## **UNIT-2**

## **2.1 Java**

**2.1.1 Java knowledge**

The Java programming language is one of the glorious tools that make programming Android a breeze compared with programming for other mobile platforms. Whereas other languages insist that you manage memory, deallocate and allocate bytes, and then shift bits around like a game of dominoes, Java’s little buddy, the Java Virtual Machine (JVM), helps take care of that for you. The JVM allows you to focus on writing code to solve a business problem by using a clean, understandable programming language (or to build that next cool first-person shooter game you’ve been dreaming of) instead of focusing on the “plumbing” just to get the screens to show up.

You’re expected to understand the basics of the Java programming language before you write your first Android application.

**2.1.2 Java: Your Android programming language**

Android applications are written in Java — not the full-blown version of Java that’s familiar to developers using Java Platform, Enterprise Edition (J2EE), but a subset of the Java libraries that are specific to Android. This smaller subset of Java excludes classes that aren’t suitable for mobile devices. If you have experience in Java, you should feel right at home developing apps in Android.

Not every class that’s available to Java programmers is available also on Android. Verify that it’s available to you before you start trying to use it. If it’s not, an alternative is probably bundled with Android that can work for your needs.

## **2.2 Android Studio**

Android Studio is the official IDE (integrated development environment) for developing Android Apps by Google. It is based on JetBrains’ IntelliJ IDEA software and has lots of amazing features which helps developer in creating Android App. Android Studio is available for free download on Windows, Mac OS X and Linux.

**2.2.1 System Requirement** – First your system OS must be either Windows, Max OS X or Linux with below requirement:

* Microsoft Windows 10/8.1/8/7/Vista/2003/XP (32 or 64 bit)
* Mac OS X 10.8.5 or higher, up to 10.10 to up 10.10.2 up 10.10.3 on 10.10.5 (Yosemite)
* GNOME or KDE or Unity desktop on Ubuntu or Fedora or GNU/Linux Debian
* Minimum RAM: 2GB
* Recommended RAM: 4GB
* Disk Space: 500 MB disk space
* Android SDK Space Requirement: At least 1 GB for Android SDK, emulator system images, and caches
* JDK: Java Development Kit (JDK) 7 or higher
* Screen Resolution: 1280×800 minimum screen resolution
* Prefer faster processor according to your budget
  + 1. [**Android Studio**](https://abhiandroid.com/androidstudio/how-to-download-android-studio) **installation**

**The second thing you need is to**[**download Android Studio**](https://abhiandroid.com/androidstudio/how-to-download-android-studio)**on your system and install it. It is available for free download on Windows, Mac OS X and Linux OS.**

#### **Android Studio – Step by Step procedure:**

* [Start New Project](https://abhiandroid.com/androidstudio/start-create-project) – Learn how to start or [create a new project](https://abhiandroid.com/androidstudio/start-create-project) in Android Studio
* [Open Project](https://abhiandroid.com/androidstudio/open-project) – Learn how to open projects and recent project
* [Reopen, Close & Save Project](https://abhiandroid.com/androidstudio/reopen-close-save-project) – Learn more about how to open saved projects and close current project in Android Studio.
* [Create New Activity](https://abhiandroid.com/androidstudio/create-new-activity-android-studio)– Learn to create New Activity in Android Studio further create XML file for designing UI and java file coding.
* [Create New Java Class](https://abhiandroid.com/androidstudio/how-to-create-new-java-class-in-android-studio) -Learn to create New Java Class in Android Studio.
* [Create Virtual Device](https://abhiandroid.com/androidstudio/create-avd-virtual-device-emulator-android-studio) – Learn more to create a new AVD (virtual device) in Emulator.
* [Run App In AVD](https://abhiandroid.com/androidstudio/run-app-avd-emulator-android-studio) – Learn how to run and test Android App in AVD of Emulator.
* [Run/Test App in Real Device](https://abhiandroid.com/androidstudio/run-android-app-real-device) – Click to know more about how to run app in real device.
* [Create Drawable Resource XML File](https://abhiandroid.com/androidstudio/how-to-create-drawable-resource-xml-file-in-android-studio) – Learn how to create drawable resource XML file in Android Studio.
* [Add/Create Landscape Layout](https://abhiandroid.com/androidstudio/add-create-landscape-layout-android-studio.html) – Learn to design app in Landscape orientation in Android Studio.
* [Create Local HTML File](https://abhiandroid.com/androidstudio/add-local-html-file-android-studio.html) – Learn how to create local [HTML](https://abhiandroid.com/ui/html) file in Android Studio.
* [Create Raw Folder](https://abhiandroid.com/androidstudio/create-raw-folder-android-studio.html) – Click & get more about how to create Raw Folder in Android Studio.
* [Add/Create Assets Folder](https://abhiandroid.com/androidstudio/create-assets-folder-android-studio-html-files.html) – Learn more to create Assets folder in Android Studio.
* [Install Genymotion Emulator](https://abhiandroid.com/androidstudio/install-genymotion-emulator-add-plugin-android-studio.html) – Learn the steps to install Genymotion Emulator in Android Studio.
* [Import/Add External JAR File](https://abhiandroid.com/androidstudio/import-add-external-jar-files-android-studio.html) – Learn how to import an External JAR file to Android Studio.
* [Change API SDK Level](https://abhiandroid.com/androidstudio/change-api-sdk-level-android-studio.html) – Learn how to change the SDK level of API in Android Studio.
* [Create/Add New Package Inside Src Folder](https://abhiandroid.com/androidstudio/add-new-package-inside-src-folder.html) – Learn to know about creating new package inside Src Folder in Android Studio.
* [Creating Folders for Adding Different Resolution Images](https://abhiandroid.com/androidstudio/create-different-folders-adding-different-resolution-images.html) – Learn how add folder for adding images with different resolution in Android Studio.
* [Create An Interface](https://abhiandroid.com/androidstudio/create-interface-android-studio.html) – Learn how to create Interface in Android Studio.
* [Add Image to Drawable Folder in Android Studio](https://abhiandroid.com/androidstudio/add-image-android-studio.html) – Learn Steps to add image to Drawable Folder in Android Studio.
* [Change Icon Of Your Android App](https://abhiandroid.com/androidstudio/change-icon-android-studio.html) – Learn and Follow steps to change the android app icon in Android Studio.
* [Add Audio To Android App](https://abhiandroid.com/androidstudio/add-audio-android-studio.html) – Follow steps defined to add audio to your android application.
* [Application Launcher Icon Size](https://abhiandroid.com/androidstudio/application-launcher-icon-size-android-studio.html) – Here you can learn about different application launcher icon sizes in android studio.
* [Basic Activity In Android Studio](https://abhiandroid.com/androidstudio/create-basic-activity.html) – Here you can learn about how to create a basic activity in android studio.
* [Implement Abstract Method](https://abhiandroid.com/androidstudio/implement-abstract-method.html) – Here you can learn about how to implement abstract methods on clicks or using shortcuts in android studio.
* [Change Package Name In Android Studio](https://abhiandroid.com/androidstudio/how-to-change-package-name-android-studio.html) – Here in this tutorial I am going to discuss step by step how to rename/change package name in Android Studio
* [Generate Signed Apk In Android Studio For Publishing & Updating App](https://abhiandroid.com/androidstudio/generate-signed-apk-android-studio.html) – If you have completed developing your Android App and now wants to publish it on Playstore.

**2.3 Eclipse**

**2.3.1 What is Eclipse?**

■ Eclipse is a universal platform for integrating development tools for integrating development tools.

■ Open, extensible architecture based on plug extensible architecture based on plug-ins

* Provide open platform for application development Provide open platform for application development tools
* – Run on a wide range of de range of operating systems operating systems
* – GUI and non GUI and non-GUI

## ■ Language Language-neutral neutral

## – Permit unrestricted content types Permit unrestricted content types

## – HTML, Java, C, JSP, EJB, XML, GIF, …

## ■ Facilitate seamless tool integration Facilitate seamless tool integration

## – At UI and deeper At UI and deeper

## – Add new tools to existing installed products Add new tools to existing installed products

## ■ Attract community of tool developers Attract community of tool developers

## – Includi Including independent software vendors (ISVs) g independent software vendors (ISVs)

## – Capitalize on popula Capitalize on popularity of Java for writing tools Eclipse created by OT Eclipse created by OTI and IBM teams responsible I and IBM teams responsible for IDE products for IDE products

– IBM VisualAge VisualAge/Smalltalk (Smalltalk IDE) lltalk (Smalltalk IDE)

– IBM VisualAge VisualAge/Java (Jav /Java (Java IDE)

– IBM VisualAge VisualAge/Micro Edition (J Edition (Java IDE)

■ Initially staffed with 40 full Initially staffed with 40 full-time developers time developers

■ Geographically dispersed development teams Geographically dispersed development teams

– OTI Ottawa, OTI Minneapolis, OTI Zurich, IBM OTI Ottawa, OTI Minneapolis, OTI Zurich, IBM Toronto, OTI Ralei Toronto, OTI Raleigh, IBM RTP, IBM St. h, IBM RTP, IBM St. Nazaire Nazaire (France) (France)

■ Effort transitioned into open source project Effort transitioned into open source project

– IBM don IBM donated initial Eclipse code base ted initial Eclipse code base

• Platform, JDT, PDE

**3.3.2 Brief History of Eclipse**

1999:April - Work begins on Eclipse inside OTI/IBM

2000:June - Eclipse Tech Preview ships Eclipse Tech Preview ships

2001:March - http://www.eclipsecorner.org/ /www.eclipsecorner.org/ opens

June - Eclipse 0.9 ships Eclipse 0.9 ships

October October - Eclipse 1.0 ships Eclipse 1.0 ships

November November - IBM donates Eclipse source base IBM donates Eclipse source base

- eclipse.org board announced eclipse.org board announced

- http://www.eclipse.org/ /www.eclipse.org/ opens

2002: June - Eclipse 2.0 ships Eclipse 2.0 ships

September September - Eclipse 2.0.1 ships Eclipse 2.0.1 ships

November November - Eclipse 2.0.2 ships Eclipse 2.0.2 ships

2003:March - Eclipse 2.1 ships

**3.3.3 How to setup Android for Eclipse IDE**

Software’s are required for running an android application on eclipse IDE. Here, you will be able to learn how to install the android SDK and ADT plug-in for Eclipse IDE. Let's see the list of software required to setup android for eclipse IDE manually.

1. Install the JDK
2. Download and install the Eclipse for developing android application
3. Download and Install the android SDK
4. Install the ADT plug-in for eclipse
5. Configure the ADT plug-in
6. Create the AVD

7.Create the hello android application

* 1. **Install the Java Development Kit (JDK)**

For creating android application, JDK must be installed if you are developing the android application with Java language. [download the JDK](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

## **2) Download and install the Eclipse IDE**

For developing the android application using eclipse IDE, you need to install the Eclipse. you can download it from this location [download the Eclipse](http://www.eclipse.org/downloads/). Eclipse classic version is recommended but we are using the Eclipse IDE for JavaEE Developers.

## **3) Download and install the android SDK**

First of all, [download the android SDK](http://developer.android.com/sdk/index.html). In this example we have installed the android SDK for windows (.exe version).

Now double click on the exe file, it will be installed. I am using the android 2.2 version here.

## **4) Download the ADT plugin for eclipse**

ADT (Android Development Tools) is required for developing the android application in the eclipse IDE. It is the plugin for Eclipse IDE that is designed to provide the integrated environment.

## **5) Configuring the ADT plugin**

After the installing ADT plugin, now tell the eclipse IDE for your android SDK location. To do so:

1. Select the **Window menu > preferences**
2. Now select the android from the left panel. Here you may see a dialog box asking if you want to send the statistics to the google.
3. Click on the browse button and locate your SDK directory e.g. my SDK location is C:\Program Files\Android\android-sdk .
4. Click the apply button then OK.

## **6) Create an Android Virtual Device (AVD)**

For running the android application in the Android Emulator, you need to create and AVD. For creating the AVD:

1. Select the **Window menu > AVD Manager**
2. Click on the **new** button, to create the AVD
3. Now a dialog appears, write the AVD name e.g. myavd. Now choose the target android version e.g. android2.2.
4. click the **create AVD**

## **7) Create and run the simple android example**

How to make android apps

To create the simple hello android application. We are creating the simple example of android using the Eclipse IDE. For creating the simple example:

1. Create the new android project
2. Write the message (optional)
3. Run the android application

**3.4 Virtualization**

This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple virtual resources; it can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource...”

**3.4.1 Types of Virtualization**

Today the term virtualization is widely applied to a number of concepts including:

* Server Virtualization
* Client / Desktop / Application Virtualization
* Network Virtualization
* Storage Virtualization
* Service / Application Infrastructure Virtualization

In most of these cases, either virtualizing one physical resource into many virtual resources or turning many physical resources into one virtual resource is occurring.

**3.4.2 Defining API Virtualization**

API virtualization is the process of using a tool that creates a virtual copy of your API, which mirrors all of the specifications of your production API, and using this virtual copy in place of your production API for testing.

Instead of setting up a whole separate server stack to mimic production, API virtualization aims to simulate the minimum behaviors of one or more API endpoints.

To illustrate, API virtualization is the equivalent of allowing you (or, in this case, your testing team) to taste a cake – its flavor, texture, and all – before it has finished baking.

With API virtualization, your development teams can create virtual APIs instead of production APIs, enabling frequent and comprehensive testing even when the API is still in the midst of being developed.

By emulating behaviors and specifications that will be present in the final production API, virtualization allows for testing much earlier in the development process, removing key bottlenecks that would otherwise delay production and time-to-market. More and more companies are using virtualization to improve productivity, reduce testing costs, and deploy higher-quality APIs in a shorter timeframe.

By quickly and easily removing dependency constraints across your organization through virtualization, you can gain a competitive advantage over other companies still waiting in the linear-development limbo.

The Android SDK includes a virtual mobile device emulator that runs on your computer. The emulator lets you prototype, develop and test Android applications without using a physical device.

In this we are going to explore different functionalities in the emulator that are present in the real android device.

## **3.4.3 Creating AVD**

If you want to emulate a real device, first crate an AVD with the same device configurations as real device, then launch this AVD from AVD manager.

## **Changing Orientation**

Usually by default when you launch the emulator, its orientation is vertical, but you can change it orientation by pressing Ctrl+F11 key from keyboard.

First launch the emulator. It is shown in the picture below −



Once it is launched, press **Ctrl+F11** key to change its orientation. It is shown below −



**3.4.4 Emulator Commands.**

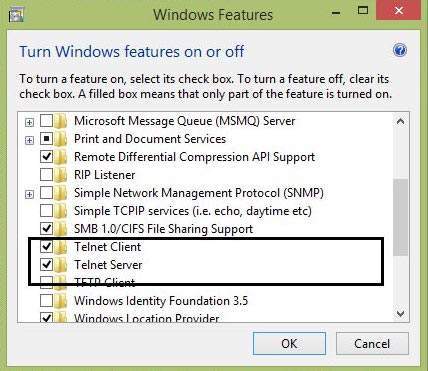
Apart from just orientation commands, there are other very useful commands of emulator that you should keep in mind while using emulator. They are listed below −

|  |  |
| --- | --- |
| **Sr.No** | **Command & description** |
| 1 | **Home**  Shifts to main screen |
| 2 | **F2**  Toggles context sensitive menu |
| 3 | **F3**  Bring out call log |
| 4 | **F4**  End call |
| 5 | **F5**  Search |
| 6 | **F6**  Toggle trackball mode |
| 7 | **F7**  Power button |
| 8 | **F8**  Toggle data network |
| 9 | **Ctrl+F5**  Ring Volume up |
| 10 | **Ctrl+F6**  Ring Volume down |

## **3.4.5 Emulator - Sending SMS**

You can emulate sending SMS to your emulator. There are two ways to do that. You can do that from DDMS which can be found in Android studio, or from Telnet.(Network utility found in windows).

### Sending SMS through Telnet.

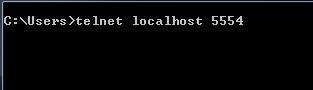


Telnet is not enabled by default in windows. You have to enable it to use it. Once enabled you can go to command prompt and start telnet by typing telnet.

In order to send SMS , note down the AVD number which can be found on the title bar of the emulator. It could be like this 5554 e.t.c. Once noted , type this command in command prompt.

telnet localhost 5554

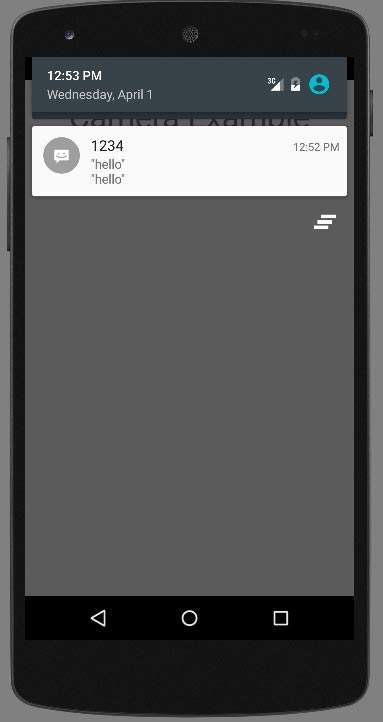
Press enter when you type the command. It is shown below in the figure.



You will see that you are now connected to your emulator. Now type this command to send message.

sms send 1234 "hello"

Once you type this command , hit enter. Now look at the AVD. You will receive a notification displaying that you got a new text message. It is shown below −



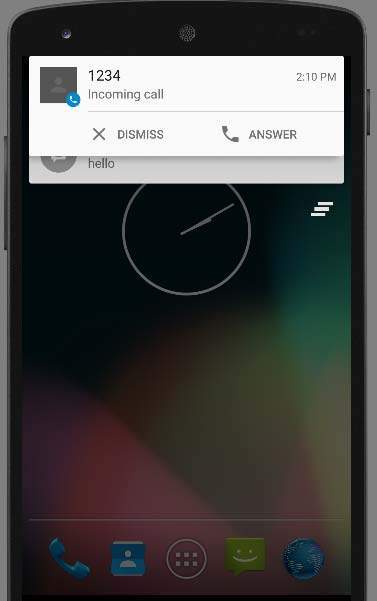
## **Emulator - Making Call**

You can easily make phone calls to your emulator using telent client. You need to connect to your emulator from telnet. It is discussed in the sending sms topic above.

After that you will type this command in the telent window to make a call. Its syntax is given below −

gsm call 1234

Once you type this command , hit enter. Now look at the AVD. You will receive a call from the number your put in the command. It is shown below −



## **Emulator - Transferring files**

You can easily transfer files into the emulator and vice versa. In order to do that, you need to select the DDMS utility in Android studio. After that select the file explorer tab. It is shown below −



Browse through the explorer and make new folder , view existing contents e.t.c.

**2.5 APIs and Android tools**

Use the [Android Management API](https://developers.google.com/android/management/) to integrate support for Android device and app management into your EMM console. The API and its companion DPC app, Android Device Policy, work together as a self-contained solution. For more information, see [Development options](https://developers.google.com/android/work/dev-options).

### Google Play EMM API

You can use the [Google Play EMM API](https://developers.google.com/android/work/play/emm-api/) to integrate support for the following tasks into your EMM console:

* Specify apps that users are allowed to download onto managed devices.
* Host app APKs outside of Google Play. (Google Play hosts only the metadata for these APKs.)
* Manage user licensees in bulk for paid apps.
* Manage app installation.

The Play EMM API doesn't include device management features. To enforce management policies on devices, you need to develop your own DPC. For more information, see [Develop a solution](https://developers.google.com/android/work/dev-options).

**2.5.2 DPC development**

For guidance on how to create a device policy controller (DPC) app, see [Build a DPC](https://developer.android.com/work/dpc/build-dpc).

A **sample DPC app** called [**Test DPC**](https://play.google.com/store/apps/details?id=com.afwsamples.testdpc) is available on Google Play. The code for Test DPC is available as an open source project on [GitHub](https://github.com/googlesamples/android-testdpc).

You can use Test DPC as a sample DPC or as a testing tool. As a testing tool, Test DPC provides an effective way to test applications and platforms in a managed context. For details, see the [Test DPC readme file](https://github.com/googlesamples/android-testdpc/blob/master/README.md). To report an issue, use the [Test DPC issue tracker](https://github.com/googlesamples/android-testdpc/issues).

When using Test DPC, keep the following in mind:

* Building Test DPC requires Android SDK v23, Android Build Tools v23.0.1, and the Android support repository.
* Test DPC supports devices running Android 5.0 Lollipop or later.
* Test DPC uses the Gradle build system.

**2.5.3 Installing and Configuring Your Support Tools**

It’s time to put these exciting Android concepts into action, but before you can do so, you need to install and configure a few tools, including the software development kits (SDKs):

✓ Java JDK: Lays the foundation for the Android SDK.

✓ Android SDK: Provides access to Android libraries and allows you to develop for Android. ✓ Eclipse IDE (integrated development environment): Brings together Java, the Android SDK, and the Android Android Development Tools (ADT) and provides tools for you to write Android programs.

✓ Android ADT: Does a lot of the grunt work for you, such as creating the files and structure required for an Android app.

## **2.6 Debugging Applications with DDMS**

The Dalvik Debug Monitor Service (DDMS) is a debugging utility that is integrated into Eclipse through a special Eclipse perspective. The DDMS perspective provides a number of useful features for interacting with emulators and handsets and debugging applications.

 The DDMS perspective, with one emulator and two Android devices connected (the Nexus S running 2.3.1 and the Samsung Galaxy Tablet running

**2.6.1 The features of DDMS are roughly divided into five functional areas:**

* Task management
* File management
* Emulator interaction
* Logging
* Screen captures

DDMS and the DDMS perspective are essential debugging tools. Now let's take a look at how to use these features in a bit more detail.

**2.6.2 Built-in debugging tools**

* Logcat
* Debug
* Lint

**2.6.3 Plugins for developer productivity**

* ADB Idea
* Codota
* Lombok Plugin

### Debugging Android: Debug

Using Logcat to log and correct code is okay for very simple apps. For more complicated apps, this form of debugging can be tedious. Instead you'll want something that lets you debug the app's executable code. Android Studio's built-in Debug tool offers many capabilities, including the following:

* Select a device on which to debug your app.
* Set breakpoints in your application code (Java, Kotlin, or C/C++).
* Examine variables and evaluate expressions at runtime.

**2.6.4 Debugging tool**

1. If your app includes C/C++ source code, you'll need to [install LLDB](https://developer.android.com/studio/intro/update#sdk-manager) from the SDK Manager (see Figure 3). Fortunately, the example app for this series (**W2A**) doesn't contain C/C++ code, so we can ignore this prerequisite.
2. You must [enable debugging on a connected device](https://developer.android.com/studio/debug/dev-options). However, if you're using an emulator (which we are for this example), you can ignore this prerequisite. Debugging is enabled by default on emulated devices.
3. You must run a debug gable [build variant](https://developer.android.com/studio/build/build-variants). By default, this is created for you, so in many cases (including this example) you don't have to worry about it.

**2.7 Android File system**

**2.7.1 Flash Memory Android File System**

### ****1. exFAT****

Originally created by Microsoft for flash memory, the exFAT file system is not a part of the standard Linux kernel. However, it still provides support for Android devices in some cases. It stands for Extended File Allocation Table.

### ****2.F2FS****

Users of Samsung smartphones are bound to have come across this type of file system if they have been using the smartphone for a while. F2FS stands for Flash-Friendly File System, which is an Open Source Linux file system. This was introduced by Samsung 4 years ago, in 2012.

### ****3. JFFS2****

It stands for the Journal Flash File System version 2. This is the default flash file system for the Android Open Source Project kernels. This version of Android File System has been around since the Android Ice Cream Sandwich OS  was released. JFFS2 has since replaced the JFFS.

### ****4. YAFFS2****

It stands for Yet Another Flash File System version 2. Funny as the name might sound like, it is actually a serious business! It has not been a part of the AOSP for a while now and is rarely found in Android smartphones. However, it does tend to make a few appearances every now and then.

## The Android OS is a popular and universally used operating system for smart phones. The Android File Systems tend to be rather complicated and have a number of users scratching their head in amusement.

## **2.7.2 Media-based Android File System**

### ****1. EXT2/EXT3/EXT4****

Ext, which stands for the EXTended file systems, are the standards for the Linux file system. The latest out of these is the EXT4, which has now been replacing the YAFFS2 and the JFFS2 file systems on Android smartphones.

### ****2. MSDOS****

Microsoft Disk Operating System is known to be one of the oldest names in the world of Operating Systems, and it helps FAT 12, FAT 16 and FAT 32 file systems to run.

### ****3. VFAT****

An extension to the aforementioned FAT 12, FAT 16 and FAT 32 file systems, the VFAT is a kernel module seen alongside the MSDOS module. External SD cards that help expand the storage space are formatted using VFAT.

## **2.7.3 Pseudo File Systems**

### ****1. CGroup****

Cgroup stands for Control Group. It is a pseudo file system which allows access and meaning to various kernel parameters. Cgroups are very important for the Android File System as the Android OS makes use of these control groups for user accounting and CPU Control.

### ****2. Rootfs****

Rootfs acts as the mount point, and it is a minimal file system. It is located at the mount point “/”.

### ****3. Procfs****

Usually found mounted at the /proc directory. The procfs file system has files which showcase the live kernel data. Sometimes this file system also reflects a number of kernel data structures. These number directories are reflective of the process IDs for all the currently running tasks.

### ****4. Sysfs****

Usually mounted on the /sys directory. The sysfs file system helps the kernel identify the devices. Upon identifying a new device, the kernel builds an object in sys/module/ directory. There are various other elements stored inside the /sys/ folder which helps the kernel communicate with various Android File Systems.

### ****5. Tmpfs****

A temporary file system, tmpfs is usually mounted on /dev directory. Data on this is lost when the device is rebooted.

**2.8 Working with emulator and smart devices**

 To perform mobile testing, you need a mobile device. This is to access that how our product will work and look like on a given mobile set.Suppose we are developing an application for flight ticket booking system. Once the product is entirely developed, as a part of mobile testing, we need to check if the application is working as expected with all the majorly used devices like Android phones, iOS, Blackberry phones, and other different types of tablets and iPads.

To do this kind of check, we need to acquire each such device and then we can check if the application behaves as per expectation. Yes you thought right, as a product owner one will defiantly find this very expensive to procure such a large number of mobile devices and carry out testing. So is there any smart alternate available?

The solution to this problem is to use Mobile Simulators and Mobile Emulators. These are primarily software programs designed to provide simulation for important features of a smartphone. They are very similar in nature, so sometimes, they are used interchangeably.

**2.8.1 Testing on an Emulator/Simulator is different from testing on a real device**

|  |  |  |
| --- | --- | --- |
|  | **Real Device** | **Emulator / Simulator** |
| **Price** | Getting real devices will cost you a lot. | It is almost free, we just need to download and install them |
| **Processing Speed** | It has faster processing; however network latency may be normal. | It is slower as compared to actual devices. It has observed less latency than real devices connected to the local network or in the cloud. |
| **Debugging** | Debugging is not that easy. | It provides step-by-step debugging of an application. Also, it provides an efficient way for capturing screenshots. |
| **Web-app Testing** | Web applications can be tested in a normal way. | Testing a web application is much easier. |
| **Reliability** | Testing on a real device has a major advantage that it always gives accurate results. | It cannot simulate all types of user interactions; hence it may lead to false results sometimes. So it scores low when it comes to reliability. |

A simulator/emulator cannot mimic the following features −

* Mobile device battery
* Mobile device’s camera
* Difficult to mimic interruptions like incoming calls and SMS.
* Not so much realistic simulation for mobile device memory usage.

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.



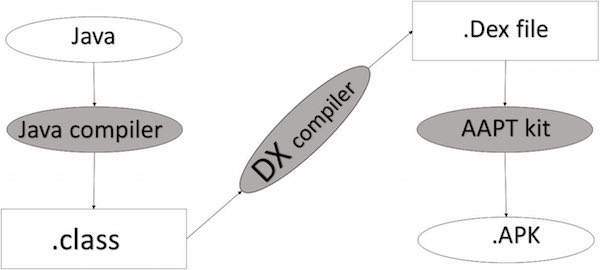
#### ANDROID DEVELOPMENT LIFE CYCLE

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.

You can check a detailed publishing process at Android official website, but this tutorial will take you through simple steps to launch your application on Google Play. Here is a simplified check list which will help you in launching your Android application −

|  |  |
| --- | --- |
| **Step** | **Activity** |
| 1 | **Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets. |
| 2 | **Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity. |
| 3 | **Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting up time zone, localization or any other specific requirement as per the targeted region. |
| 4 | **Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices. |
| 5 | **SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target. |
| 6 | **Application Pricing** Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies. |
| 7 | **Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere. |
| 8 | **Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: [**Preparing for Release**](https://developer.android.com/tools/publishing/preparing.html). |
| 9 | **Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide. |

## **2.8.2 Export Android Application Process**



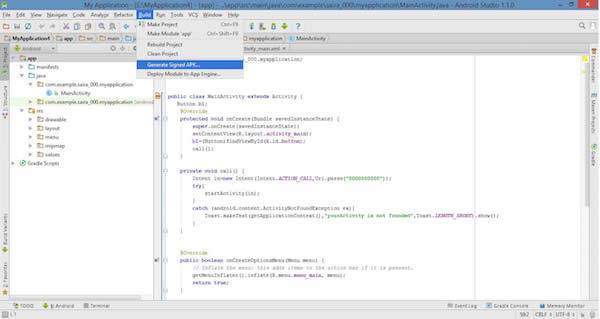
#### APK DEVELOPMENT PROCESS

Before exporting the apps, you must some of tools

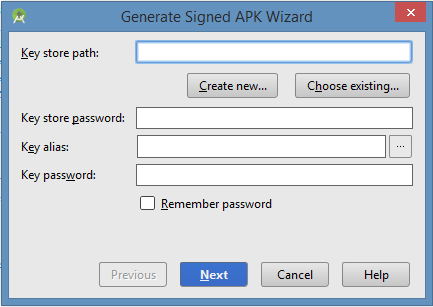
* **Dx tools**(Dalvik executable tools ): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time
* **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file**to**.Apk**
* **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

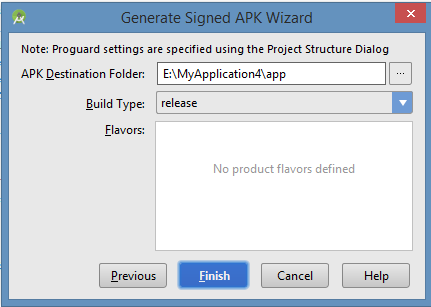
To export an application, just open that application project in Android studio and select **Build → Generate Signed APK** from your Android studio and follow the simple steps to export your application −



Next select, **Generate Signed APK** option as shown in the above screen shot and then click it so that you get following screen where you will choose **Create new keystore** to store your application.



Enter your key store path,key store password,key alias and key password to protect your application and click on **Next** button once again. It will display following screen to let you create an application −



Once you filled up all the information,like app destination,build type and flavours click **finish** button While creating an application it will show as below

Creating Application

Finally, it will generate your Android Application as APK formate File which will be uploaded at Google Play marketplace.

**2.9 A Basic Android Application**

To develop Android applications and, in the process, install the Eclipse Android Development Tools (ADT) plug-in. It gives you the power to generate new Android applications directly from within the Eclipse File menu.

Follow these steps to create your first Android application project:

1. In Eclipse, choose File➪New➪Other. Select Android Application Project.

2. Enter Hello Android as the application name. The application name is the name of the application as it pertains to Android. When the application is installed on the emulator or physical device, this name appears in the application launcher.

The Project and Package names should auto complete for you.

The Project Name field is important. The descriptive name you provide identifies your project in the Eclipse workspace. After your project is created, a folder in the workspace is named with the project name you define.

3. In the Package Name box, type com.dummies.android.helloandroid. This is the name of the Java package. (See the nearby sidebar “Java package nomenclature.”)

4. Select Android 4.1 from the Build SDK drop-down list and API 8: Android 2.2 from the Minimum Required SDK drop-down list, and then click Next.

5. (Optional) Create an application icon for your project and click Next. 6. In the Create Activity box, choose BlankActivity and click Next. The New Blank Activity screen appears.

7. Enter MainActivity in the Activity Name box. The New Blank Activity screen defines what the initial activity is called — the entry point to your application. When Android runs your application, this file is the first one to be accessed. A common naming pattern for the first activity in your application is MainActivity.java. (How creative.)

8. Click the Finish button. You’re done! You should see Eclipse with a single project in the Package Explorer. Different names; same function.

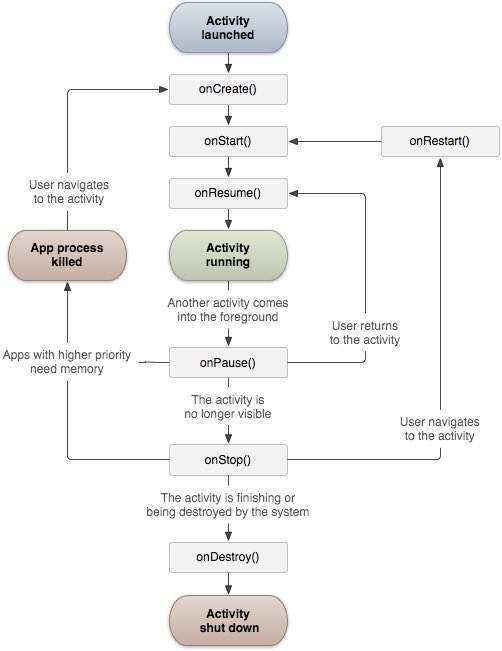
**2.10 Activity**

Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity. as shown in the below Activity life cycle diagram: (*image courtesy : android.com* )

**2.10.1 Activity life cycle**

The rectangles represent callback methods you can implement to respond to events in the activity. The shaded ovals represent the major states of the activity. The activity life cycle is a large and complex topic, and the following sections cover only the basics. If you want to read more about activity life

The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.



**2.10.2 The movement of an activity life cycle:**

|  |  |
| --- | --- |
| **Sr.No** | **Callback & Description** |
| 1 | **onCreate()**  This is the first callback and called when the activity is first created. |
| 2 | **onStart()**  This callback is called when the activity becomes visible to the user. |
| 3 | **onResume()**  This is called when the user starts interacting with the application. |
| 4 | **onPause()**  The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed. |
| 5 | **onStop()**  This callback is called when the activity is no longer visible. |
| 6 | **onDestroy()**  This callback is called before the activity is destroyed by the system. |
| 7 | **onRestart()**  This callback is called when the activity restarts after stopping it. |

Example

This example will take you through simple steps to show Android application activity life cycle. Follow the following steps to modify the Android application we created in *Hello World Example*chapter −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android studio to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter. |
| 2 | Modify main activity file *MainActivity.java* as explained below. Keep rest of the files unchanged. |
| 3 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

**2.10.3 Lifecycle methods**

You may be interested in monitoring these three loops in your activity:

✓ The **entire lifetime** takes place between the first call to onCreate() and the final call to onDestroy(). The activity performs all global setupin onCreate() and releases all remaining resources in onDestroy(). For example, if you create a thread to download a file from the Internet in the background, it may be initialized in the onCreate() method. That thread can be stopped in the onDestroy() method.

✓ The **visible lifetime** of the activity takes place between the onStart() and onStop() methods. During this time, the user can see the activity onscreen (though it may not be in the foreground interacting with the user, which can happen when the user is interacting with a dialog box). Between these two methods, you can maintain the resources that are needed to show and run your activity.

✓ The **foreground lifetime** of the activity begins at the call to onResume() and ends at the call to onPause(). During this time, the activity is in front of all other activities and is interacting with the user. An activity normally toggles between onResume() and onPause() multiple times, for example, when the device goes to sleep or when a new activity handles a particular event — therefore, the code in these methods must be fairly lightweight.

**Viewing activity methods**

The entire activity life cycle boils down to these methods:

public class Activity extends ApplicationContext {

protected void onCreate(Bundle savedInstanceState);

protected void onStart();

protected void onRestart();

protected void onResume();

protected void onPause();

protected void onStop();

protected void onDestroy();

}

All methods can be overridden, and custom code can be placed in all of them. All activities implement onCreate() for initialization and may also implement onPause() for clean-up. You should always call the superclass (base class) when implementing these methods.

**2.10.4 Creating Your First Activity**

You may have already created your first activity if you created a project using the New Android roject Wizard in Chapter 3: the MainActivity activity. Open the MainActivity.java file in your project to enhance it in the following sections.

**Starting with onCreate**

The entry point into your application is the onCreate() method. The code for the MainActivity.java file already contains an implementation of the onCreate() method. It’s where you start writing code! For now, your code should look like this:

public class MainActivity extends Activity {

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

}

}

You write the initialization code directly below the setContentView() method.

Be sure to always include this method call to your onCreate() method:

super.onCreate(savedInstanceState);

It’s required for the application to run. This line directs the base Activity class to perform setup work for the MainActivity class. If you omit this line of code, you receive a runtime exception.

**2.11 Intents**

All Android activities are started or activated with an intent. Intents are message objects that make a request to the Android runtime to start an activity or other app component in your app or in some other app. You don't start those activities yourself.

When your app is first started from the device home screen, the Android runtime sends an intent to your app to start your app's main activity (the one defined with the MAIN action and the LAUNCHER category in the Android Manifest). In addition to starting activities, intents are also used to pass data between activities. When you create an intent to start a new activity, you can include information about the data you want that new activity to operate on.

**2.11.1 Intent types**

There are two types of intents in Android

1. Explicit intents specify the receiving activity (or other component) by that activity's fully-qualified class name. Use an explicit intent to start a component in your own app (for example, to move between screens in the user interface), because you already know the package and class name of that component.

2. Implicit intents do not specify a specific activity or other component to receive the intent. Instead you declare a general action to perform in the intent. The Android system matches your request to an activity or other component that can handle your requested action.

**2.11.2 Intent objects and fields**

An Intent object is an instance of the Intent class. For explicit intents, the key fields of intent include the following:

1. The activity class (for explicit intents). This is the class name of the activity or other component that should receive the intent, for example, com.example.SampleActivity.class. Use the intent constructor or the intent's setComponent(), setComponentName() or setClassName() methods to specify the class.

2. The intent data. The intent data field contains a reference to the data you want the receiving activity to operate on, as a Uri object.

3. Intent extras. These are key-value pairs that carry information the receiving activity requires to accomplish the requested action.

4. Intent flags. These are additional bits of metadata, defined by the Intent class. The flags may instruct the Android system how to launch an activity or how to treat it after it's launched.

**2.11.3 Starting an activity with an explicit intent**

To start a specific activity from another activity, use an explicit intent and the startActivity() method. Explicit intents include the fully-qualified class name for the activity or other component in the Intent object. All the other intent fields are optional, and null by default.

Ex:

**intent message intent=new intent(this, showmessageAvtivity.class);**

**startActivity(messageIntent);**

**The Intent constructor takes two arguments for an explicit intent.**

 An application context. In this example, the activity class provides the content (here, this ).

 The specific component to start ( ShowMessageActivity.class ).

**Passing data between activities with intents**

In addition to simply starting one activity from another, you also use intents to pass information between activities. The intent object you use to start an activity can include intent data (the URI of an object to act on), or intent extras, which are bits of additional data the activity might need. In the first (sending) activity, you

1. Create the Intent object.

2. Put data or extras into that intent.

3. Start the new activity with startActivity(). In the second (receiving) activity, you: 1. Get the intent object the activity was started with. 2. Retrieve the data or extras from the Intent object.

## **2.12 Intent Filters**

Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

Following is an example of a part of **AndroidManifest.xml** file to specify an activity **com.example.My Application.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data −

<activity android:name=".CustomActivity"

android:label="@string/app\_name">

<intent-filter>

<action android:name="android.intent.action.VIEW" />

<action android:name="com.example.My Application.LAUNCH" />

<category android:name="android.intent.category.DEFAULT" />

<data android:scheme="http" />

</intent-filter>

</activity>

Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activity using either the **android.intent.action.VIEW**, or using the **com.example.My Application.LAUNCH** action provided their category is **android.intent.category.DEFAULT**.

The **<data>** element specifies the data type expected by the activity to be called and for above example our custom activity expects the data to start with the "http://"

There may be a situation that an intent can pass through the filters of more than one activity or service, the user may be asked which component to activate. An exception is raised if no target can be found.

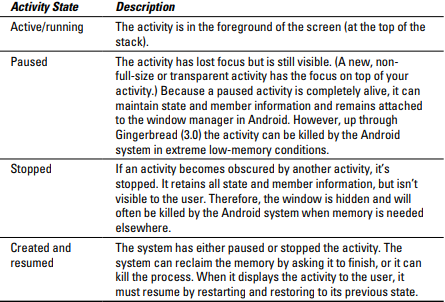
There are following test Android checks before invoking an activity −

* A filter <intent-filter> may list more than one action as shown above but this list cannot be empty; a filter must contain at least one <action> element, otherwise it will block all intents. If more than one action is mentioned then Android tries to match one of the mentioned actions before invoking the activity.
* A filter <intent-filter> may list zero, one or more than one categories. if there is no category mentioned then Android always pass this test but if more than one categories are mentioned then for an intent to pass the category test, every category in the Intent object must match a category in the filter.
* Each <data> element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme, host, port**, and **path** for each part of the URI. An Intent object that contains both a URI and a data type passes the data type part of the test only if its type matches a type listed in the filter.

**2.13 Activity stack**

Activities in the system are managed as an activity stack. When a new activity is created, it’s placed on top of the stack and becomes the running activity. The previous running activity always remains below it in the stack and returns to the foreground only when the new activity exits.

**2.13.1 An activity has essentially four states**

****