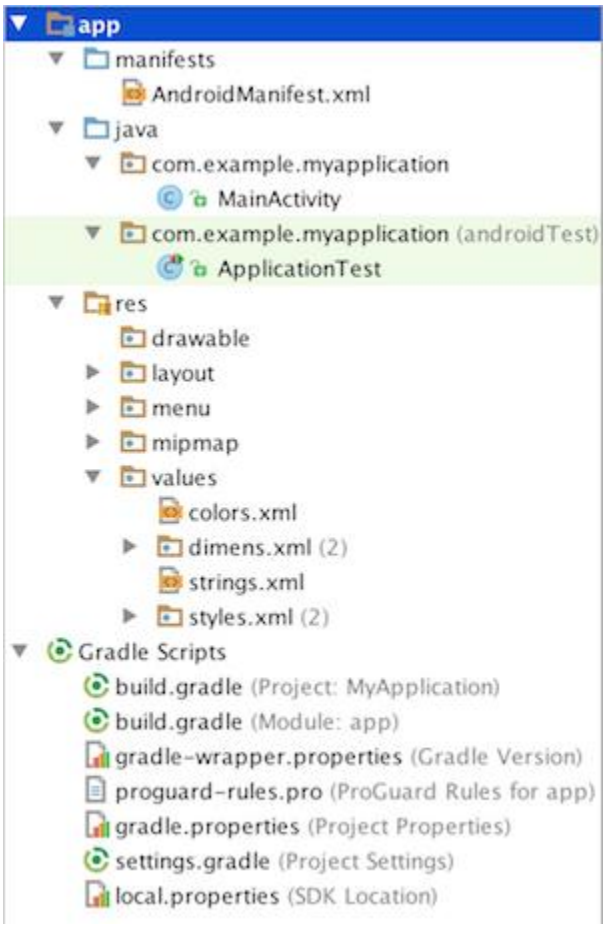


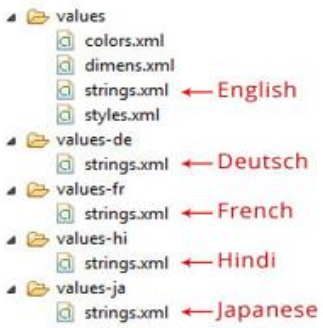
Figure 16. Application-level view of the project

AndroidManifest.xml	is a compulsory file to have in every Android application. It controls the nature of your application. For example, it allows you to set the name and icon of your application. It also describes activities and other components of your application, as well as permissions and libraries required to run your application.
java.<package>	contains the source codes for your application, including tests.
res/drawable	contains bitmap files (eg. png, jpeg), 9-Patch images (stretchable and repeatable images), and drawable shapes (xml).
res/layout contains	XML screen layout files that can be linked to Activities or other components of your application. The layout files are

	referenced by its filename, while the each component inside the layout can be referenced by their respective ids.
res/menu	contains XML files for menu application.
res/mipmap	contains your app-launcher icons (icon displayed on the home screen) for various screen resolutions.
res/values contains XML	value files for specifying colors, dimensions, strings, styles, etc. Besides for neat organization, classifying values under this directory is useful for reusability if many components of the application need to use the same value.
build.gradle (Module: app)	allows us to customise properties for build system, specific to the module app. It will override default build settings used by the manifest. You will usually add dependency libraries specific to your application here as well.

Localization

Android Multi-Language App



Locale Code	Language / Country	Location of strings.xml	Location of background image
Default	Default	res/values/	res/drawable/
en-rUS	English / US	res/values/	res/drawable-en-rUS/
it-rIT	Italian / Italy	res/values-it/	res/drawable-it-rIT/
fr-rFR	French / France	res/values-fr/	res/drawable-fr-rFR/
fr-rCA	French / Canada	res/values-fr/	res/drawable-fr-rCA/
en-rCA	English / Canada	res/values/	res/drawable-en-rCA/
ru-rRU	Russian / Russia	res/values-ru/	res/drawable-ru-rRU/

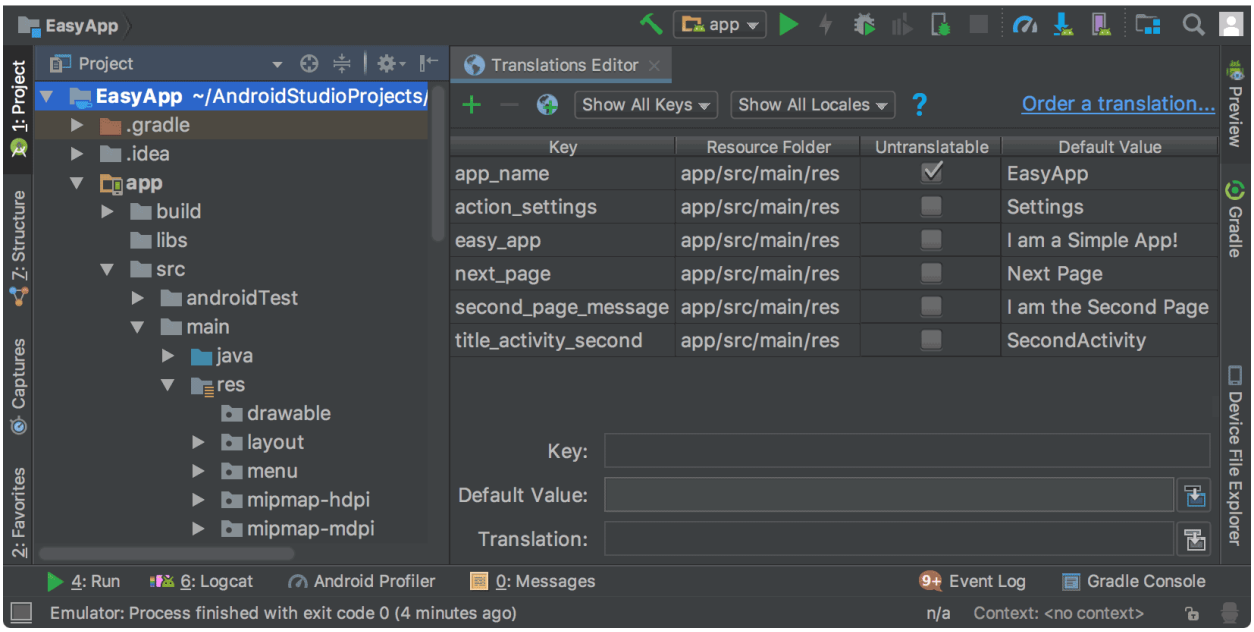


Figure 17. The Translations Editor showing app text before translation

Android Components

- **ACTIVITY**

- An activity represents a single screen with a user interface.
- Subclass of Activity-class
- One app may have one or several activities
- Each activity is given a default window to draw in
- Window consists of views (widgets)

- **Activity Stack:**
  - ✓ Android keeps navigation history of activities the user has visited: Activity Stack or the Back Stack
  - ✓ Pressing Back displays the previous Activity!
  - ✓ User cannot go further than the last visit of home

- **Back Stack**

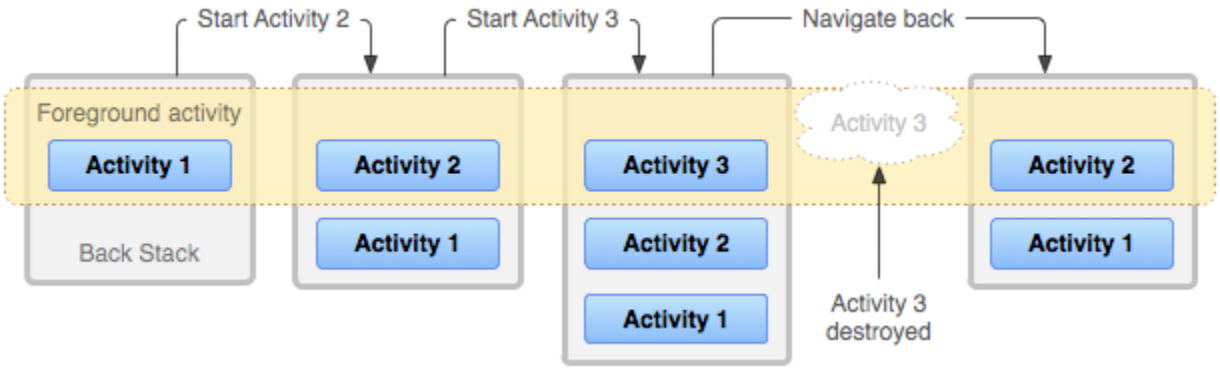


Figure 18. Activity Stack

- **Tasks**
  - ✓ Task is a sequence of activities
  - ✓ Task can hold activities from several apps
  - ✓ Activity that starts the task is called root activity -Usually started from home screen
  - ✓ New task is started when new app is launched. Also new task can be started on certain activities (opening browser, maps etc)
  - ✓ Recent task switcher shows the recent tasks

➢ Activity Lifecycle

```
public class MyActivity extends Activity {
        protected void onCreate(Bundle savedInstanceState);
        protected void onStart();
        protected void onRestart();
        protected void onResume();
        protected void onPause();
        protected void onStop();
        protected void onDestroy();
}
```

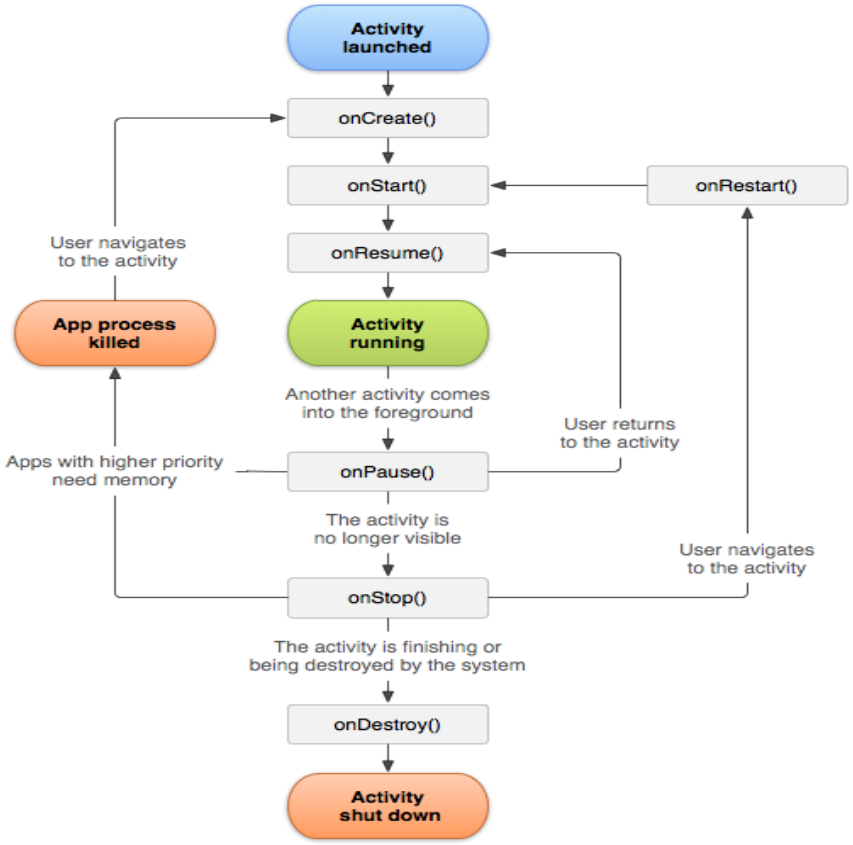


Figure 19. Activity Lifecycle

onCreate()	Create the user interface.
onStart()	When visible to user.
onResume()	Activity is visible again, initialize fields, register listeners, bind to services.
onPause()	Activity still partially visible, but most often is an indication that the user is leaving the activity and it will soon enter the stopped state. Release resources, save app data, unregister listeners, unbind services.
onStop()	Activity is no longer visible to user. Time or CPU intensive shut down operations like writing information to database.

- **Activity Visibility graph**

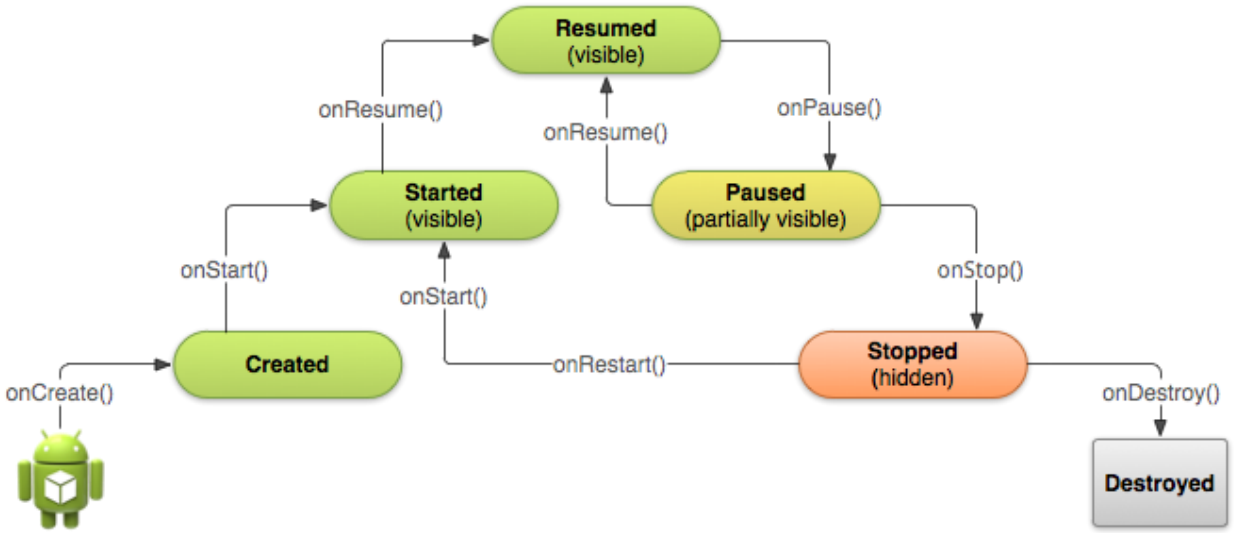


Figure 20. Activity visibility graph

- **State Information**

Think about following situation:

- ✓ User opens app and Activity 1 opens.
- ✓ User is prompt a name in EditText.
- ✓ User writes his/her name.

- ✓ User navigates from Activity 1 to Activity 2
- ✓ User presses back button and Activity 1 is opened again
- ✓ Is the name still there?

➢ **How to Store State Information**

- ✓ Store state: – onSaveInstanceState(Bundle)
- ✓ Read state – onRestoreInstanceState(Bundle)
- ✓ This will store data only temporarily: for app lifetime!
- ✓ Data will be held in memory until the app is closed!

- ✓ **Store**

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    String text = textfield.getText().toString();
    savedInstanceState.putString("someKey", text);
    super.onSaveInstanceState(savedInstanceState);
}
```

- ✓ **Load**

```
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    if (savedInstanceState != null) {
          String strValue = savedInstanceState.getString("someKey");
          if (strValue != null) {
             textfield.setText(strValue);
          }
      }
}
```

Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are –

Fragments	Represents a portion of user interface in an Activity.

Views	UI elements that are drawn on-screen including buttons, lists forms etc.
Layouts	View hierarchies that control screen format and appearance of the views.
Intents	Messages wiring components together.
Resources	External elements, such as strings, constants and drawable pictures.
Manifest	Configuration file for the application.

- **Intent**

An Android **Intent** are message passing mechanism that lets you declare your intentation than an action be performed with a particular piece of data

- ✓ startActivity to launch an Activity,
- ✓ broadcastIntent to send it to any interested BroadcastReceiver components,
- ✓ startService(Intent) or bindService (Intent, ServiceConnection, int) to communicate with a background Service.

- **Types of Intents**

Explicit intent: Explicit intent going to be connected internal world of application, suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent.

below image is connecting first activity to second activity by clicking button.

```
Intent i = new Intent(FirstActivity.this, SecondActivity.class);

startActivity(i);
```

Implicit intent: These intents do not name a target and the field for the component name is left blank.

- ✓ Implicit Intents are mechanism that lets open anonymous application's components

- ✓ Android will at run time resolve the best class suited to performing the action – Your app will use other app's functionality without knowing exactly which application!
- ✓ Various native apps provide components that can be called implicitly
- ✓ Implicit Intent's Pieces

Action	The general action to be performed, for example ACTION_DIAL, ACTION_VIEW
Data	The data to operate on expressed in Uri Example: - ACTION_VIEW, content://contacts/people/1 - ACTION_VIEW, content://contacts/people/ - ACTION_DIAL, tel://123456
Category	The category is an optional part of Intent object and it's a string containing additional information about the kind of component that should handle the intent. The addCategory() method places a category in an Intent object, removeCategory() deletes a category previously added, and getCategories() gets the set of all categories currently in the object
Extras	This will be in key-value pairs for additional information that should be delivered to the component handling the intent. The extras can be set and read using the putExtras() and getExtras() methods respectively.
Flags	These flags are optional part of Intent object and instruct the Android system how to launch an activity, and how to treat it after it's launched etc.

For example –

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);
startActivity(read1);

---
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:123456"));
startActivity(intent);

---
```

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://..");
startActivity(intent)
```

For example, let's assume that you have an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity would send an ACTION_SEND along with appropriate chooser, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

```
Intent email = new Intent(Intent.ACTION_SEND, Uri.parse("mailto:"));

email.putExtra(Intent.EXTRA_EMAIL, recipients);

email.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());

email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());

startActivity(Intent.createChooser(email, "Choose an email client from..."));
```

Lot of Native Android Actions:

All the String constants can be found from Intent class. These are native actions.

Activity	- ACTION_ANSWER (=“android.intent.action.ANSWER”) - ACTION_CALL - ACTION_BUG_REPORT - ACTION_POWER_USAGE_SUMMARY - …
Broadcast	- ACTION_BATTERY_CHANGED - ACTION_BATTERY_LOW - ACTION_BOOT_COMPLETED - ACTION_AIRPLANE_MODE_CHANGED - ACTION_CAMERA_BUTTON -…

- **Data flow using Intent**

Sending the data:

```
Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra(“key”, “value”);
startActivity(intent);
```

Receiving the Information:

```
public class SecondActivity extends Activity {
    @Override
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.main);
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            String value1 = extras.getString("key");
        }
public class GettingResultsBack extends Activity {
    private static final int REQUEST_CODE = 10;
...
    public void clickButton(View v) {
        if (v == settings) {
            Intent intent = new Intent(this, Settings.class);
            startActivityForResult(intent, REQUEST_CODE);
        }
    }

 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```
        if (requestCode == REQUEST_CODE) {
            if (resultCode == RESULT_OK) {
                Bundle bundle = data.getExtras();
                String result = bundle.getString("somevalue");
            }
        }
    }
}
```

Getting results Back:

```
@Override
public void onBackPressed() {
  Intent intent = new Intent();
  number = phoneNumber.getText().toString();
  intent.putExtra("phonenumber", number);
  setResult(RESULT_OK, intent);
  super.onBackPressed();
}
```

- **Intent Filters**

  ✓ How does Android know which application (and component) handles the request?
  ✓ Intent Filters are used to register components as being capable of performing an action on particular kind of data
  -Tell Android that your app can service request from other apps
  ✓ How?
  Use application's manifest file and add inter-filtertag

  Using Intent Filter

- ✓ **Intent action** –Name of the action being serviced. Should be unique, so use Java package naming conventions
- ✓ **Intent type / data** –type of data given to component (MIME type)
- ✓ **Intent category** –Additional info about the component that should handle the action.
  CATEGORY_BROWSABLE, CATEGORY_PREFERENCES...

Implementing Own Intent Filter

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.verma.android.demos”> <uses-sdk android:minSdkVersion="8" />
<uses-permission android:name="android.permission.INTERNET">
</uses-permission>
<application
  android:icon="@drawable/icon"
  android:label="@string/app_name">
  <activity
            android:name=".PlaySound"
            android:label="@string/app_name">
  <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
   <intent-filter>
      <action android:name="com.verma.android.demos.PLAYSOUND" />
      <data android:scheme="http" />
      <category android:name="android.intent.category.DEFAULT" />
   </intent-filter>
  </activity>
</application>
 </manifest>
```

Implementing Own Intent Filter

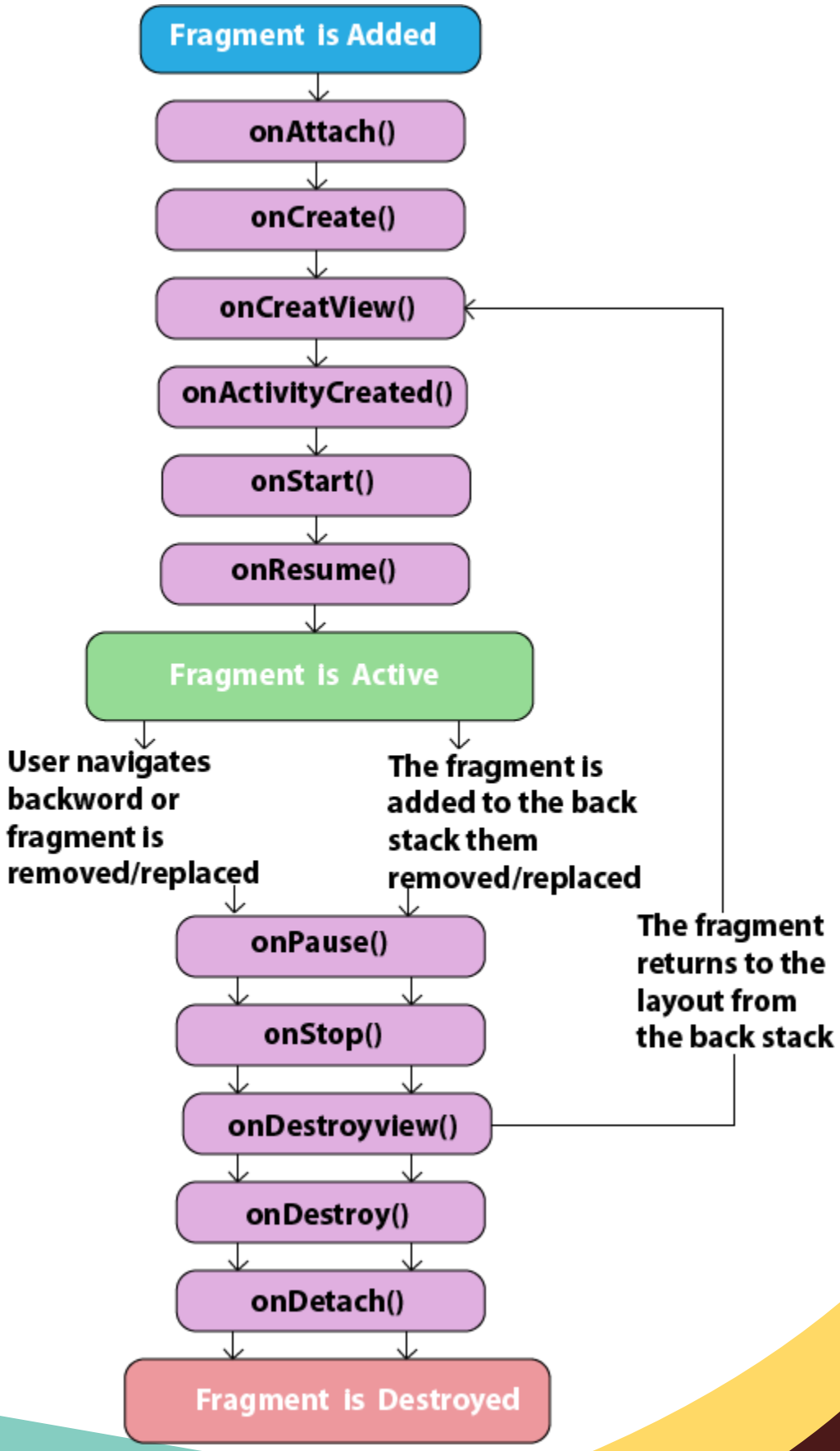
```
 Intent intent = new Intent("com.verma.android.demos.PLAYSOUND", Uri. parse
("http:/ /www….fi/ music.mp3"));

startActivity(intent);
```

```
public class PlaySound extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Intent i = getIntent();
        Uri uri = i.getData();
        if (uri != null) {
            MediaPlayer mp = MediaPlayer.create(this, i.getData());
            mp.start();
        }
    }
}
```

- **Android Fragments**

  - ✓ Android Fragment is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity.
  - ✓ Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.
  - ✓ Each fragment has its own life cycle methods that is affected by activity life cycle because fragments are embedded in activity.
  - ✓ The FragmentManager class is responsible to make interaction between fragment objects.


Fragment is Added
onAttach()
onCreate()
onCreatView()
onActivityCreated()
onStart()
onResume()
Fragment is Active
User navigates backword or fragment is removed/replaced
The fragment is added to the back stack them removed/replaced
onPause()
onStop()
onDestroyview()
onDestroy()
onDetach()
Fragment is Destroyed
The fragment returns to the layout from the back stack

- Android Fragment Lifecycle Methods

No.	Method	Description
1	onAttach(Activity)	it is called only once when it is attached with activity.
2	onCreate(Bundle)	It is used to initialize the fragment.
3	onCreateView(LayoutInflater, ViewGroup, Bundle)	creates and returns view hierarchy.
4	onActivityCreated(Bundle)	It is invoked after the completion of onCreate() method.
5	onViewStateRestored(Bundle)	It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6	onStart()	makes the fragment visible.
7	onResume()	makes the fragment interactive.
8	onPause()	is called when fragment is no longer interactive.
9	onStop()	is called when fragment is no longer visible.
10	onDestroyView()	allows the fragment to clean up resources.
11	onDestroy()	allows the fragment to do final clean up of fragment state.
12	onDetach()	It is called immediately prior to the fragment no longer being associated with its activity.

DAY2

- **SERVICE**

➢ Service is NOT a separate process or thread
➢ Provides two features
  ✓ Tell the system, that we want to do something in the background. (startService())
  ✓ Even if the app closes!
➢ The ability to expose functionality to other apps (bindService())
➢ Service is a simple class, you must implement separate threads by yourself.

- **Started and Bounded Service**

➢ **Started Service**
  ✓ Service is started when an app component calls startService()
  ✓ Service can run in background forever. Usually started service performs a single operation and does not return to caller
  ✓ When operation is done, the service should stop itself
➢ **Bounded Service**
  ✓ Service is bounded when app component binds to it by calling bindService()
  ✓ Bound service may interact with the service (not just start and stop).
  ✓ Bound Service is run only as long as app is bound to the service
➢ Service can be both, run indefinitely and allow bounding.

```
public class MyActivity extends Activity {
   private void startMyService() {
      if (isPreAndroidO()) {
         Log.d(TAG, "Running on Android N or lower");
         startService(intent);
      } else {
         Log.d(TAG, "Running on Android O");
         startForegroundService(intent);
      }
   }

   private void stopMyService() {
```

```
        stopService(intent);
    }
}
```

- **IntentService**

- IntentService has separate thread built in!
- Class inherites IntentService, overrides onHandleIntent()
- Everything is done on separate thread
- Cannot directly interact with UI

```
public class PullService extends IntentService {
        @Override
        protected void onHandleIntent(Intent workIntent){
        // Gets data from the incoming Intent
        String dataString = workIntent.getDataString();
        ...
        // Do work here, based on the contents of dataString ...
        }
}
```

- **About Bound Service**

- Create bound service if you want to
  - ✓ interact with the service from activities or other components
  - ✓ Expose some of app's functionality to other apps (IPC)

- To create bound service, implement onBind()

- A bound service typically lives only while it serves another application component and does not run in the background indefinitely.

- Creating a Bound Service
  - ✓ Private Service for your own app
    - Extend Binder class
    - Example: music application that needs to bind an activity to its own service that's playing music in the background.

- ✓ Service for other apps (work across processes)
  - Use a Messenger

- Extending Binder
  - ✓ If your service is private to your app, you can use Binder
  - ✓ Binder? Defines programming interface that clients can use
  - ✓ Binder can
    - Contain public method that the client can call
    - Return the Service object itself, so all the public methods from the service is available

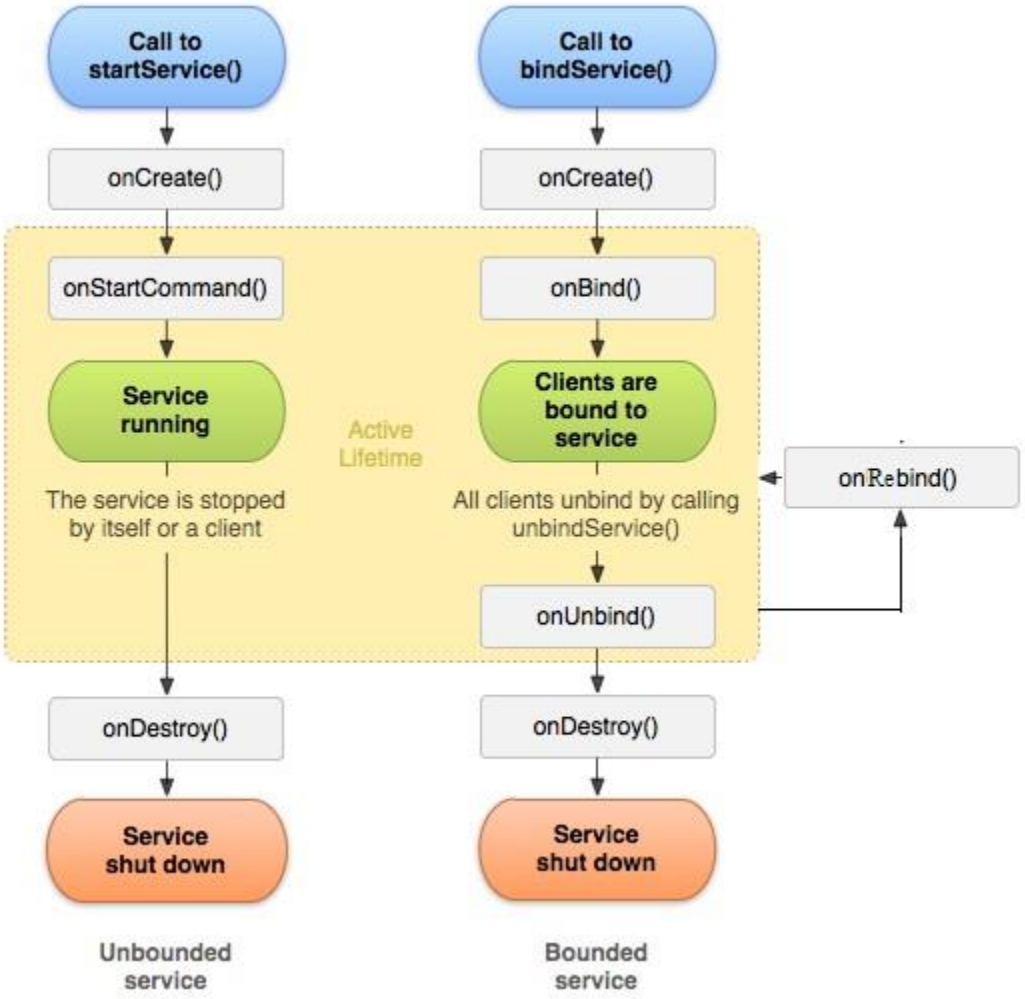


Figure 22. Service Lifecycle

- How?
  - ✓ **Service**
    - Create the Binder object in your Service
    - Return the object in onBind() method
  - ✓ **Client**
    - Receive Binder object from onServiceConnected() method

BROADCAST RECIVER

- A broadcast receiver is a component that does nothing but receive and react to broadcast announcements
- Your app can
  - ✓ Receive and react to system services (example: battery low)
  - ✓ Receive and react to other apps broadcast announcements
  - ✓ Initiate broadcasts to other apps
- App is given fixed amount of time to react on the broadcast! (10 secs!)

- **Registering:**
  - ✓ To register Broadcast Receiver, you can
    - Dynamically register with `registerReceiver` (in code)
    - Statically public receiver and <register> tag in AndroidManifest.xml
  - ✓ Note: if you are doing this dynamically and you are working in local, you don't have modify manifest -file.

- **Registering in Code**

```
public class MyBroadCastReceiver extends BroadcastReceiver {
        @Override
        public void onReceive (Context context, Intent intent) {
           Log.d("IncomingReceiver", "Time Tick");
        }
}

public class BroadCastReceiverActivity extends Activity {
        private MyBroadCastReceiver s;
        private IntentFilter filter;
        @Override
        public void onResume() {
            super.onResume();
```

```
            filter = new IntentFilter("android.intent.action.TIME_TICK");
             s = new MyBroadCastReceiver();
             registerReceiver(s, filter);
        }
        @Override public void onPause() {
            unregisterReceiver(s);
            super.onPause();
         }
}
```

- **Normal vs. Ordered Broadcast**

➢ Normal Broadcasts: Sent with sendBroadcast. All broadcasts are run in undefined order, often at the same time.
➢ Ordered Broadcasts: Sent with sendOrderedBroadcast. Each receiver executes in turn. Possible to propagate a result to next receiver. Order can be controlled using android: priority tag.

- **CONTENT PROVIDERS**

- A content provider makes a specific set of the application's data available to other applications
- => Share data to other apps
- Any app with appropriate permission, can read and write the data.
- Many native databases are available via the content providers,
  For example Contact Manager
- Files, SQL database
- Common interface for querying the data

- **About Content Provides**

- The result of the query: simple table in Query object
- Content provider exposes a public URI that uniquely identifies its data set
  - ✓ URIs begin with content://
  - ✓ Android provides constants for native content providers, for example: *ContactsContract.Contacts.CONTENT_URI*

- **Querying Native Content Provider**

  You need

- URI: *ContactsContract.Contacts.CONTENT_URI*
- Names of data fields (result comes in table
  ContactsContract.Contacts.DISPLAY_NAME
- Data types of those fields *String*
- Remember to modify the manifest file for permissions!

- Query

```
Cursor cur = managedQuery(
        ContactsContract.Contacts.CONTENT_URI, // What Content Provider
        null,   // Which columns to return (all columns)
        null,   // Which rows to return (all rows)
         null,  // Selection arguments (none)
        null);     // In what order
        if (cur.moveToFirst()) {
```

```
        // Get column DISPLAY_NAME
        int nameColumn = cur.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME);
        do {
        // Get the field values
              String name = cur.getString(nameColumn);
              System.out.println(name);
        } while (cur.moveToNext());
}

Modifying the Manifest
<uses-permission android:name="android.permission.READ_CONTACTS"> </uses-permission>
```

- **Implementing your own Content Provider**

- Set up a system for storing data. For example, SQLite or flat file
- Extend ContentProviderclass
- Declare the Content Provider in manifest

- Extend Content Provider

```
public class MyContentProvider extends ContentProvider
        {
         public static final Uri CONTENT_URI = Uri.parse("content://
com.verma.mobile.android.demo.day2.contact");
        @Override
        public int delete(Uri uri, String selection, String[] selectionArgs) {...}
        @Override
        public String getType(Uri uri) {...}
        @Override
```

```
        public Uri insert(Uri uri, ContentValues values) {...}

        @Override

        public boolean onCreate() {...}

        @Override

        public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs,
String sortOrder) {...}

        @Override

        public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)
{...}
}
```

- **ASYNCTASK**

Android application runs on a single thread when launched. Due to this single thread model tasks that take longer time to fetch the response can make the application non-responsive. To avoid this we use android AsyncTask to perform the heavy tasks in background on a dedicated thread and passing the results back to the UI thread. Hence use of AsyncTask in android application keeps the UI thread responsive at all times.

The basic methods used in an android AsyncTask class are defined below:

- **doInBackground()** : This method contains the code which needs to be executed in background. In this method we can send results multiple times to the UI thread by publishProgress() method. To notify that the background processing has been completed we just need to use the return statements
- **onPreExecute()** : This method contains the code which is executed before the background processing starts
- **onPostExecute()** : This method is called after doInBackground method completes processing. Result from doInBackground is passed to this method
- **onProgressUpdate()** : This method receives progress updates from doInBackground method, which is published via publishProgress method, and this method can use this progress update to update the UI thread

The three generic types used in an android AsyncTask class are given below :

- **Params** : The type of the parameters sent to the task upon execution
- **Progress** : The type of the progress units published during the background computation
- **Result** : The type of the result of the background computation

```
public class AsyncTaskTestActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...

        new MyTask().execute("my string paramater");
    }


    private class MyTask extends AsyncTask<String, Integer, String> {

        @Override
        protected void onPreExecute() {

        }

        @Override
        protected String doInBackground(String... params) {

            String myString = params[0];

            int i = 0;
            publishProgress(i);

            return "some string";
        }

        @Override
        protected void onProgressUpdate(Integer... values) {

        }

        @Override
        protected void onPostExecute(String result) {
            super.onPostExecute(result);

        }
    }
}
```

DAY3

- **Notifications**

A **notification** is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

- **Create and Send Notifications**

- Step 1 - Create Notification Builder
- Step 2 -Setting Notification Properties
- Step 3 - Attach Actions
- Step 4 - Issue the notification

```
STEP 1

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)


STEP 2

mBuilder.setSmallIcon(R.drawable.notification_icon);

mBuilder.setContentTitle("Notification Alert, Click Me!");

mBuilder.setContentText("Hi, This is Android Notification Detail!");


STEP 3

Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class);

// Adds the Intent that starts the Activity to the top of the stack
```

```
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);


STEP 4

NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);


// notificationID allows you to update the notification later on.

mNotificationManager.notify(notificationID, mBuilder.build());
```

- **Alert dialogs**

A Dialog is small window that prompts the user to a decision or enter additional information.

List dialog: It has used to show list of items in a dialog box.For suppose, user need to select a list of items or else need to click a item from multiple list of items.At this situation we can use list dialog.

Single-choice list dialog: It has used to add single choice list to Dialog box. We can check or uncheck as per user choice.

- **About 3rd party libraries (Volley, Retrofit etc),**

Retrofit and volly both are the REST client libraries

Volley-: Volley is a networking library it offers great features like synchronous requests, asynchronous requests, prioritization, making multiple requests at the same time, ordered requests and of course caching

Retrofit -: Retrofit is a REST client for Android, through which you can make easy to use interfaces which can turn any Android app into a powerful one. Retrofit can perform Async and sync requests with automatic JSON parsing without any effort

- **About Push Notification**

A push notification is a message that pops up on a mobile device. App publishers can send them at any time; users don't have to be in the app or using their devices to receive them. ... Each mobile platform has support for push notifications - iOS, Android, Fire OS, Windows and BlackBerry all have their own services.

In recent times, Google moved from Google Cloud Messaging (GCM) to Firebase Cloud Messaging (FCM). Just like GCM, FCM is a cross-platform messaging solution that allows you to send messages. FCM is completely free and there are no limitations.

Do your Self

Application 1.

1. Activity Creation – display Name and list of Branch.
2. Handling click listeners.
3. On Boot completed start the activity
4. Creating and storing database using SQLite (SQLite helper class). ie Save information into Database.
5. Display the information in list.
6. Creating services calling and stopping service (for display the current time in UI).

Application 2:

View the first application data

Launching App into Market space

Useful Links

http://developer.android.com

https://java.com/en/download/manual.jsp

https://developer.android.com/studio/index.html

https://en.wikipedia.org/wiki/Android_version_history

https://developer.android.com/guide/components/activities/tasks-and-back-stack

https://firebase.google.com/docs/cloud-messaging/

THANKS

Sudarshan Vadyar

Sourav Kumar Verma

Parveez Ahamed

Divya N P